

GRASP

GRASP User's Manual

www.ticra.com



TICRA
Pioneering Antennas

TICRA is an antenna engineering company with world leading expertise in software tools and consultancy services within design, analysis and measurement techniques for antennas. Based in the centre of Copenhagen, Denmark, TICRA employs highly skilled professionals with over 240 man-years of expertise in their fields.

GRASP User's Manual

GRASP Version 10.3.0

March 2014

Copyright ©2002-2014

TICRA
LÆDERSTRÆDE 34 · DK-1201 COPENHAGEN K
TELEPHONE +45 33 12 45 72
E-MAIL ticra@ticra.com

DENMARK
TELEFAX +45 33 12 08 80
<http://www.ticra.com>

VAT REGISTRATION NO. DK-1055 8697
TICRA FOND, CVR REG. NO. 1055 8697

TABLE OF CONTENTS

1. Introduction	1
2. SW-Package Overview	2
3. Getting Started with GRASP - the Wizard	4
3.1 The Single Reflector Wizard	5
4. The GRASP Main Window	10
4.1 The Objects Window	12
4.2 The Commands Window	28
4.3 The Jobs Window	31
4.4 The Results Window	33
5. File Organization	43
6. Tutorial Examples	44
6.1 Single Reflector with Three Struts of Polygonal Cross-Section	45
6.2 Dual Reflector with Blockage	61
6.3 Single Shaped Reflector with Circular Polarisation	73
6.4 Dual Gridded Reflector	80
6.5 Single Reflector with Feed Array	97
6.6 Compact Antenna Test Range with Serrated Edges	103
6.7 Plane Wave Expansion	110
6.8 Scalable Dual Reflector Using Real Variables	122
6.9 Offset Reflector in Radome	129
6.10 Dual Reflector with Panels and Subreflector Support Struts	142
7. Batch-Mode Operation	155
8. GRASP 10 for GRASP9 Users	157

8.1	The Objects Window	158
8.2	The Commands Window	159
8.3	The Jobs Window	160
8.4	The Results Window	160
9.	Reference Section	162
9.1	Guide to the Reference Section	163
9.2	Alphabetical List of Classes and Command Types	166
9.3	Classes	176
9.4	Command Types	969
9.5	Applicable Units and the dB-Scale	1033
9.6	File Extensions	1035
9.7	File Formats	1038
	Appendices	1117

1. Introduction

Welcome to the GRASP User's Manual. The present document provides a description of the GRASP software and a series of tutorial examples.

In Chapter 2 a general overview of the GRASP software is first given. In Chapter 3 the user is guided in setting up an antenna geometry with the GRASP wizard. A description of the objects and commands generated by the wizard is given in Chapter 4, together with a presentation of the results obtained by running an analysis of the antenna. The features of the GRASP Graphical User Interface are in this way presented. Chapter 5 describes the files organization, while Chapter 6 contains a series of application examples in which more software features and capabilities are described. Chapter 7 finally explains how to run the software in batch mode. To conclude, Chapter 8 is for all users who are already familiar with GRASP9 and wish to know the main differences with GRASP 10.

A complete and detailed description of all classes, commands and input/output file formats of GRASP is found in the *Reference Section* in Chapter 9.

A full description of the GRASP program with mathematical definitions of the applied terms, explanations of the applied geometries and electromagnetic models as well as output data capabilities is given in the GRASP Technical Description, which may be opened from the Help Menu.

To take full advantage of the present manual, it is recommended you have a version of GRASP running.

2. SW-Package Overview

GRASP is a general software tool for design and analysis of reflector antennas and antenna farms. Single and dual reflector antennas, multi-reflector and multi-feed antennas, as well as gridded and shaped reflector antennas can be set-up and analyzed. Reflector imperfections can be considered by including thermal and mechanical distortions, gaps between reflector panels, supporting struts and scatterer material properties.

The electromagnetic methods which may be applied for the analysis are

- Physical Optics (PO) with Physical Theory of Diffraction (PTD)
- Geometrical Optics (GO) and Geometrical Theory of Diffraction (GTD)
- Spherical Wave Expansion (SWE)
- Plane Wave Expansion (PWE)
- Methods of Moments (MoM), only available as add-on

It is assumed that the user has an engineering understanding of the advantages and limitations of these methods. For more details, please see the GRASP Technical Description and the Manuals of the GRASP add-ons, all under the Help menu.

GRASP consists of two main components: A Graphical User Interface (GUI) and a back-end module.

The GUI assists the user in:

1. Setting up the project, i.e. defining the antenna geometry, manually or by using the wizard
2. Setting up the commands to be executed
3. Launching the computation invoking the back-end module
4. Visualizing the results computed by the back-end module

The back-end module performs the antenna analysis as well as a number of tasks related to project management, i.e. logging, file input/output, 3D drawing, and license management.

In the installation directory, the GUI is the "grasp.exe" and the back-end module is the "grasp-analysis.exe" executables. To launch GRASP, click on the GRASP-Icon or navigate to the directory in which GRASP is installed and

double-click on the "grasp.exe" program. This will open the GUI. The analysis module can be used for batch-mode operation without the GUI, as explained in Chapter 7. We will assume in this document that GRASP is launched with the GUI.

3. Getting Started with GRASP - the Wizard

When GRASP is launched (or a new project is requested from the file menu), the main window shown in Figure 3-1 will appear. The user has here the possibility of using the wizard, opening a blank project, or opening an existing project.

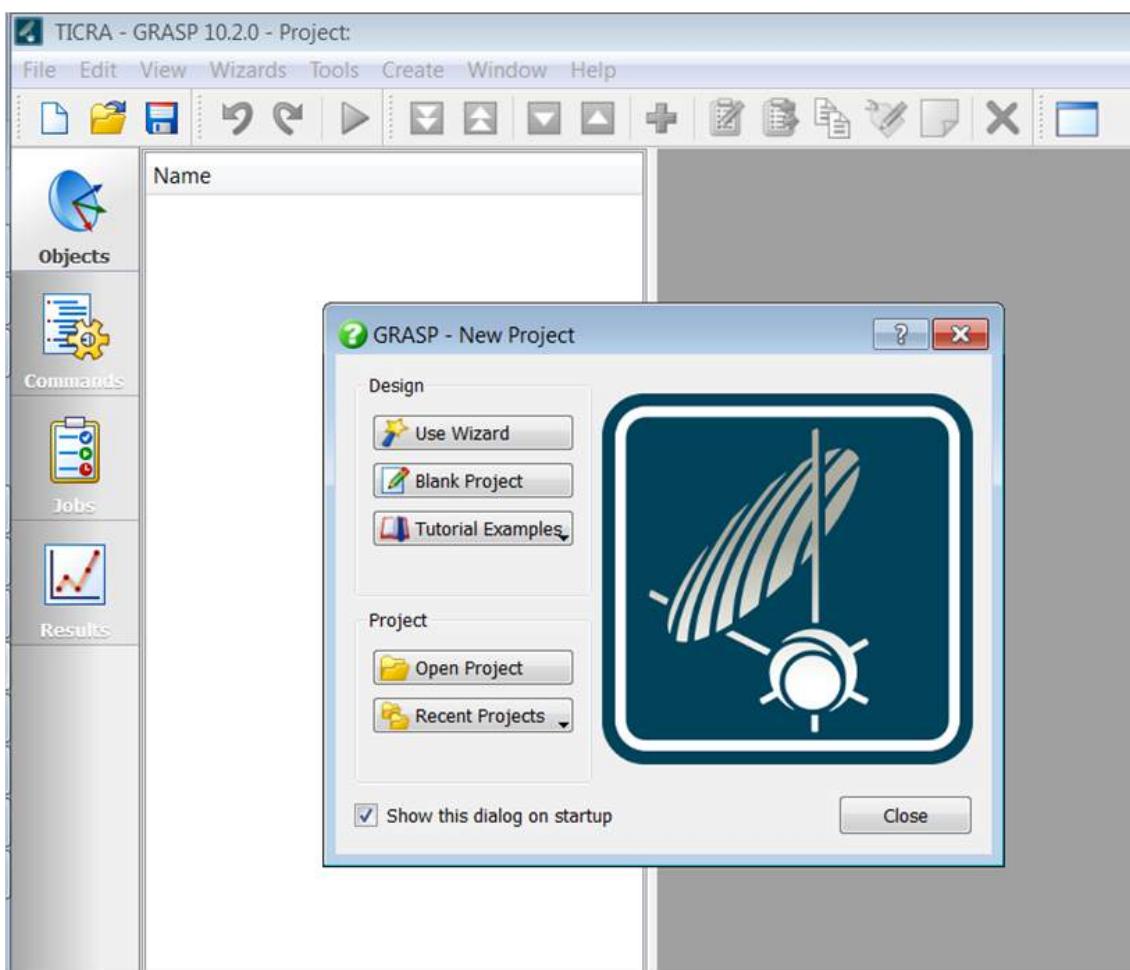


Figure 3-1 The GRASP opening window.

The data defining a GRASP project are kept in entities denoted objects. The data in an object are denoted the attributes of the object.

We will in this chapter explain how to design a single reflector using the wizard. If the wizard is skipped, the user needs to define all the necessary objects and commands in the OBJECTS window and COMMANDS window with the corresponding editors. More details can be found in Sections 4.1 and 4.2.

For a list and explanation of all possible objects and commands please refer to the *Reference Section* in Chapter 9. For more examples, please see Chapter 6.

3.1 The Single Reflector Wizard

The antenna we want to set-up with the wizard is shown in Figure 3-2:

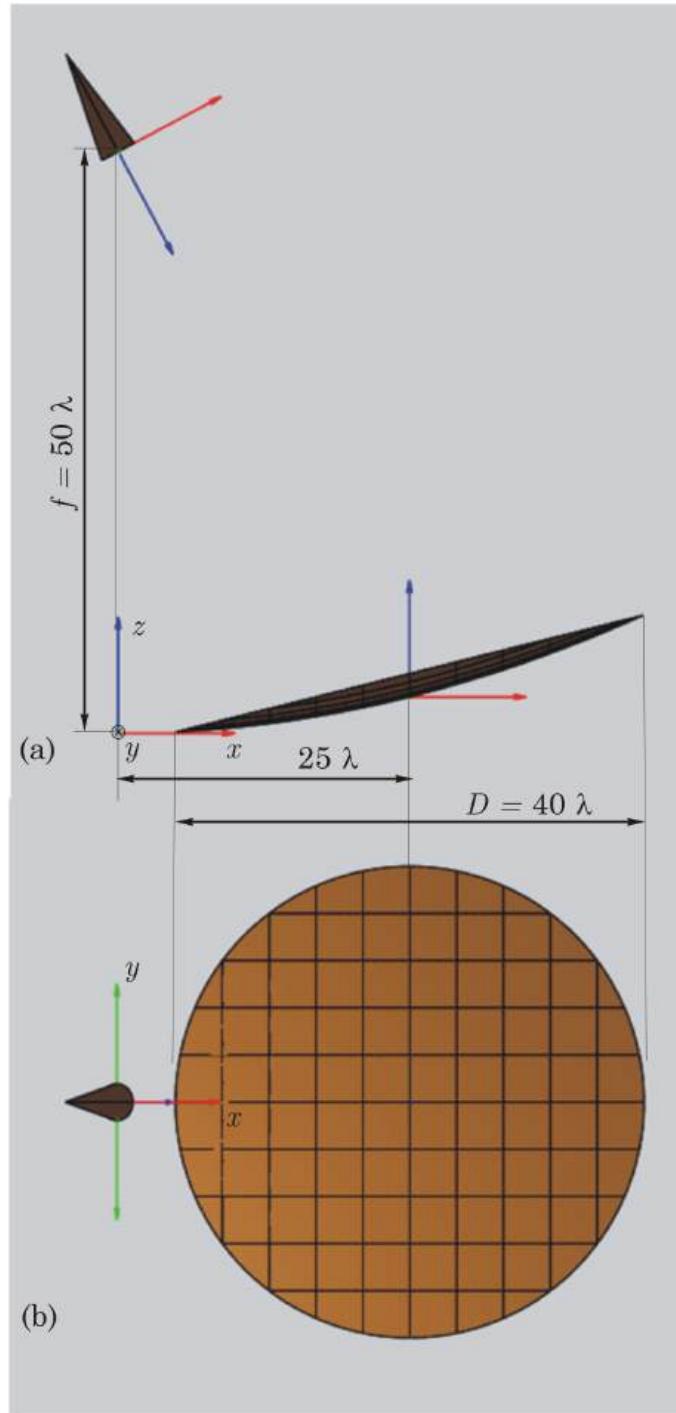


Figure 3-2 Sketch of the geometry to be analyzed - an offset paraboloidal reflector with feed, side view (a) and top view (b).

It consists of a single offset parabolic reflector defined in the xyz -coordinate system by a circular projected aperture of diameter $D=40$ m and a focal length $f= 50$ m, corresponding to an f over D ratio $f/D = 1.25$. The distance from the paraboloid axis to the reflector projected aperture center is 25 m, resulting in a clearance from the paraboloid axis to the nearest edge of the reflector of 5 m.

TIP: A GRASP project has already been set up for this antenna and can be opened from the box TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The case is named SINGLE REFLECTOR WITH ONE FEED. However, it is recommended to follow the procedure given here in order to be familiar with GRASP. The project files of the project are located in the TESTCASES/SINGLE_REFL_WITH_ONE_FEED subdirectory in the installation directory.

We will assume a wavelength, λ , of 1 m as an easy way for having all lengths given in wavelengths.

TIP: The default length unit is meter (m). If another default length unit (such as mm or inch) is desired, then close the wizard and choose TOOLS > OPTIONS ... in the GRASP main window and specify the desired units under tab GLOBAL.

TIP: Tool tips are generated by hovering the mouse over an item (text or depicted object).

TIP: Right click an item to get more options.

The wizard is started by pressing the USE WIZARD button in the GRASP - NEW PROJECT window. If this window is not available, the wizard may also be started by choosing REFLECTOR WIZARD in the WIZARDS menu of the GRASP main window. By doing so, the window shown in Figure 3-3 opens.



Figure 3-3 The opening window of the GRASP - REFLECTOR WIZARD.

We choose a SINGLE REFLECTOR SYSTEM and in the FREQUENCY OR WAVELENGTH-field at the bottom we choose WAVELENGTH, as we want the system to be specified in terms of wavelength. We then press NEXT and reach the window of Figure 3-4.

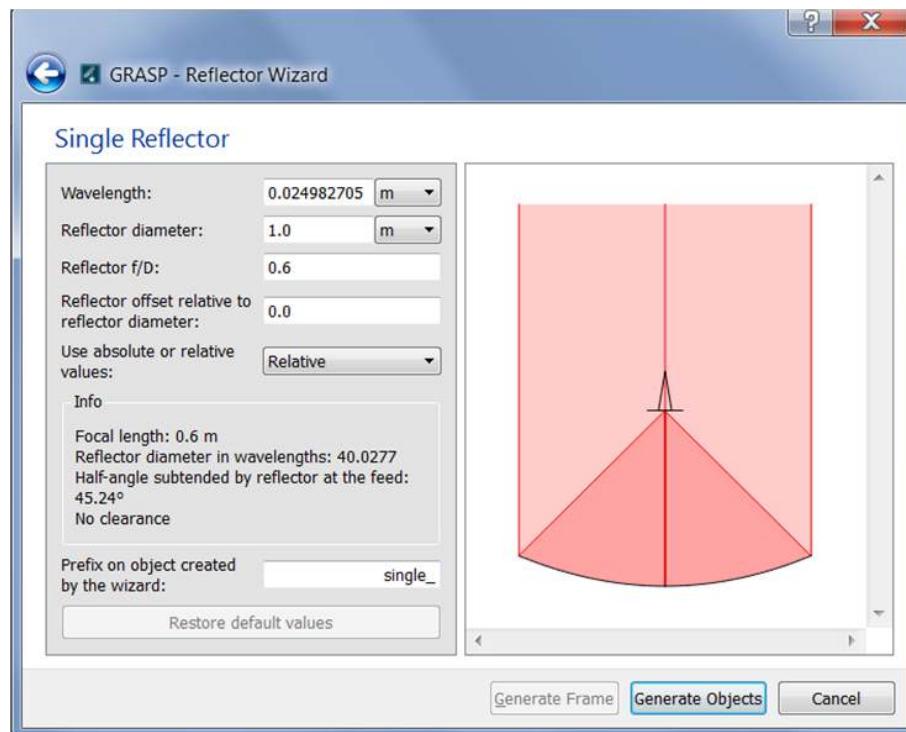


Figure 3-4 The GRASP - REFLECTOR WIZARD ready for specifications.

Default values are suggested as input. We change the wavelength to 1 m and specify the main reflector diameter equal to 40 m. Insertion of these numbers causes the drawn cross section of the reflector to be changed according to the - so far - given specifications. We then define the f/D ratio equal to 1.25 and the reflector offset, i.e. the distance between the axis of the reflector and the axis of the paraboloid, 25 m, relative to the reflector diameter equal to 0.625. Our desired reflector is thus defined and shown in the cross section figure to the right of the specifications, Figure 3-5.

We could as well have given the last lengths in absolute measures. In that case we should have changed RELATIVE to ABSOLUTE in the box below the specification of the reflector offset.

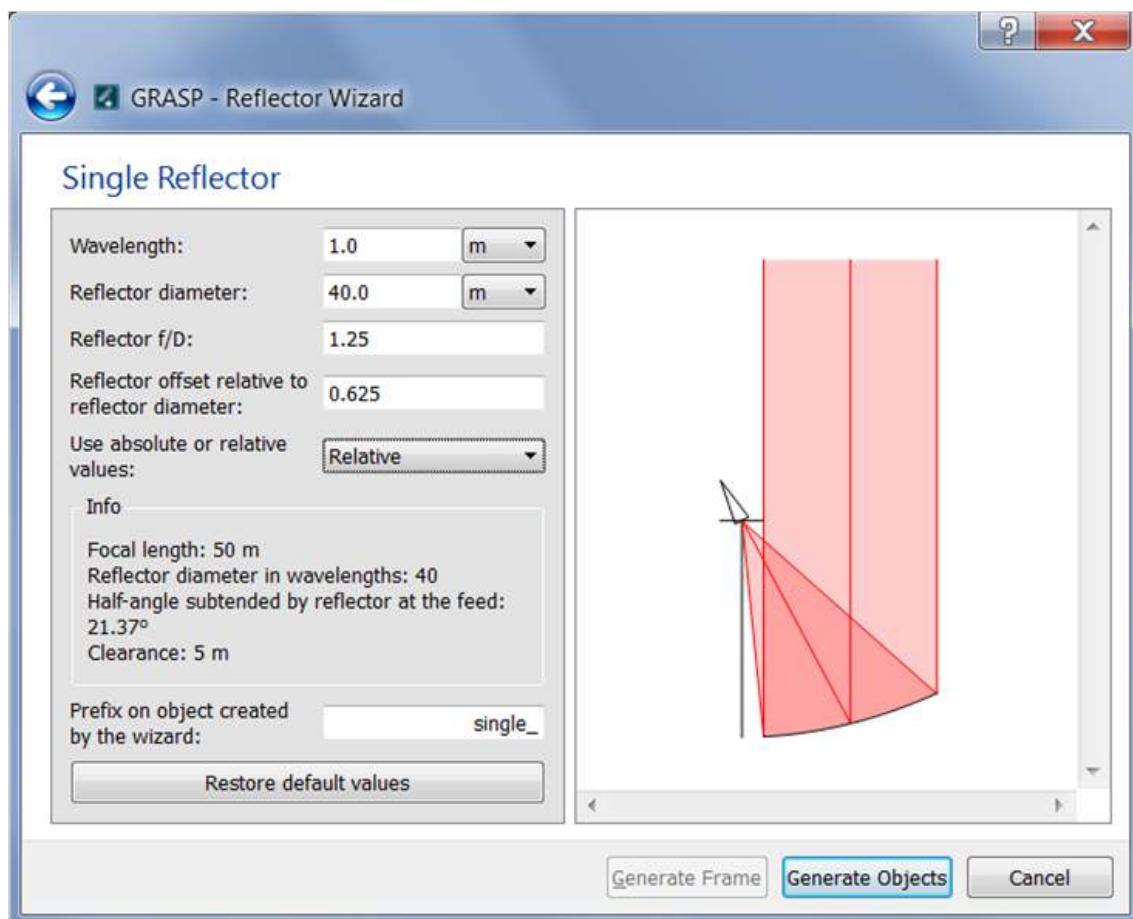


Figure 3-5 The GRASP - REFLECTOR WIZARD. The antenna is sketched as soon as the specifications are given.

In the info-block below the specifications we find information on derived parameters, namely that the focal length is 50 m, the diameter of the reflector measured in wavelengths is 40, and the clearance is 5 m, i.e. the distance from the focal point to the nearest GO reflected ray. As the reflector aperture is circular then lines from the focal point to the edge form a circular cone, and we are informed that the half angle of this cone is 21.37°. Finally, we can specify a prefix for the names of the objects data entities that the wizard is going to create. We accept the default prefix SINGLE_.

The sketch of the antenna may be zoomed by the mouse scroll wheel or by the +/- buttons of the keyboard.

By pressing the GENERATE OBJECTS button the wizard closes and the antenna and its specifications appear in the GRASP main window.

We will in the following chapter describe the GRASP main window and the objects and commands generated by the wizard for the designed reflector.

It is noted that a wizard is also available for a Dual Reflector System and a Ring-Focus System (also called Axial Displaced Reflector System), see Figure 3-3. The geometries of these systems are best understood by opening and using the relative wizards.

4. The GRASP Main Window

After closing the wizard, the main window of GRASP looks like Figure 4-1, and consists of the following parts:

- A top line identifying the program and the version number (here GRASP 10.0.0, later versions have been released):
TICRA - GRASP 10.0.0 - PROJECT: *
The asterisk indicates that the project has not been saved.
- At the top: a line with the GRASP menus to manipulate the project, open the wizards and generate objects (FILE, EDIT, ..). Below, a menu bar, with icons for the most commonly applied tools.
- On the left: four tabs with large icons for the processing. Here the top tab OBJECTS is open.
- A wide left column window with a tree of the defined objects. This tree is denoted the OBJECT EXPLORER, as written at the bottom of the window.
- On the right: a sub-window showing a 3D view of the defined structure, in the present case our single-reflector antenna.

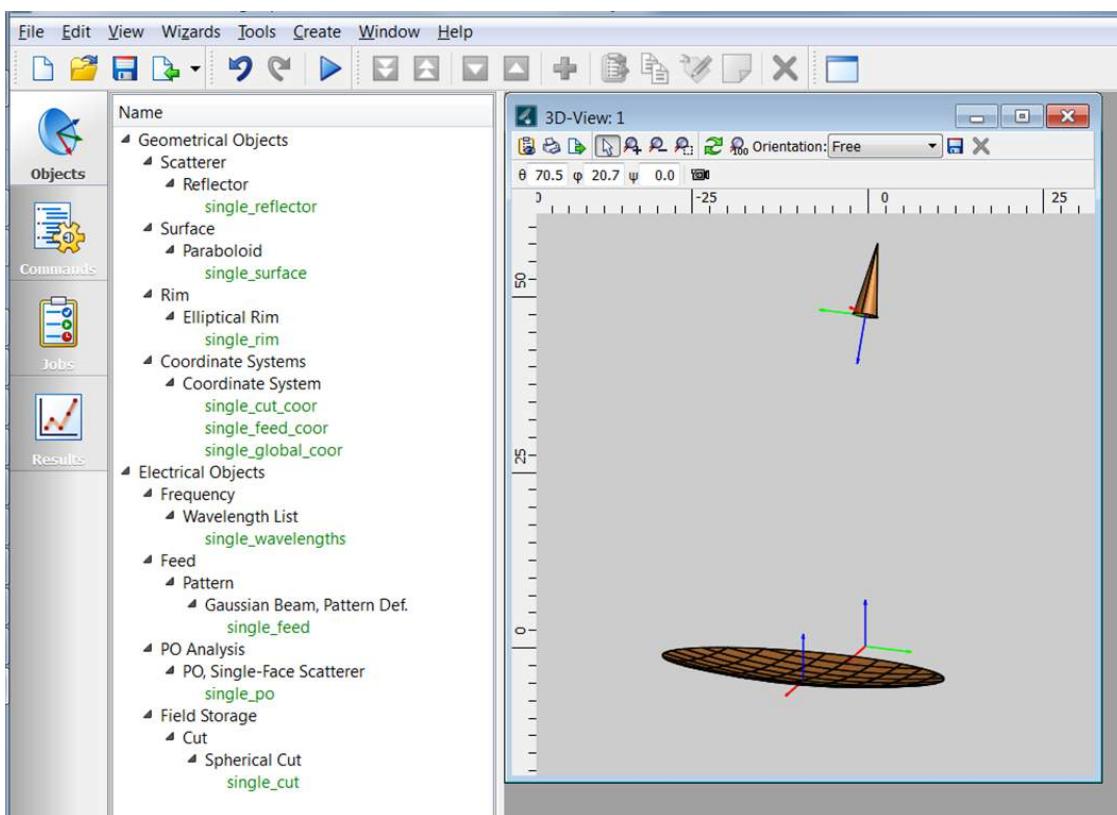


Figure 4-1 The main window of GRASP after finalizing the wizard.

The four tabs on the left represent the steps necessary for running an analysis with GRASP. From the top we see:

- OBJECTS, in which the geometry of the antenna (the geometrical objects) as well as the methods of calculation (the electrical objects) are defined
- COMMANDS, in which the commands for the analysis are given
- JOBS, in which the analysis is started and previous analyses may be reviewed
- RESULTS, in which the results of the analysis are shown.

Further, below the tabs, a blue triangle, a 'play' button, represents the icon for starting an analysis.

The sub-window named 3D-VIEW: 1 shows a parallel projection of the antenna. We can see the reflector, three coordinate systems with axes in colours, and the feed, sketched as a conical horn and pointing towards the reflector. The different items depicted here relate to the geometrical objects and will be discussed in the following section. The features of the 3D-view-window will be presented in Section 4.1.3.

We will now describe the four tabs in detail, in order to perform an analysis of the single reflector antenna defined by the wizard in Chapter 3.

We start by saving the project clicking on SAVE PROJECT As... in the menu FILE. The suggested name NEWPROJECT is changed to SINGLEREFL and the project is saved in a suitable folder location, here C:/USER/TEMP/GRASP is chosen. When the save procedure is executed, the top line in the GRASP window shows the project name. The asterisk has disappeared as the project has not been changed after the saving:

TICRA - GRASP 10.0.0 - PROJECT: SINGLEREFL.GXP

The project is available in the GRASP installation directory, under TESTCASES/SINGLE_REFLECTOR.

4.1 The Objects Window

Clicking the tab OBJECTS opens a window called OBJECT EXPLORER where all defined project objects are listed. A search field is found at the bottom to search for an object in the list. If GRASP is opened as BLANK PROJECT the OBJECT EXPLORER will be empty. To define the objects, the user must then click on CREATE on the menu bar.

For the project SINGLEREFL we find the objects listed in Figure 4-1. It is observed that objects are held in two major groups: GEOMETRICAL OBJECTS and ELECTRICAL OBJECTS. Any object may be specified from the GRASP menu CREATE.

The GEOMETRICAL OBJECTS specify the geometry of the antennas and scatterers to be considered in the analysis.

The ELECTRICAL OBJECTS specify the frequencies (alternatively the wavelengths) of operation, the source elements, the electromagnetic method of calculation (PO/PTD, GO/GTD, MoM, PWE, SWE) and electrical properties. Moreover, grids or cuts of the points at which the field shall be determined are specified here.

The OBJECT EXPLORER is fully expanded showing all objects of the project. The list may be collapsed and expanded by clicking the small triangles to the left of the names, or by applying the EXPAND and COLLAPSE commands in the VIEW menu.

All objects are described in detail in the *Reference Section* in Chapter 9.

4.1.1 Geometrical Objects

The first geometrical object we encounter is a SCATTERER of type REFLECTOR with name SINGLE_REFLECTOR.

TIP: The object names are written in green while the categorizing names are in black (may be changed in the menu TOOLS > OPTIONS ... under the tab DISPLAY).

TIP: If we click on an object name, the object is highlighted (changes its color) in the 3D-view window for an identification. This is useful when many objects are present.

TIP: Right-clicking on an object name gives the possibility of editing, duplicating, renaming and deleting the object.

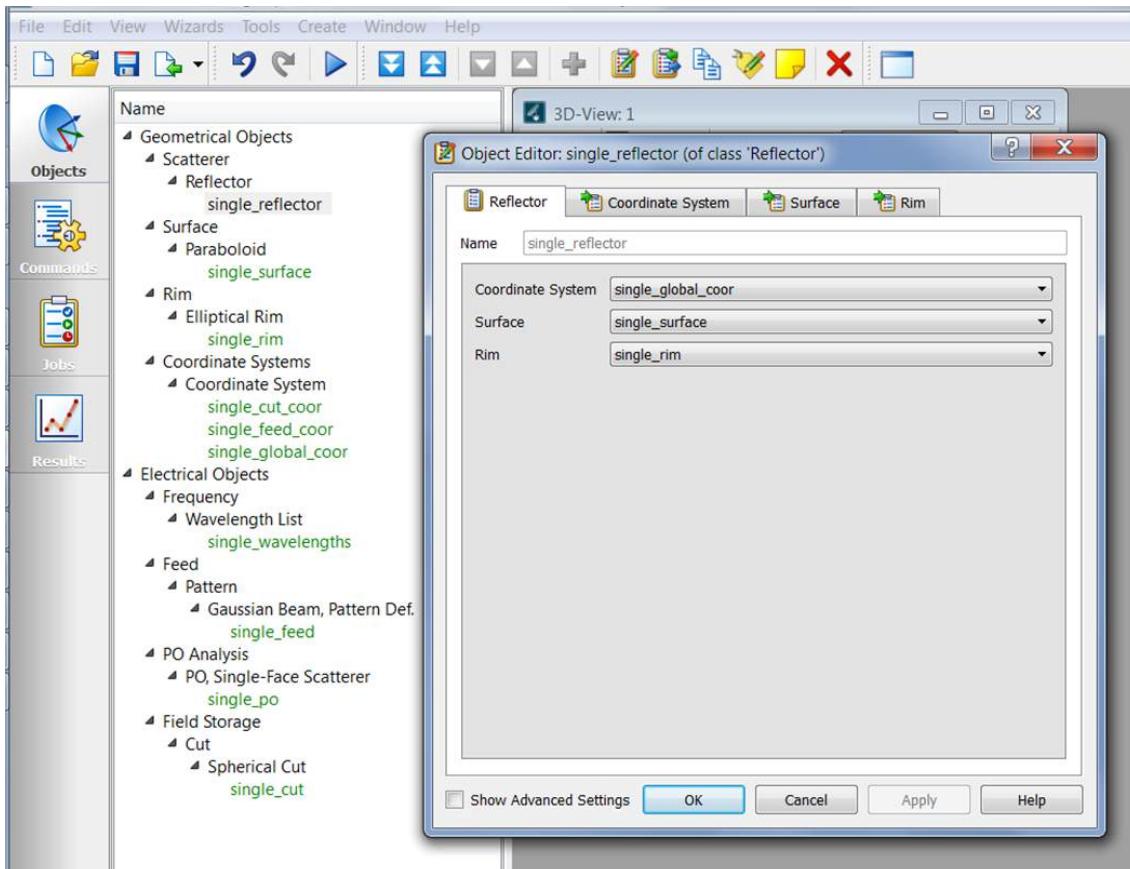


Figure 4-2 The OBJECT EDITOR window with data for the SINGLE_REFLECTOR object with the tab REFLECTOR opened.

The object is opened by double-clicking the object name or by right-clicking the object name and choosing EDIT. When we do so, we obtain Figure 4-2. The window shows the tab REFLECTOR active and the attributes of the reflector. These are:

- the *Coordinate System*, SINGLE_GLOBAL_COOR, in which the reflector is defined by its surface and its limiting rim,
- the *Surface*, SINGLE_SURFACE, in general a function of (x, y) in the reflector coordinate system
- the *Rim*, SINGLE_RIM, defining the rim of the reflector by its projection on the (x, y) -plane in the reflector coordinate system.

The above listed attributes which specify the objects defining the reflector may be opened for inspection or for editing by clicking the corresponding

objects in the OBJECTS EXPLORER, or from the tabs in the present OBJECT EDITOR window. If necessary, additional attributes may be specified for a reflector, such as one or more holes, surface distortions, serrations and electrical properties. Additional attributes are available for several classes, but only when the box SHOW ADVANCED SETTINGS is marked.

TIP: The advanced settings can once for all be switched on and off in the TOOLS > OPTIONS ... menu.

TIP: The meaning of the attributes can be explored by the tool tips appearing when the pointer is hovered over the name of the attributes. Moreover, clicking the button HELP (or pressing F1) within the OBJECT EDITOR opens the description of the class with discussions and illustrations of its application.

If we open the tab COORDINATE SYSTEM we obtain Figure 4-3:

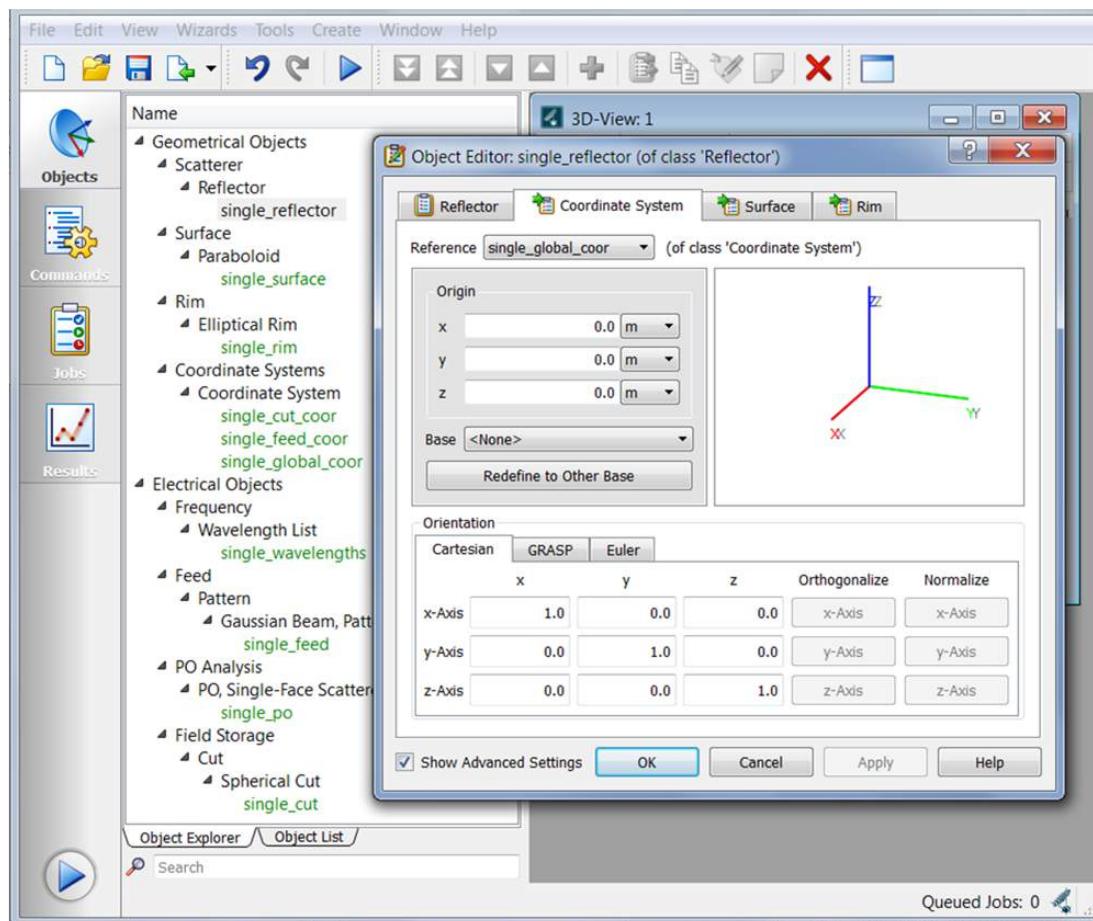


Figure 4-3 The OBJECT EDITOR window with data for the SINGLE_REFLECTOR object and the tab COORDINATE SYSTEM opened, showing data for the object SINGLE_GLOBAL_COOR.

As it can be seen, a coordinate system is defined by the position of its ORIGIN and its ORIENTATION relative to a BASE, which is an already defined coordinate system. In the present case the reflector coordinate system SINGLE_GLOBAL_COOR has as base the unnamed coordinate system of GRASP

called <NONE>. The <NONE> coordinate system is the only coordinate system of GRASP which cannot be edited, i.e. rotated and translated.

The origin of a coordinate system is given by the x -, y - and z -value. The orientation can be defined

1. In the Cartesian system, by the unit vectors expressing the direction of the x -, y - and z -axes of the BASE coordinate system
2. In the so-called GRASP notation, by rotating the coordinate system the three angles θ , ϕ and ψ
3. In Euler notation, by rotating the coordinate system the three angles α , β and γ

TIP: In the Cartesian definition it is possible to orthogonalize one of the directional vectors with respect to the two others and to normalize any of the vectors to unit length.

For a definition of the GRASP notation and Euler notation please refer to the [Reference Section](#) in Chapter 9, or press HELP (or F1). The orientation is also illustrated in the OBJECT EDITOR window for coordinate systems. Here, the axes shown in grey are the ones of the BASE coordinate system, while the colored axes are the ones of the system being defined. The convention in GRASP is that the x -axis is red, y -axis is green and the z -axis is blue (mnemonic rule: x , y and z in the order as in the name of the colour code RGB). From the origin and orientation, it is possible to see that the reflector coordinate system SINGLE_GLOBAL_COOR coincides with the BASE coordinate system.

At the bottom of the frame ORIGIN the box REDEFINE TO OTHER BASE is found. By opening this box it is possible to *redefine* the actual coordinate system such that it is defined in another coordinate system. The actual system is not moved by this procedure and the procedure must not be confused with *specifying* another base (in the box one line above) for the actual system whereby the system in general will be moved with respect to the global system.

If we open the tab SURFACE we obtain Figure 4-4. It is seen that the surface of the reflector, SINGLE_SURFACE, is a PARABOLOID with a focal length of 50 m and vertex at $(x, y, z) = (0.0 \text{ m}, 0.0 \text{ m}, 0.0 \text{ m})$ in the reflector coordinate system.

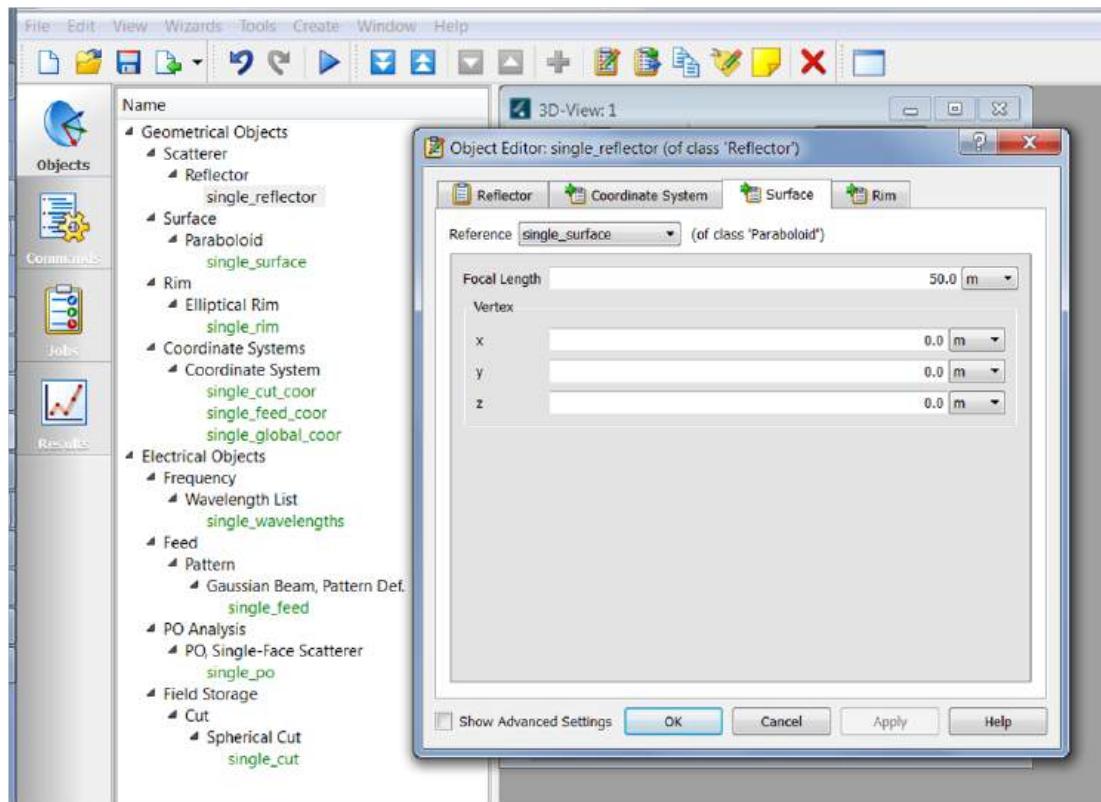


Figure 4-4

The OBJECT EDITOR window for the SINGLE_REFLECTOR object with the tab SURFACE opened, showing data for the object SINGLE_SURFACE.

Finally, the tab RIM, see Figure 4-5, shows that the rim of the reflector SINGLE_RIM is an ELLIPTICAL RIM centered at $(x, y) = (25.0 \text{ m}, 0.0 \text{ m})$ and having half axes of 20.0 m, i.e. it is a circle.

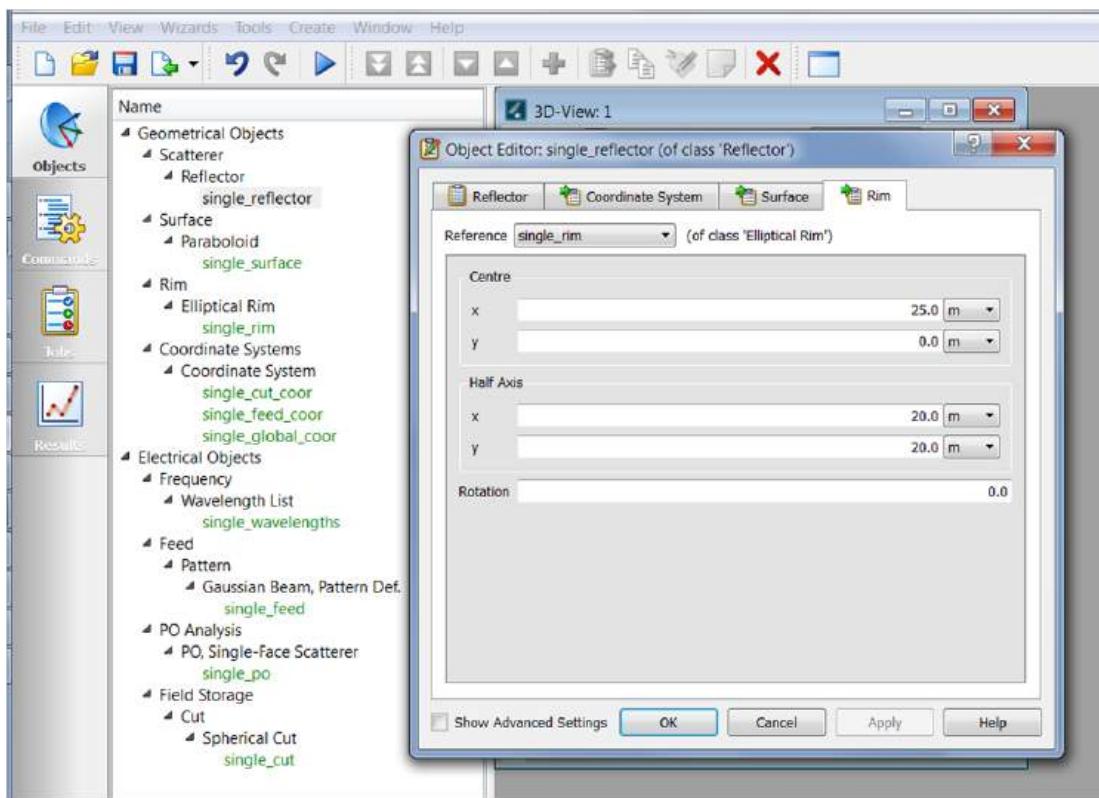


Figure 4-5 The OBJECT EDITOR window for the SINGLE_REFLECTOR object with the tab RIM opened, showing data for the object SINGLE_RIM.

The OBJECT EDITOR window is closed by choosing OK or - as no changes have to be stored - CANCEL at the bottom of the window.

We can then have a look at the SINGLE_FEED_COOR in which the feed is defined. To do that we can either double click on it in the 3D-view window, or in the Object Explorer. The corresponding OBJECT EDITOR will open, see Figure 4-6:

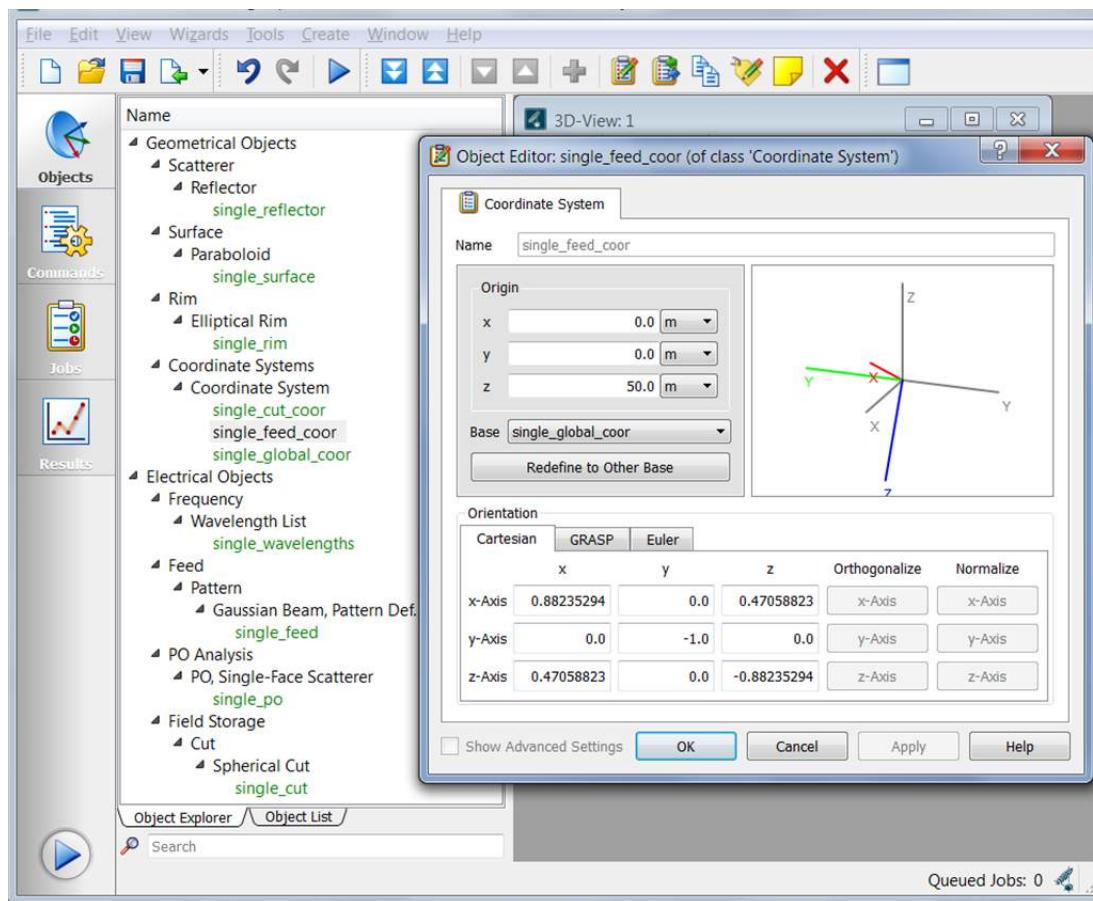


Figure 4-6 The OBJECT EDITOR window with data for the feed coordinate system in SINGLE_FEED_COOR.

We can see that the origin of the feed coordinate system is at the focal point of the paraboloidal reflector surface at $(x, y, z) = (0.0 \text{ m}, 0.0 \text{ m}, 50.0 \text{ m})$ in the reflector coordinate system. This is specified by BASE: SINGLE_GLOBAL_COOR. The orientation of the axes of the feed coordinate system is determined by the wizard such that the z -axis points to the center of the reflector surface. This choice results in the most symmetrical illumination of the reflector. The orientation of the feed coordinate system can be better understood by looking at Figure 4-7, where the orientation is given with GRASP angles (obtained by clicking the GRASP tab). It can be seen that the z -axis of the feed system is obtained by rotating the original z -axis an angle THETA of 151.9° in the direction PHI = 0° . The direction of the y -axis is finally controlled by the choice of PSI equal to 180° . A detailed explanation of the GRASP and Euler rotation angles may be found in the description of the respective coordinate system classes, see *Coordinate Systems* or press F1 in the GUI when the coordinate system is opened in the OBJECT EDITOR.

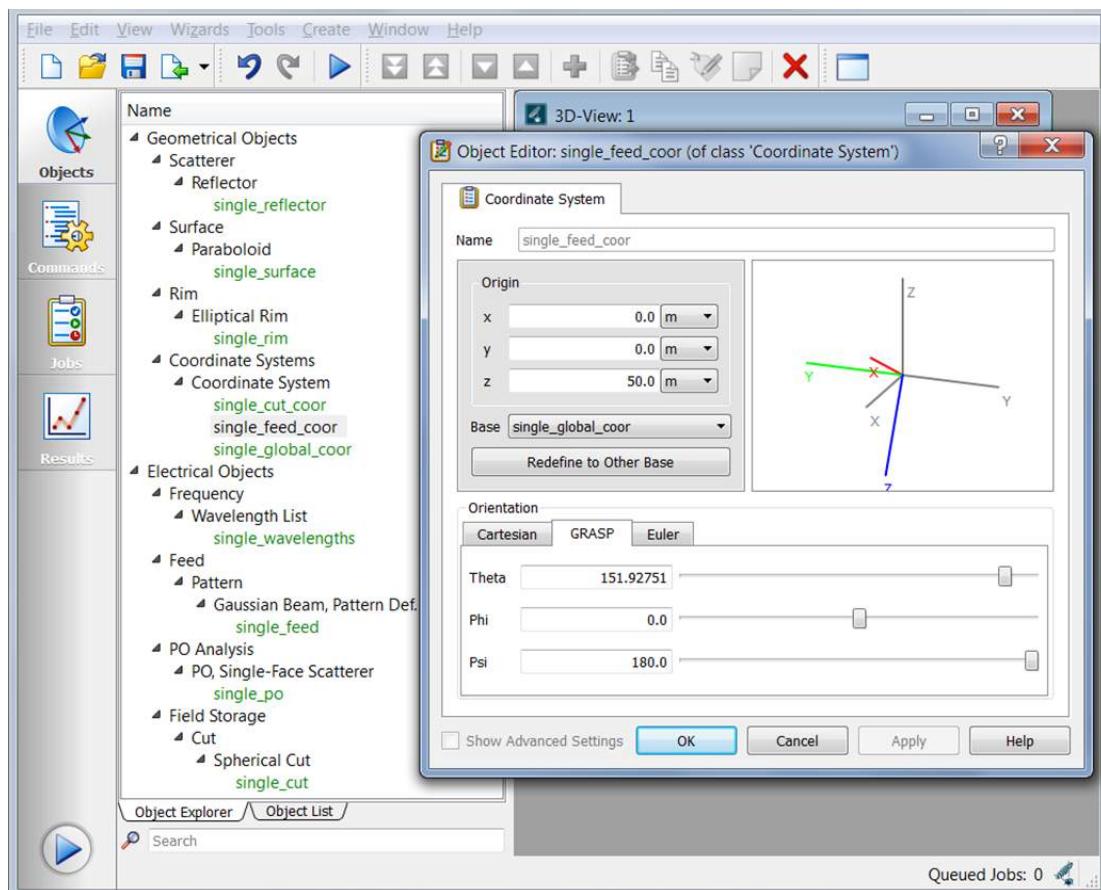


Figure 4-7 Feed coordinate system with the orientation given in the GRASP definition.

4.1.2 Electrical Objects

The next group of objects listed in the OBJECT EDITOR is the ELECTRICAL OBJECTS. The first of these objects (with the name in green) is the object SINGLE_WAVELENGTHS of class WAVELENGTH LIST of type FREQUENCY. Double clicking the name opens the OBJECT EDITOR for this object, as shown in Figure 4-8:

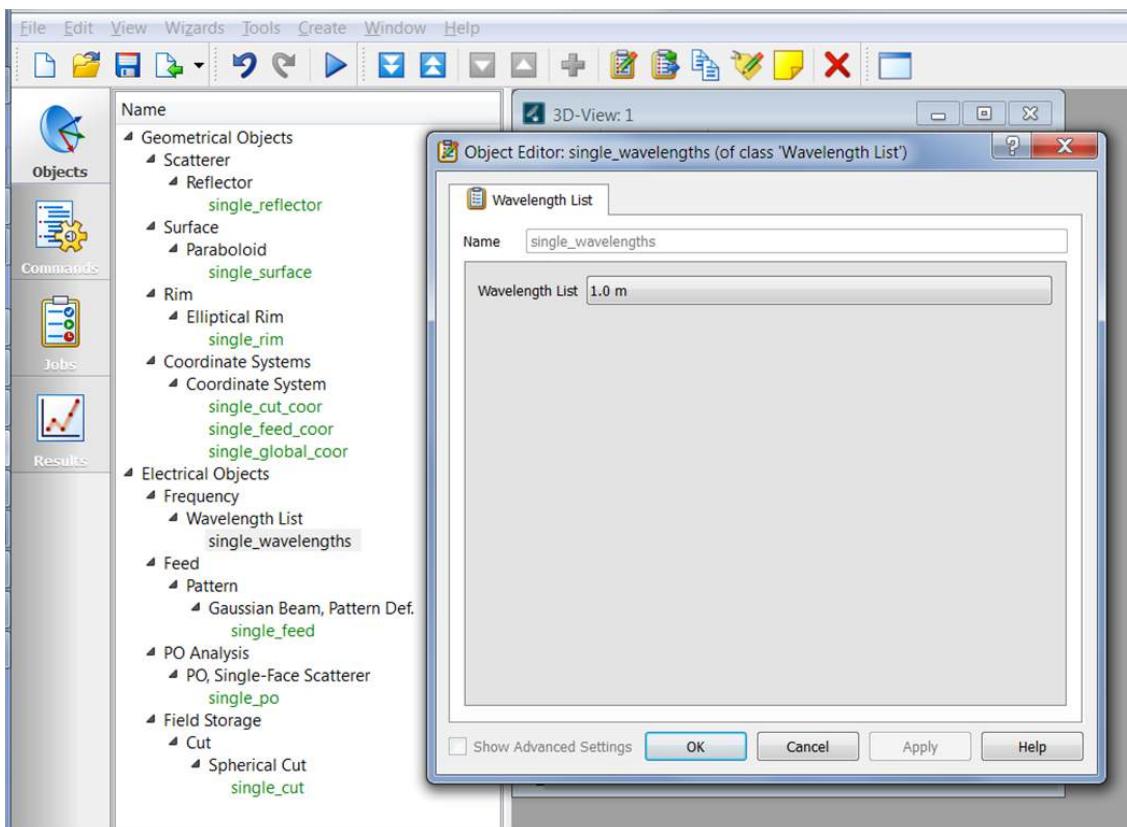


Figure 4-8 The OBJECT EDITOR window with data for the wavelength object SINGLE_WAVELENGTHS.

In the box WAVELENGTH LIST the wavelength is given (1.0 m). By clicking the box, a sub-window opens in which additional wavelengths may be specified. We leave this window by clicking on CANCEL. Wavelengths (or frequencies) may also be specified by a number of wavelengths (or frequencies) within a given range through the menu CREATE > ELECTRICAL OBJECTS > FREQUENCIES.

The next electrical object is SINGLE_FEED, i.e. a feed of class GAUSSIAN BEAM, PATTERN DEF. This means that the feed radiates a Gaussian beam, a circular-symmetric beam which presents a good approximation to the main beam of typical feed patterns, both in the far field as well as in the near field region. This is especially important when the reflector is in the near field of the feed¹. Double clicking the object opens the OBJECT EDITOR window for the SINGLE_FEED, Figure 4-9:

¹The Gaussian beam is described in detail in the GRASP Technical Description.

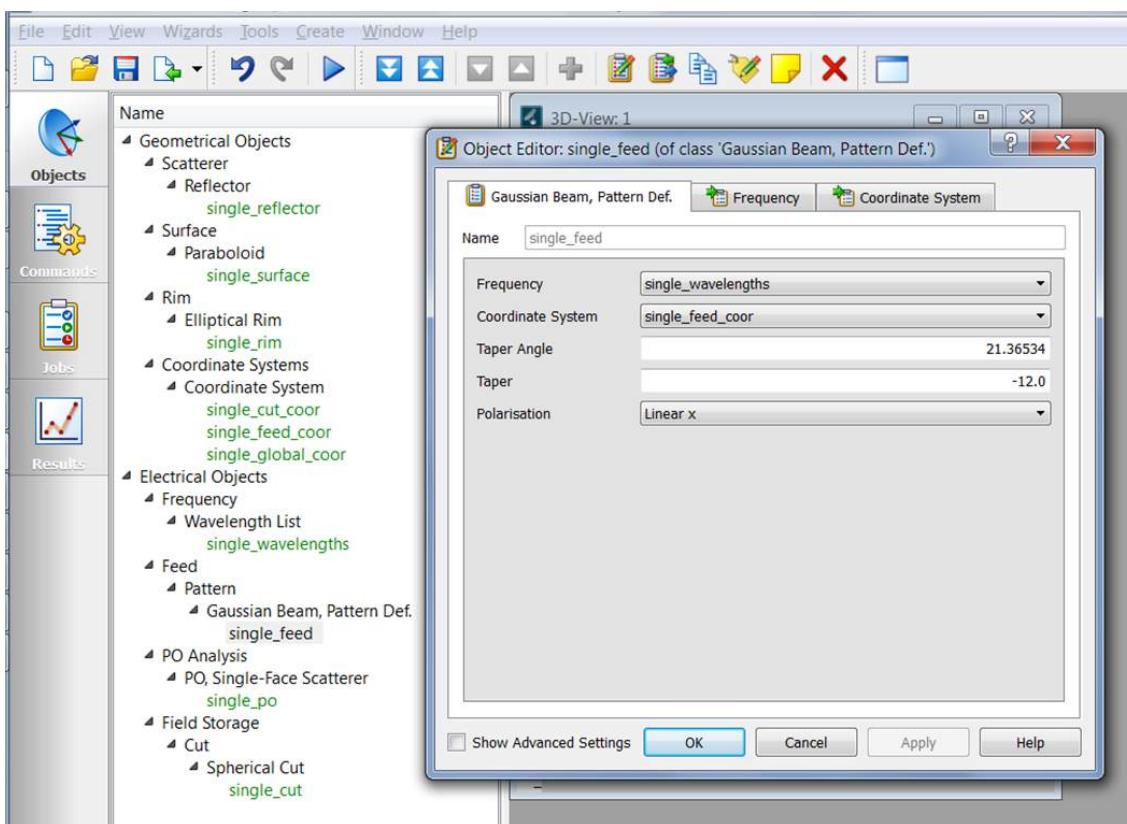


Figure 4-9 The OBJECT EDITOR window with data for the feed object SINGLE_FEED.

We here see the frequency at which the feed shall operate, SINGLE_WAVELENGTHS, and the coordinate system in which the feed is defined, SINGLE_FEED_COOR. The feed radiates along the z -axis of the feed coordinate system with a TAPER of -12 dB (relative to peak) at the TAPER ANGLE 21.36534° . This angle is the half cone angle discussed in connection with Figure 3-5.

The TAPER of the feed is set by the wizard by default to -12 dB as this value provides the maximum directivity. We further see that the polarization of the feed is LINEAR_X (linear polarized along the feed x -axis). Further attributes are available when SHOW ADVANCED SETTINGS is checked.

TIP: Remind that tool tips for the various attributes are given when the pointer is hovered over the name of the attribute.

The last two objects for our single-reflector design, SINGLE_PO and SINGLE_CUT, concern the computation of the antenna pattern.

The first of these objects, SINGLE_PO, belongs to the class PO, SINGLE-FACE SCATTERER and defines the field computation carried out by physical optics (PO). Opening the object by double clicking the name results in Figure 4-10:

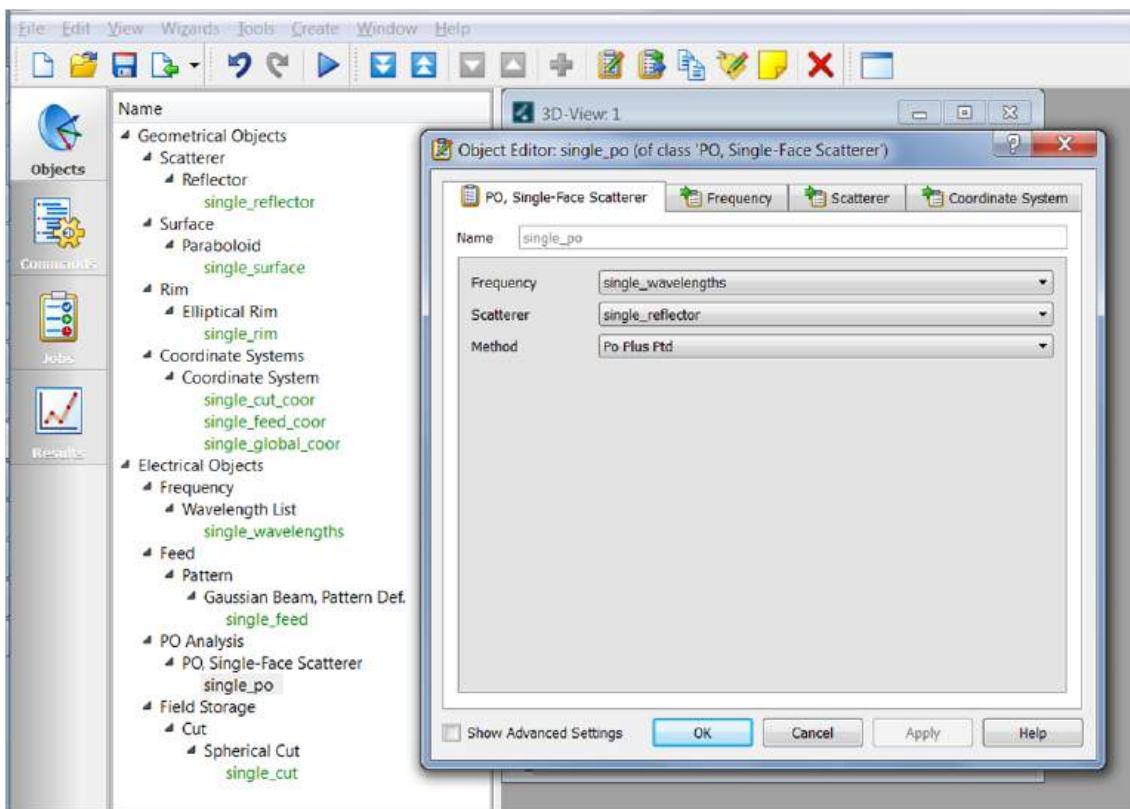


Figure 4-10 The OBJECT EDITOR window with data specifying the PO calculations.

The frequency to be used is specified by the object SINGLE_WAVELENGTHS, and the scatterer is given by the reflector SINGLE_REFLECTOR. The method to be applied is specified by PO_PLUS_PTD, i.e. to the PO currents over the reflector are added the PTD currents from the edges of the reflector.

We finally consider the last electrical object, i.e. SINGLE_CUT. We open it in the OBJECT EDITOR by double clicking its name and obtain Figure 4-11:

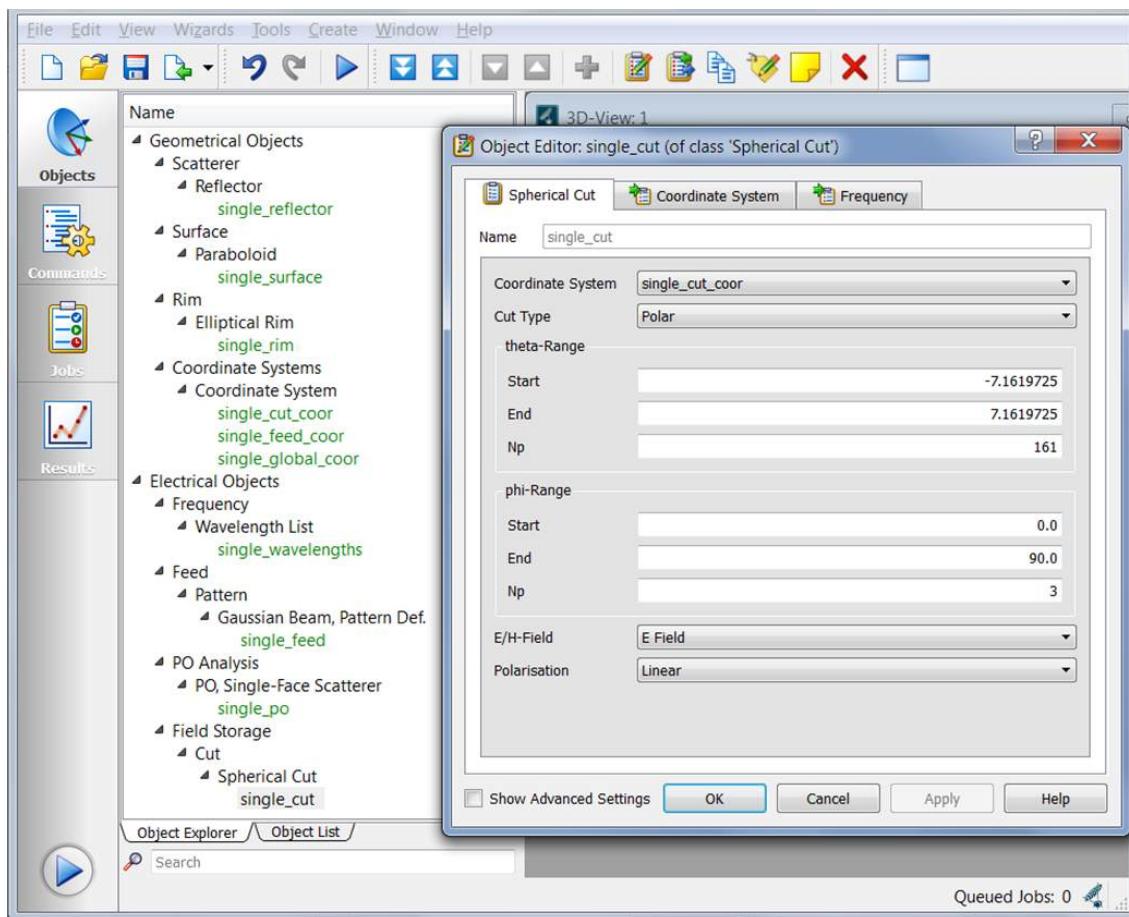


Figure 4-11 The OBJECT EDITOR window with data for object SINGLE_CUT specifying the far field directions in which the pattern shall be calculated.

This object defines the points at which the field shall be calculated and stored. It is of class SPHERICAL CUT, i.e. defined on a spherical surface to be specified in the given coordinate system: SINGLE_CUT_COOR. The attribute CUT TYPE is specified to POLAR, which means that θ varies while ϕ is constant for the cut(s), θ and ϕ being usual spherical coordinates. The range for the θ -values are given in the box THETA RANGE. Here, the wizard has inserted values for θ starting at -7.16° and ending at 7.16° with 161 points. This angular range will cover the main beam and the first few side lobes. The range for the ϕ -values are given in the next box, PHI RANGE. Three cuts are specified, $\phi = 0^\circ, 45^\circ, 90^\circ$. The last attributes have default values: the E-field shall be determined in linear polarization. The file format is suggested to be in the TICRA standard format which is a simple ASCII format.

We finally save the project with FILE > SAVE PROJECT (or by Ctrl+s from the keyboard).

TIP: Mark the check box SHOW ADVANCED SETTINGS to specify, e.g., field points in the near field.

4.1.3 3D-View

In this section we will present the features of the window in which the modelled antenna structure is presented in a 3D-view. The window is available only when the OBJECTS-tab is open.

Most of the items of the menu bar of this window (apart from the direction specifications θ, ϕ, ψ) may also be chosen from the VIEW menu or from right clicking the canvas. We will consider these items in the following.

Orientation

The field denoted by ORIENTATION is by default set to FREE, see Figure 4-1, i.e. the figure may be rotated in all orientations.

The figure in the view is **rotated** by holding down the left mouse button and dragging the mouse; the figure is **translated** in the window by holding down the right mouse button and dragging the mouse. The figure is rotated around a point centrally in the figure or around an object if that object has been highlighted. Finally, when the ORIENTATION is set to FREE, then choosing RESET VIEW will fit the figure to the window in DIMETRIC orientation.

Other orientations may be chosen, i.e. projections on the xy -, xz - and yz -planes. In all these projections the figure may be rotated, but only in the plane of the window.

Rotating a figure corresponds to observe the geometry from a direction which is given by the spherical coordinates (θ, ϕ) in a coordinate system which by default is the basic coordinate system of GRASP, i.e. the <NONE> coordinate system described in Figure 4-3 in Section 4.1. These angles are shown in the menu bar together with the angle ψ . The latter angle corresponds to a rotation of the figure without changing the direction from which it is seen. Angular values may also be typed into these fields. A chosen orientation may be saved (the icon to the right of menu point ORIENTATION, and later restored from the ORIENTATION-menu. The orientation may be specified to be given in another coordinate system in the VIEW SETTINGS EDITOR, see below for a description of this.

Cursor Mode and Zoom

The default cursor mode is the SELECT CURSOR MODE which appears as an arrow by which objects may be selected. When the cursor is hovered over an object then the name of the object will be given. This is very useful when many objects have been defined. If the mouse is clicked here, then the object will be highlighted; further, by double clicking, the object will open in the OBJECT EDITOR.

Alternatively, a zoom of the objects is possible. The ZOOM IN CURSOR MODE and the ZOOM OUT CURSOR MODE zoom on a mouse click around the point marked by the cursor; with the RECTANGLE ZOOM MODE a rectangle may be marked and zoomed to fit the window. The zoom may be reset applying the RESET ZOOM button (with a 100 - for 100% - in the icon).

TIP: When an object is highlighted, the zoom will be adjusted such that the object fits the window when the RESET ZOOM is applied.

The View Settings Editor

Right clicking an object will open an object menu by which all objects existing at the position may be listed. An example is given in Figure 4-12. Further, at the bottom of the list, it is possible to deselect the grid lines at surfaces as well as the surfaces themselves. The latter may be useful in prints of the 3D-View.

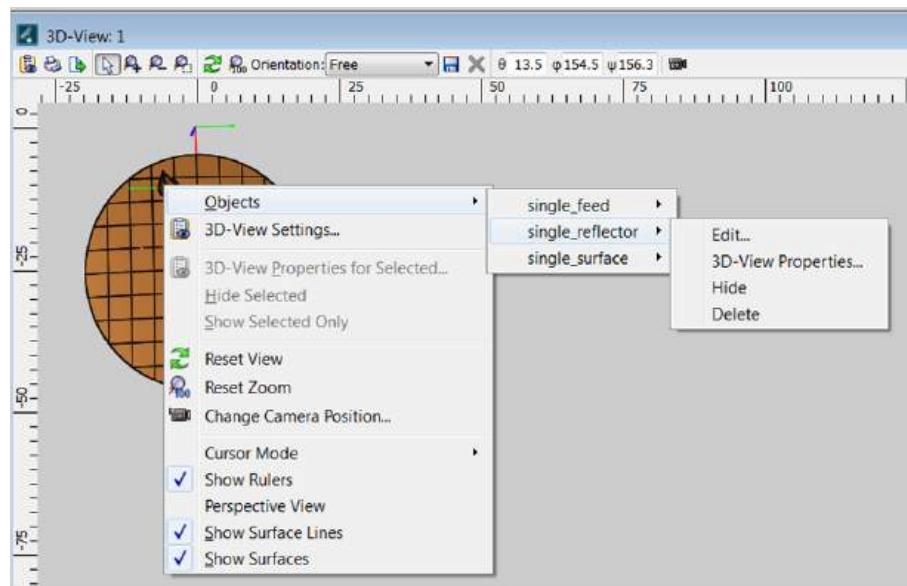


Figure 4-12 A right click at an object opens for manipulating the object. The cursor (not shown in the figure) was pointed at the position where the upper left corner of the text window appears, and here the three listed objects were found beneath the cursor position.

When an object in the Objects list, Figure 4-12, is selected then it is possible to:

- EDIT...: the object opens in the OBJECT EDITOR,
- 3D-VIEW PROPERTIES...: the object is selected in the 3D-VIEW SETTINGS editor (see below),
- HIDE: the object is hidden from the depiction (see later on how to reveal it again), and
- DELETE: the object is deleted, also from the OBJECT EXPLORER (an undo is available in the menu EDIT if an object is deleted by a mistake).

In the 3D-VIEW SETTINGS window, view settings for the various objects may be specified. The window may also be opened directly after right-clicking

the canvas (or from the left menu of Figure 4-12) and choosing 3D-VIEW SETTINGS.... In such cases the editor opens in a general mode, see Figure 4-13:

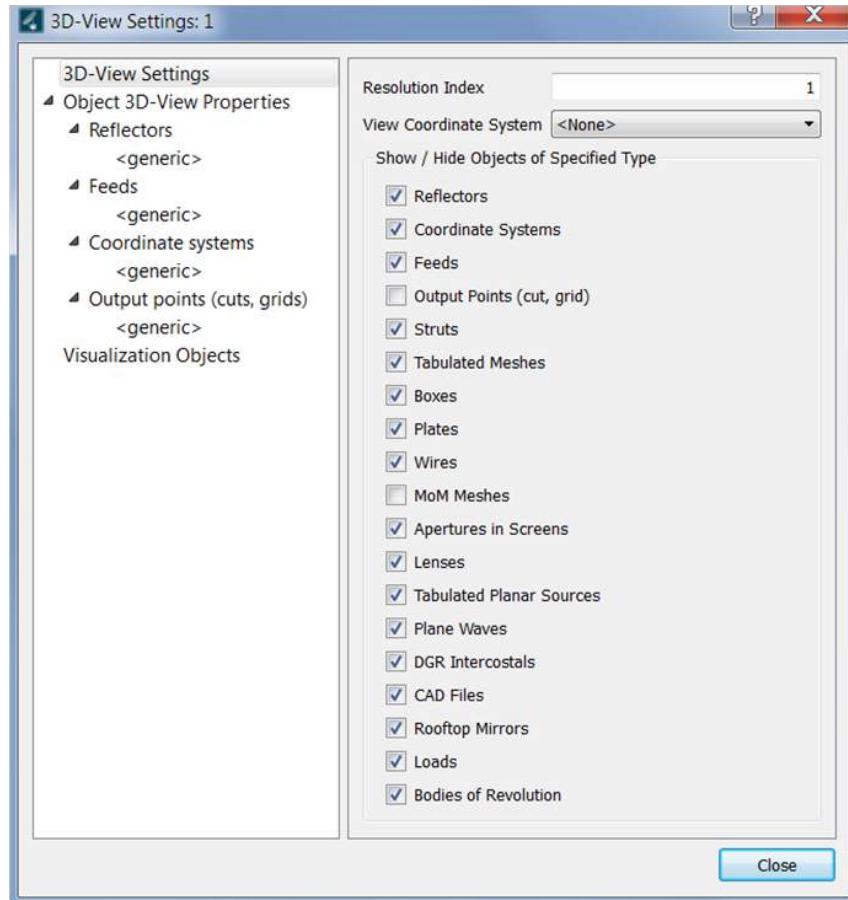


Figure 4-13 The editor for the 3D-VIEW SETTINGS.

The window 3D-VIEW SETTINGS has at the left a tree with the topics to edit, and to the right the panel for editing.

The first item in the tree is 3D-VIEW SETTINGS. When this is marked, it is in the right pane possible to set visual properties common for all objects. The tick marks indicate for which type of objects all the objects shown.

At the first line of the right pane we find the Resolution Index by which the resolution of the 3D-View may be improved, hover over the line to see a detailed explanation.

In the next line the VIEW COORDINATE SYSTEM may be specified. It is here possible to choose the coordinate system for the view, i.e. the coordinate system in which the geometry is presented. It is this system which has the z-axis pointing up in the view when the Orientation is set to Dimetric (in the menu line of the 3D-View).

In the second item in the left part of the window, OBJECT 3D-VIEW PROPERTIES, geometrical objects defined in the project may be chosen and given individual view properties. The objects are listed by class, and clicking a class name (e.g. Coordinate Systems) opens at the right part of the window a

possibility, CUSTOMIZE 3D-VIEW PROPERTIES, to choose among the defined objects (here coordinate systems) and set specific properties for each, for example to set the Axis Length for the coordinate axes of SINGLE_FEED_COOR.

The same point is reached from right-clicking an object in the 3D-view window, choosing OBJECTS and clicking the 3D-VIEW PROPERTIES of the requested object (such as the coordinate system SINGLE_FEED_COOR), cf. Figure 4-12.

When some view settings are changed for an object (for a coordinate system it may be the length of the axes) then this object is included in the tree in the left half of the editor window. Objects which are not included will then have visual properties as specified by <GENERIC>.

The last entry in the 3D-VIEW SETTINGS window is VISUALIZATION OBJECTS. By the available objects it is possible to visualize the flow of power in a scattering system by showing rays radiating from a source (or from a given point) in specified directions. When the ray path is interrupted by a scatterer the ray is optically reflected. An example is shown in Figure 4-14.

For sources defined as a Gaussian beam it is possible to illustrate a beam tube which is especially useful in beam waveguide systems. Examples on this may be found in the Remarks section under *Gaussian Beam Tube*.

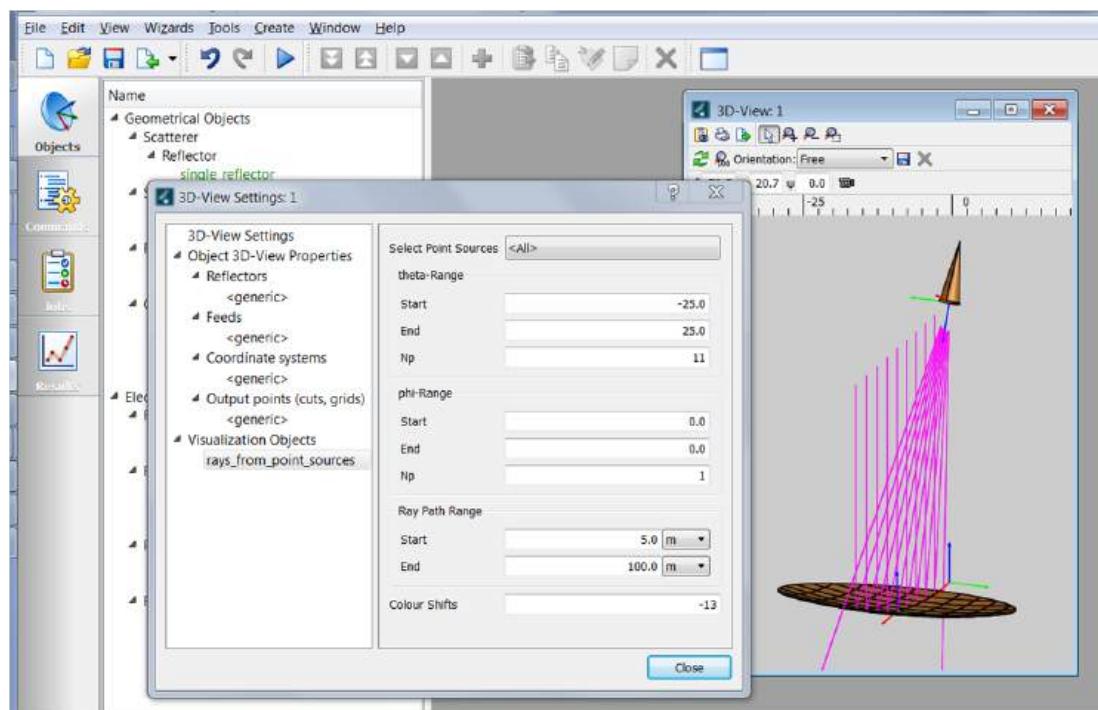


Figure 4-14 Example on a visualization object, RAYS_FROM_POINT_SOURCES, defined in the 3D-VIEW SETTINGS window. 11 rays are drawn in the THETA-RANGE from -25° to 25° in the plane $\phi = 0^\circ$ (PHI-RANGE). The ray paths are drawn starting 5 m from the source and ranging 100 m.

Hide or Display an Object

An object may be hidden, i.e. not displayed in the view, by selecting the object (left click) and next, after a right click on the canvas, choosing HIDE SELECTED. Hidden objects may be displayed again after right clicking the canvas.

Constructing New Objects

New objects may be included through the menu point CREATE. Thus, a new object of class REFLECTOR may be generated by choosing CREATE > GEOMETRICAL OBJECTS > SCATTERER > REFLECTOR.

TIP: If the class name is known but it is difficult to find the path to the class in the CREATE menu, then search for the class in the Alphabetical List of Classes, Section 9.2, find the class description and open it. The path to the class is then given under the heading Links.

When the class is chosen, the OBJECT EDITOR opens for specification of the attributes of the new object. Some attributes have default values which may be applied, but for other attributes the user must give a specification. These attributes are indicated by a red dot until a specification is given.

Some attributes are seldom used and only shown when the field SHOW ADVANCED SETTINGS has been checked.

Tool tips are - as mentioned - shown when the pointer is hovered over the attribute name. For some attributes, the specifications may be taken from a list. In order to see tool tips for a specifications, this has to be chosen before the pointer is hovered over the specification.

4.2 The Commands Window

In the previous Section 4.1 we have defined the antenna by specifying its geometrical and electrical objects. We shall now describe the commands for running the analysis of the antenna.

The commands generated by the wizard are shown by clicking the tab COMMANDS to the left in the GRASP main window. The GRASP commands consist of the command itself followed by two sets of objects denoted SOURCE and TARGET. In general, the SOURCE objects are the command input and the TARGET objects are the command output.

For the single reflector antenna specified by the wizard, two commands have been defined, see Figure 4-15

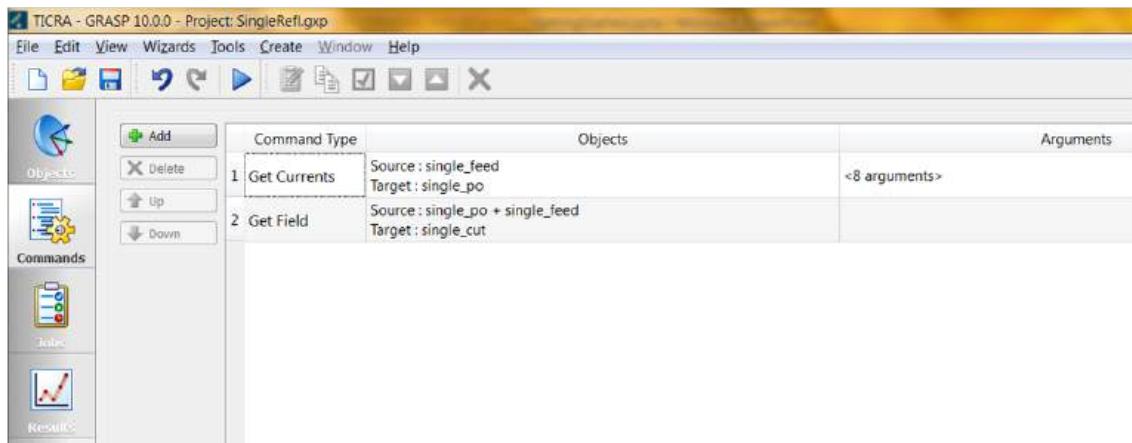


Figure 4-15 The COMMAND EDITOR window with the commands generated by the wizard.

The commands are:

- a GET CURRENTS command by which the PO (and PTD) currents on the reflector are determined, and
- a GET FIELD command by which the field radiated by these currents are calculated by integration of the currents.

We will in the following consider the commands in details.

By double clicking the first command, GET CURRENTS, the corresponding window of the COMMAND EDITOR opens, see Figure 4-16

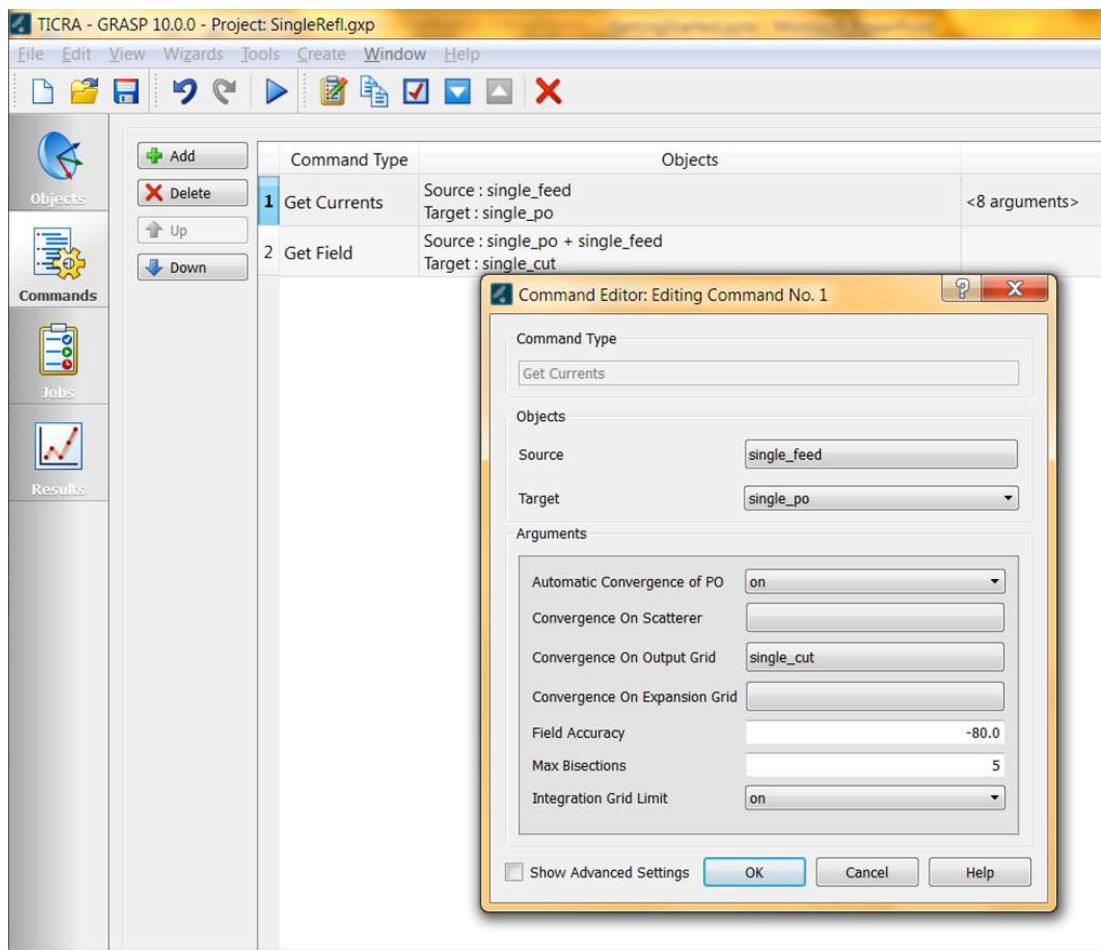


Figure 4-16 The COMMAND EDITOR window for the command GET CURRENTS.

The GET CURRENTS command has as SOURCE object given by the feed, SINGLE_FEED. This feed illuminates the reflector and induces hereby currents upon this. These currents are calculated by PO and PTD as specified in the TARGET object SINGLE_PO. Additional arguments for the computation are specified in the lower part of the window. The AUTOMATIC CONVERGENCE OF PO has been set to ON, as a general recommendation. It is noted that the PO currents (as well as the PTD currents) shall be determined on a grid of sufficient density in order to provide a sufficiently accurate integration of the currents. The density of the grid depends on the directions in which the field shall be calculated, therefore it must be specified where these field points are situated. In the present case, the field shall be calculated in the far field as specified by the object SINGLE_CUT, and only here. The accuracy of the field calculation is as default set to FIELD ACCURACY -80.0 dB. It shall be mentioned that the convergence test only assures the specified accuracy in a sparse grid of field points compared to the specified grid and therefore may result in a degraded accuracy at points not included in the sparse grid. A detailed description of the PO integration may be found in the GRASP Technical Description as well as in the description of the command GET CURRENTS.

TIP: To open the Reference Section on a specific command: Double click the command in the COMMAND EDITOR and then press F1 or click at the field HELP.

TIP: It is possible to see all arguments of a command by right clicking the field in the column ARGUMENTS in the main window and then choosing EXPAND ARGUMENTS.

The second command is GET FIELD. It can be seen from Figure 4-15 that this command takes as input SINGLE_PO and SINGLE_FEED, i.e. by this command the field contribution from the currents on the reflector and from the feed are determined and added. The calculated pattern is stored in the target SINGLE_CUT. There are no attributes to this commands and opening the command in the COMMAND EDITOR gives no further information.

We are now ready to execute the calculations.

4.3 The Jobs Window

The next tab to the left in the GRASP window is JOBS. The Jobs Manager shows the jobs that have been run and the ones that are running. Since no jobs have been run yet, the window is empty, see Figure 4-17

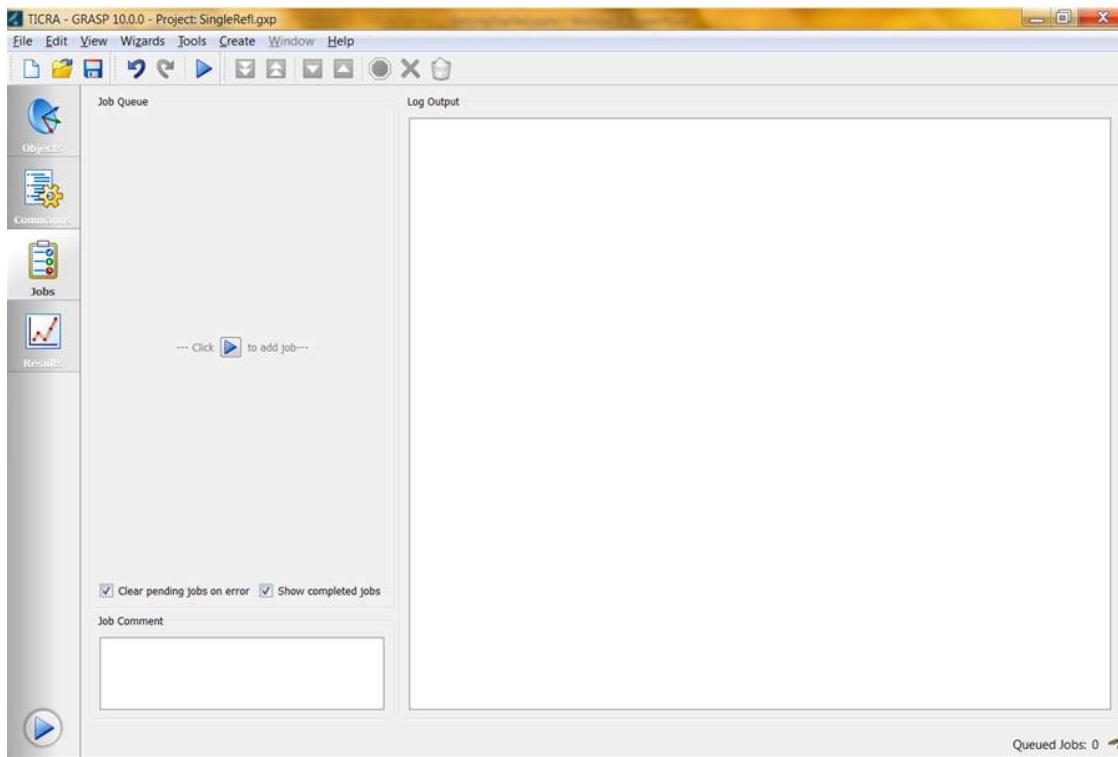


Figure 4-17 The JOBS window for monitoring the jobs in GRASP.

Clicking the blue right-pointing arrow (as a 'play' button) submits the job with the given objects and the given commands. A job may at any time be submitted by clicking this arrow in one of the menu bars.

TIP: Right-clicking on a job allows one to delete the job from the jobs queue and the corresponding results contained in the Results Window. This can also be done by clicking on the red cross tab on the upper bar.

TIP: The recycle bin icon on the upper bar allows one to delete all jobs in once.

When a job is submitted, a sub-window pops up where the name of the job, here JOB_01, is written and a description can be given, see Figure 4-18. After clicking OK, the analysis will start. During the execution, the runtime information is written in the window named LOG OUTPUT, see Figure 4-19. The job is also listed in the window JOB QUEUE, and the comments associated to it are shown below in the window JOB COMMENT.

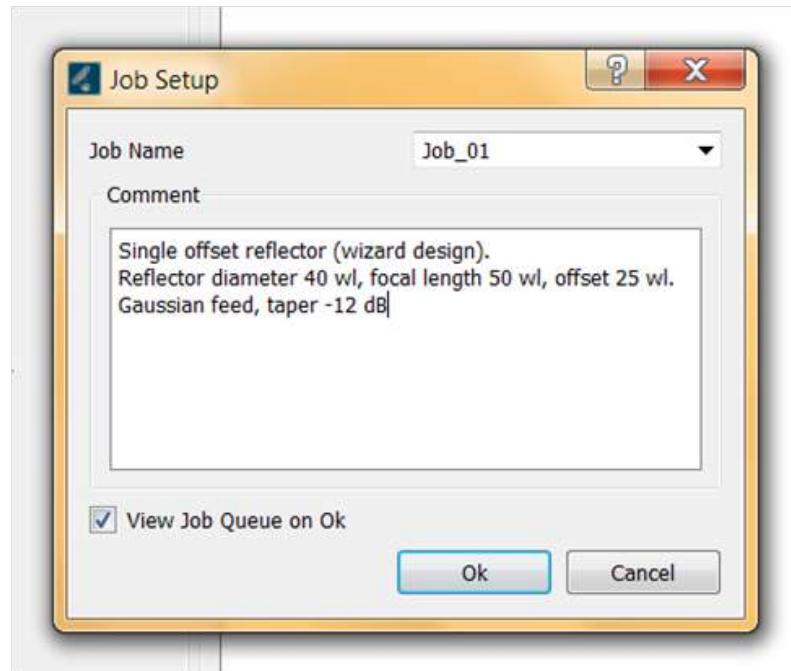


Figure 4-18 The JOB SETUP window with the comment describing the job (input given by the user).

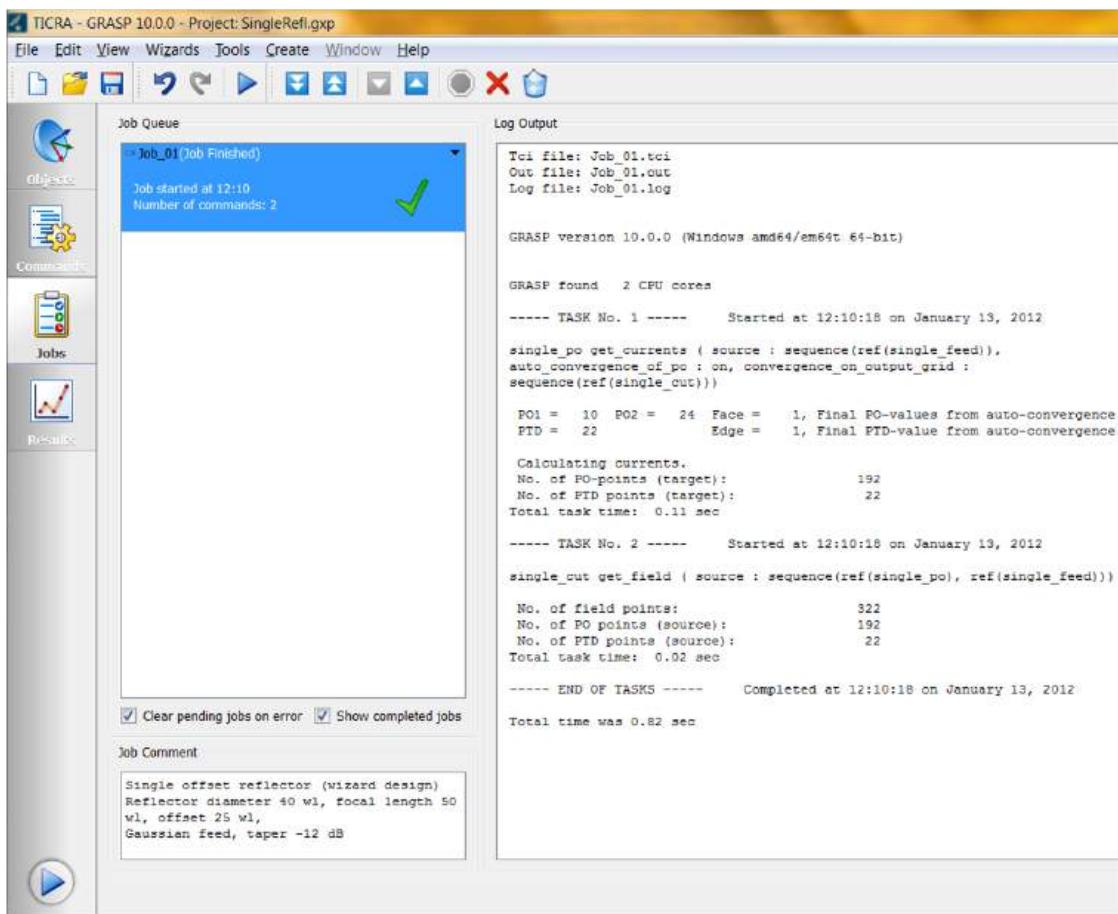


Figure 4-19 The JOBS window with screen output from the execution of the first job.

In order to show as many features as possible, we launch another job, JOB_02, with a feed taper of -18 dB. We thus go back to the OBJECTS editor and open the object SINGLE_FEED, see Figure 4-9, and set the feed taper to -18 dB. A new job, JOB_02, is thus launched and the COMMENT '-18 dB feed taper' is inserted.

The obtained results will be inspected in the following section.

4.4 The Results Window

Clicking the tab RESULTS, we access the Results window. Here all results can be plotted and post-processed. In addition, the Results window may be used to delete Jobs or to revert to a previous configuration. The Results Window is divided into three parts:

- A tree view to the upper left, in the so-called RESULTS EXPLORER. In the tree view results are sorted by the job to which they belong. Each entry in the tree contains a number of sub-entries depending on the job. Double-clicking on the entry leads to a plot or a listing of the calculated result. Other features are available from the associated right-click-menus.

Also results from batch runs may be displaced in this window, see Chapter 7 and Section 4.4.5 for details.

- A window in the lower left, the so-called JOB COMMENT window. Here the comments inserted before running the job are visualized for each job.
- A plot canvas to the right. The plot canvas may contain an unlimited number of plots, with results from the present job, or comparison of results from different jobs. The plots are saved to the project files, and consequently are re-plotted when the project is re-opened.

By right-clicking on a job name, it is possible to:

1. REVERT TO, which will load the objects and commands used in the selected job and use these as the active set-up.
2. DELETE JOB, which will delete the calculated results from the disk and remove the selected entry from the Results Explorer.

4.4.1 Job status output

For each job in the RESULTS EXPLORER we also find a section denoted JOB STATUS OUTPUT. Expanding this section we find three sets of information which may be inspected in separate sub-windows:

- JOB COMMENTS showing the user specified comment to the job. Comments can be changed, if necessary, by double-clicking on JOB COMMENT.
- ADDITIONAL JOB OUTPUT showing detailed output from the execution of the job. The first part concerns initialization of the objects and is followed by results from the execution of the individual commands (each starting with 'TASK No.'.). Some commands present important data when executed. In the present case results from the PO auto-convergence (generated by the command GET CURRENTS) may be found as 'TASK No. 1'.
- SCREEN OUTPUT where the log output shown in the Job Manager is shown again.

4.4.2 One dimensional .cut files

The Results Window for our single reflector case is shown in Figure 4-20, where we have marked JOB_01 in the RESULTS EXPLORER and clicked EXPAND BELOW (icon in the menu bar, or in the VIEW menu).

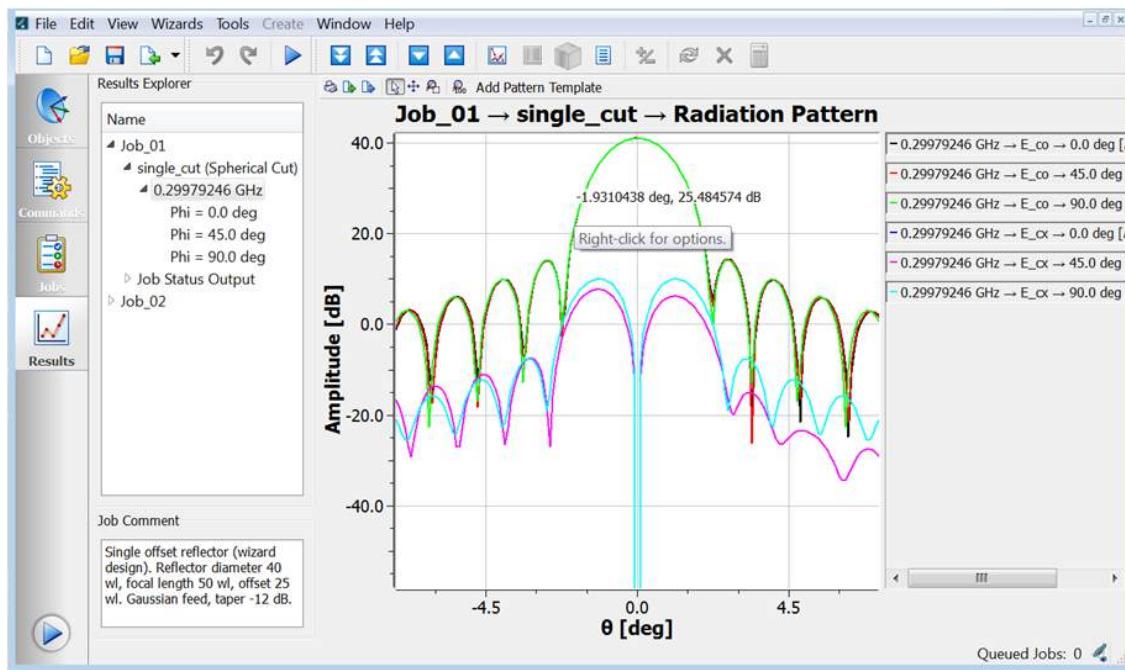


Figure 4-20 The RESULTS window for the SINGLE_REFLECTOR_CASE.

The tree in the RESULTS EXPLORER shows that we have results for the object SINGLE_CUT, which is the object in which we had specified the output range for the pattern cuts. We have a pattern for one frequency, 0.299.. GHz, on three phi cuts, i.e. $\phi = 0.0^\circ$, 45.0° and 90.0° . The cuts can be plotted individually, or together in the same window, by double clicking the entry frequency 0.29979246 GHz.

Patterns are shown in the plot canvas to the right, see Figure 4-20. The curves show the co-polar patterns (with legend E_CO) and the cross-polar pattern (with legend E_CX). It is noted that the cross-polar component is zero for $\phi = 0.0^\circ$.

TIP: The different parts of the plot may be edited and manipulated by right-clicking at a curve, at the canvas, at the axes, at the text along the axes or at the legend area in the right side.

TIP: By right-clicking at a curve it is possible to normalize the peak to 0, show the peak value, as well as cut, copy, paste and export the curve.

By enabling the peak value we find a value of 41.03 dB, while by hovering the cursor over the first side lobe we read 2.7° and 14.5 dB. This means that the side-lobe level is -26.5 dB relative to the peak.

TIP: You may zoom in by means of the zoom button RECTANGLE ZOOM MODE in the menu line of the plot window. Remember to reset the cursor to SELECT MODE (menu button with a wide arrow) in order to see the position of the cursor in the window.

As mentioned in Section 4.3 a JOB_02 with a decreased feed taper was executed. By expanding JOB_02 in the RESULTS EXPLORER and double clicking the frequency we get the patterns for the -18 dB feed taper. We can also double click on PHI=0.0 DEG for JOB_02 and later on PHI=0.0 DEG for JOB_01, to compare the cuts. We obtain in this way two plots. In the plot of JOB_01 we right click on the plotted curve and change the colour to blue, using the menu Properties. We then right click on the blue curve and choose COPY. Later, we right click on the canvas of the Job_02 plot and choose PASTE, whereby the curve for the -12 dB taper is pasted into the figure and may directly be compared to the curve for -18 dB taper. We obtain in this way Figure 4-21, where it is seen that the decreased feed taper used in JOB_02 produces a lower side-lobe level relative to the pattern obtained in JOB_01. The legend text can be changed by right-clicking on the plot legend.

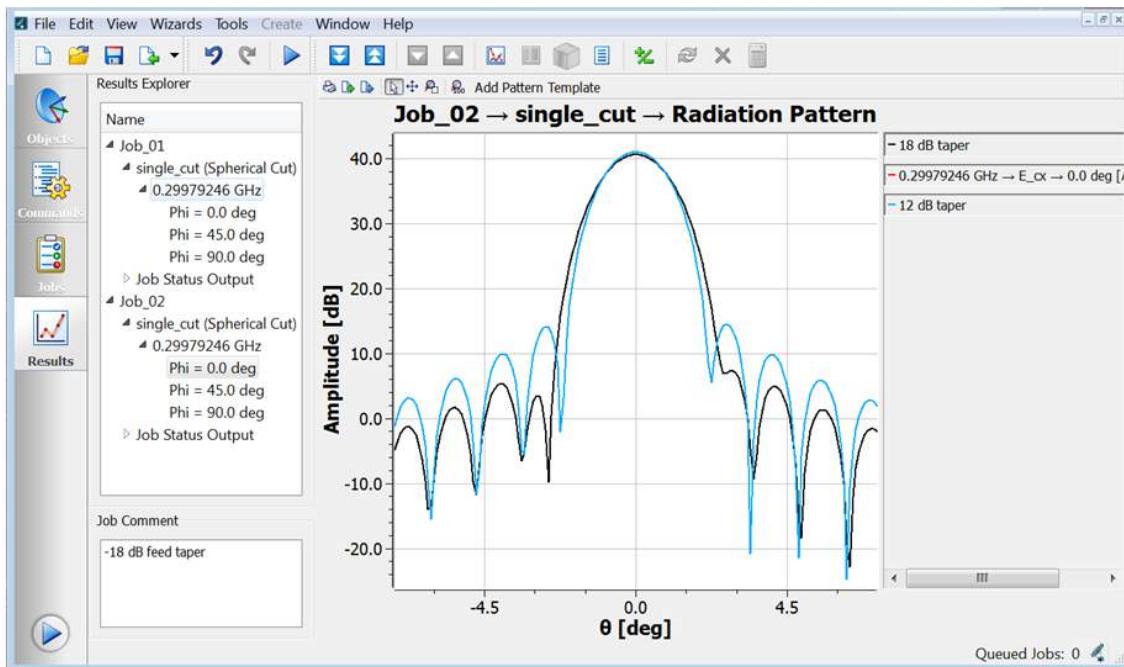


Figure 4-21 The RESULTS window showing the amplitude result at $\phi=0^\circ$ for -12 dB feed taper (blue curve) and -18 dB feed taper (red curve).

TIP: The curves have each a legend to the right. If we click at a legend the corresponding curve is hidden.

The displayed results are by default the amplitude patterns. Phase patterns may be shown by right-clicking an entry in the tree of the RESULTS EXPLORER.

Curves may also be added or subtracted. To do that, first right click one entry (data for one curve) in the RESULTS EXPLORER and choose ADD/SUBTRACT... in the menu bar (the green "+/-" icon). The second entry is thus chosen. It is the complex field values - in amplitude and phase - which are added or subtracted.

It is finally possible to add other data sets to an existing plot, by double-clicking on the top tree level of a cut, i.e. one step above the frequency field. Alternatively, this is possible by right-clicking on the top tree level of a cut

and choosing PLOT 2D CURVE. An example is shown in Figure 4-22. There we can see that we have two plots open, i.e. Job_03 and Job_05. Suppose we now want to plot the .cut file computed in Job_01 in the plot of Job_05. In order to do that we activate the PLOT SETUP window, as described above, and choose Job_05 in the field ON.

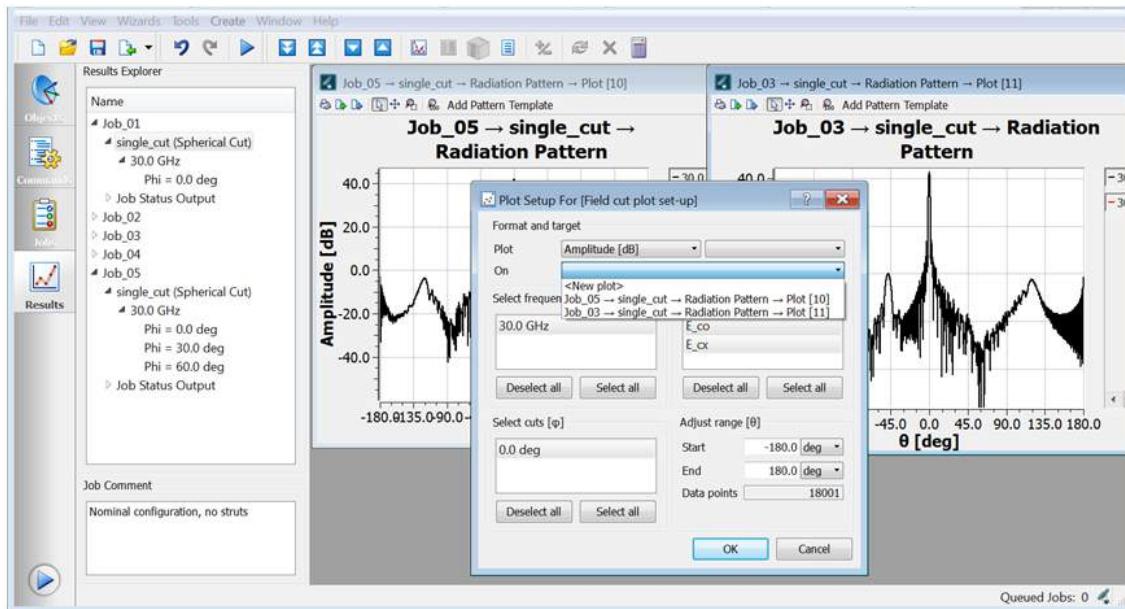


Figure 4-22 The PLOT SET UP window.

4.4.2.1 Pattern template

It is possible to add a pattern template to the plot of a certain .cut file. This is easily done by clicking on the ADD PATTERN TEMPLATE tab in the upper bar of the plot. Once this is done, the user must create the pattern template, by choosing between a piecewise linear and a piecewise logarithmic template. An example is given in Figure 4-23 where a piecewise logarithmic template is defined and added to a pattern plot.

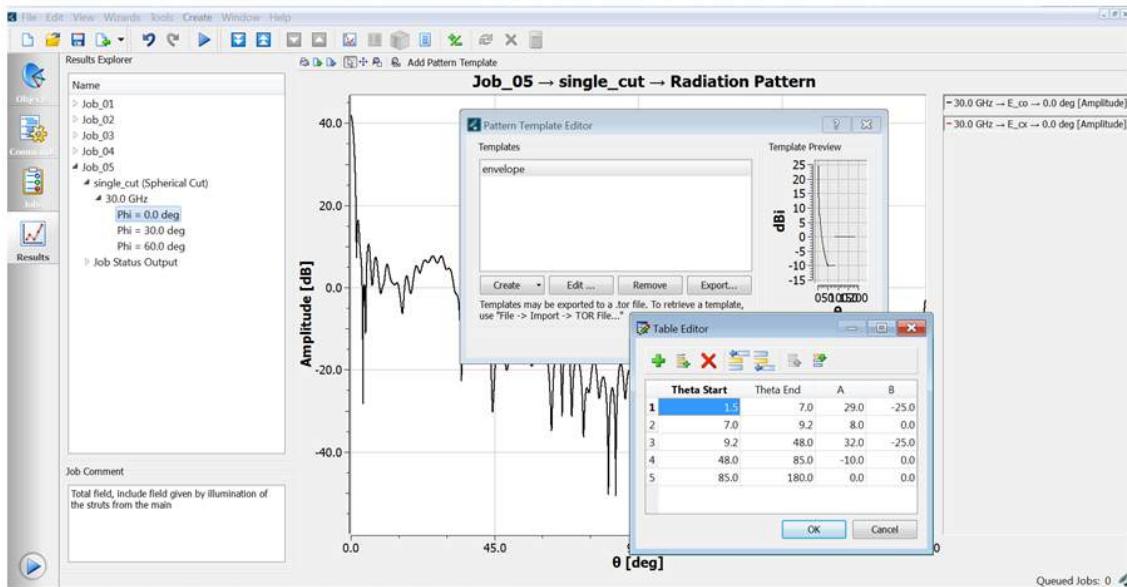


Figure 4-23 The PATTERN TEMPLATE EDITOR for a piecewise logarithmic template.

The template is shown in red in Figure 4-24. Once the template is created, it is then possible to EDIT and REMOVE it or to EXPORT the template into a .tor file. Once exported into a .tor file, the template can be later on imported by clicking on FILE> IMPORT> TOR FILE.

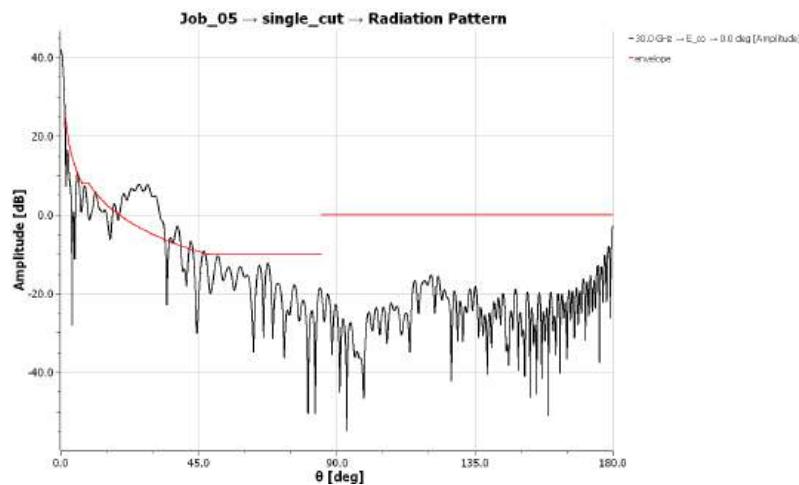


Figure 4-24 Radiation pattern with added template.

4.4.3 Two dimensional .grd files

The Results Window of the tutorial example DUAL_GRIDDED_REFLECTOR is shown in Figure 4-25.

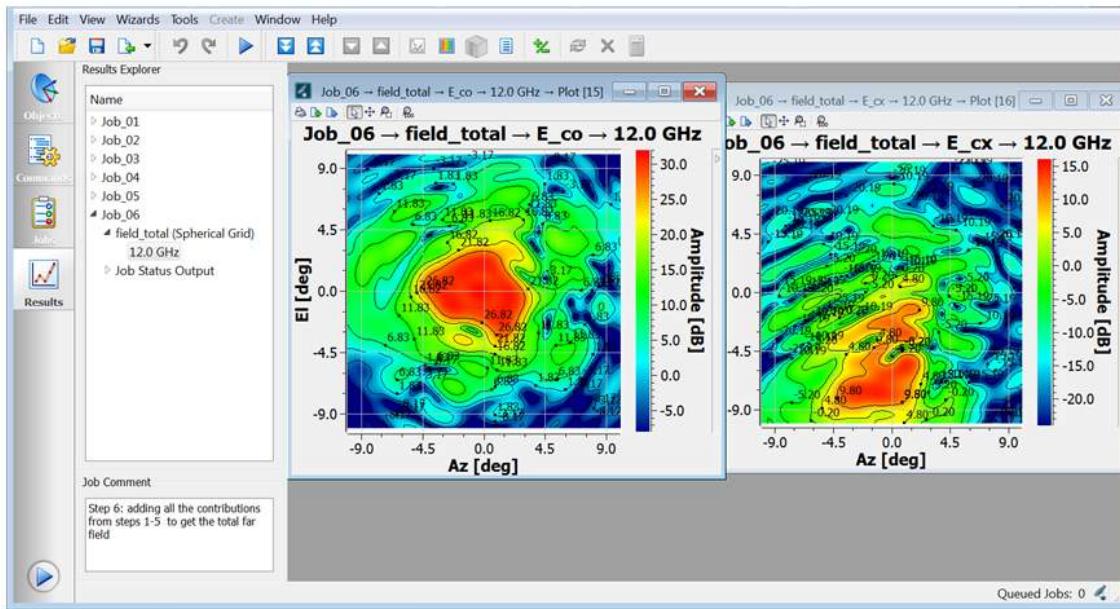


Figure 4-25 A typical 2D plot of a .grd file: the image mode is plotted together with the contour plots.

Two dimensional .grd files, for example the one computed in Job_06, are plotted by double clicking on the entry of interest in the Results Explorer. When this is done for the first time, both the image mode and the contours are plotted, as it is seen in Figure 4-25. Right-clicking on the plot gives the possibility of changing general as well as beam specific features, as depicted in Figure 4-26 and described more in detail in the following.

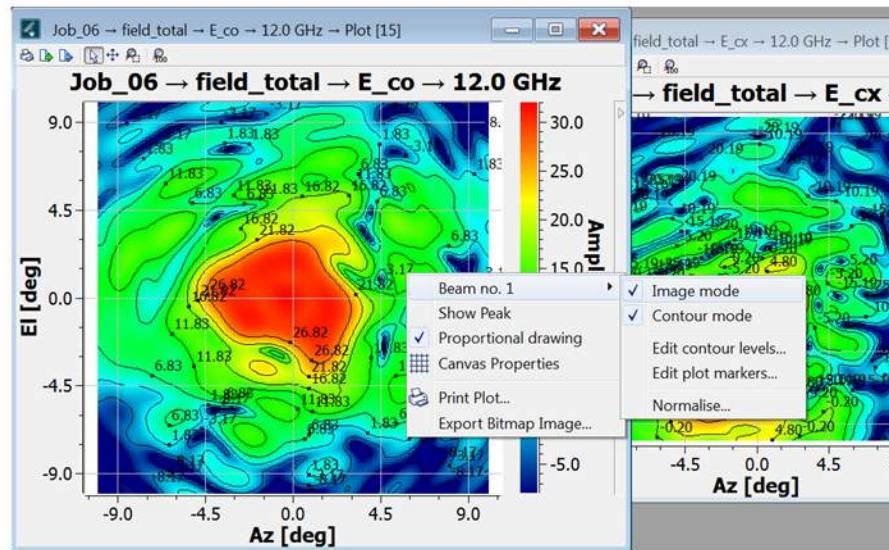


Figure 4-26 Activating general as well as beam specific features of a .grd plot.

For each beam, it is possible to choose between IMAGE MODE and CONTOUR MODE, as well as EDIT CONTOUR LEVELS and EDIT PLOT MARKERS. The displayed contour levels can be specified by tabulated values or equidistant

values. Both can be considered absolute values or values relative to a peak or a given level. Different symbols and styles are available for peak markers and contour markers.

Right-clicking on the plot allows to SHOW PEAK and enable PROPORTIONAL DRAWING, which means that the length of units are the same along *x* and *y* in the drawing.

Grids may also be added or subtracted, as it is done with cuts. To do that, first right click one entry (data for one plot) in the RESULTS EXPLORER and choose ADD/SUBTRACT... in the menu bar (the green "+/-" icon). The second entry is thus chosen. It is the complex field values - in amplitude and phase - which are added or subtracted.

It is finally possible to add other data sets to an existing plot, as it is done with cuts, by double-clicking on the top tree level of a grid, i.e. one step above the frequency field. Alternatively, this is possible by right-clicking on the top tree level of a grid and choosing PLOT GRID.

4.4.4 Postprocessing

The Postprocessing icon is available in the upper bar of the Results window. Alternatively, it can be accessed by right-clicking on the top tree level of a cut or grid file in the Results Explorer.

The features available at the moment are:

- Cubic interpolation between the points of a cut or grid plot
- Convert polarization
- Apply an offset in dB to the amplitude values
- Shift the phase by a certain amount of degrees

4.4.5 Import of files

Different types of data files can be imported into GRASP, see 4-27. This is possible by clicking on FILE> IMPORT or directly on the IMPORT icon in the upper bar tab. Files of format .tor and .tci are imported and shown in the Objects and Commands window. DATA FILES and INDEX FILES are imported and shown in the Results window. This will be further described in the following.

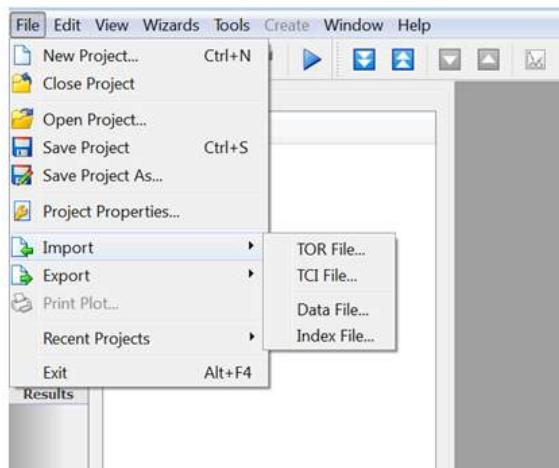


Figure 4-27 The IMPORT icon.

4.4.5.1 Import of data files

Files of type .cut and .grd can be imported into the present project. Once the file that has to be imported is selected, the TICRA DATA IMPORT WIZARD will show the type of file, number of frequencies in the file, beams, and type of polarization, and a default frequency of 0 GHz, see Figure 4-28. Once the wizard is closed, the imported file will appear in the Results Explorer, at the bottom, as shown in Figure 4-29. From there, the file will be plotted as it is done with all .cut and .grd files. The imported file can be deleted by right-clicking and selecting DELETE JOB.

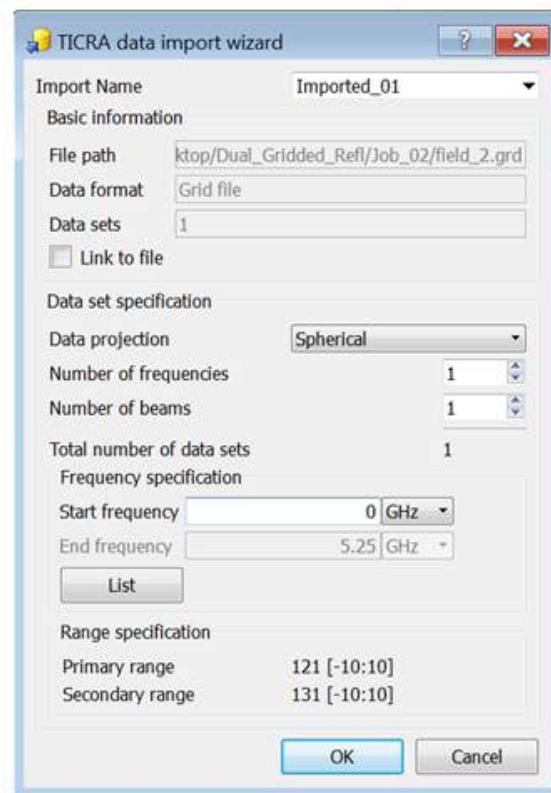


Figure 4-28 The TICRA DATA IMPORT WIZARD.

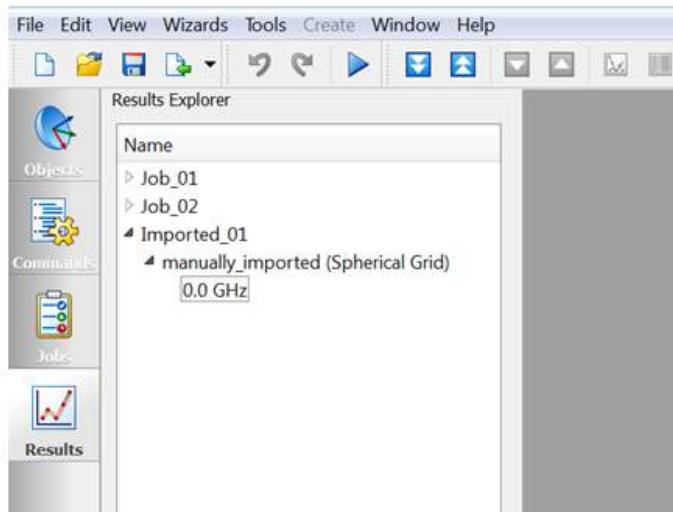


Figure 4-29 The imported data file in the Results Explorer.

4.4.5.2 Import of index files

All files of type .xml which are generated in each job can be imported. By doing that, all data files, frequencies and Job Status Output generated in the job are imported in the project, as shown for example in Figure 4-30.

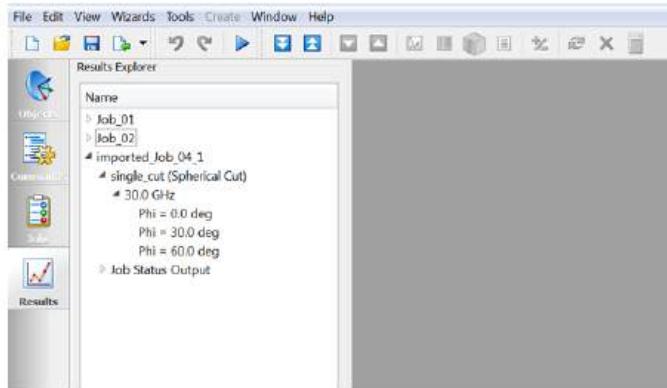


Figure 4-30 The imported index file in the Results Explorer.

5. File Organization

GRASP saves a project in a file structure that requires no user intervention. However, some information on the organization of the files may be useful and will be described in the following.

Let us consider the previously saved project SINGLEREFL (cf. Section 4). This project was saved in a folder named SINGLEREFL containing a sub-folder for each submitted job, i.e. JOB_01 and JOB_02. Further, we find the folder WORKING and the GRASP project file, SINGLEREFL.GXP, which holds all data describing the latest version of the project.



Figure 5-1 All information on the project SINGLEREFL is stored in the folder SINGLEREFL.

When the project is **saved**, the defined objects and the commands are saved in a TOR-file (TICRA Object Repository) and a TCI-file (TICRA Command Interface) named by the project (SINGLEREFL.TOR and SINGLEREFL.TCI, respectively) in the sub-folder WORKING.

When a job is **executed**, the execution takes place in WORKING, where all input and output files are located. If a file exists, it will be overwritten, i.e. it is only the last instance which is kept in the sub-folder WORKING.

When the execution of a job is finalized, the corresponding TOR-file and TCI-file are stored along with all output files in a sub-folder named by the job name (JOB_01 etc.)

The output is similarly stored in a LOG-file and an OUT-file (referred to as the standard output file). The contents of these files can be seen in the GUI under the RESULTS-tab as SCREEN OUTPUT and ADDITIONAL JOB OUTPUT, respectively.

When the user **reverts to a previous job** then the objects and commands used to generate the previous job are restored. If objects or commands refer data files then it is the existing files in the sub-folder WORKING which are applied. The output files from the reverted job are not copied to the sub-folder WORKING.

6. Tutorial Examples

A number of tutorial examples will here be presented to highlight features and capabilities of GRASP. The antenna configurations represent cases which are commonly encountered in practice. The project files associated to these examples are located in the TESTCASES folder in the GRASP installation directory. Projects files can also be opened by clicking on TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window.

It is noted that some of the examples also appear in the GRASP Technical Description where they are described from a more theoretical point of view. For those examples, the attention is here given to objects and commands definition.

The following test cases are available:

Contents

6.1	Single Reflector with Three Struts of Polygonal Cross-Section	45
6.2	Dual Reflector with Blockage	61
6.3	Single Shaped Reflector with Circular Polarisation	73
6.4	Dual Gridded Reflector	80
6.5	Single Reflector with Feed Array	97
6.6	Compact Antenna Test Range with Serrated Edges	103
6.7	Plane Wave Expansion	110
6.8	Scalable Dual Reflector Using Real Variables	122
6.9	Offset Reflector in Radome	129
6.10	Dual Reflector with Panels and Subreflector Support Struts	142

6.1 Single Reflector with Three Struts of Polygonal Cross-Section

This tutorial case deals with the analysis of a single reflector with a circularly symmetric paraboloidal surface illuminated by a feed horn located at the focal point of the paraboloid. The feed is held in place by three identical struts of polygonal cross-section. The example will show how to compute the scattering from the antenna and the struts. The GRASP project can be opened by clicking on TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The project files are located in the TESTCASES/SINGLE_REFLECTOR_WITH_THREE_STRUTS sub-directory in the installation directory.

6.1.1 Geometry

The antenna operates at 30 GHz and has the following geometrical characteristics:

reflector diameter : 50 cm
 reflector focal length : 25 cm
 half-angle from the feed to the reflector edge: $= 53.1^\circ$

The three struts are 25 cm long and around 1 cm wide. Each strut is defined in its own coordinate system, where the z -axis coincides with the strut axis. The strut cross-section in the xy -plane of the strut coordinate system is depicted in Figure 6-1.

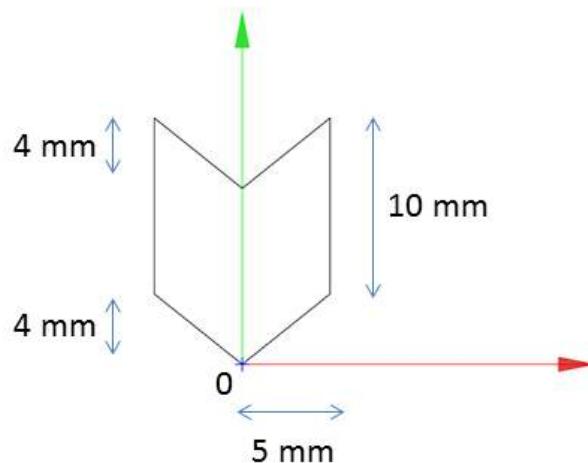


Figure 6-1 Cross-section of the strut in the xy -plane of the strut coordinate system.

A drawing of the antenna geometry is shown in Figure 6-2.

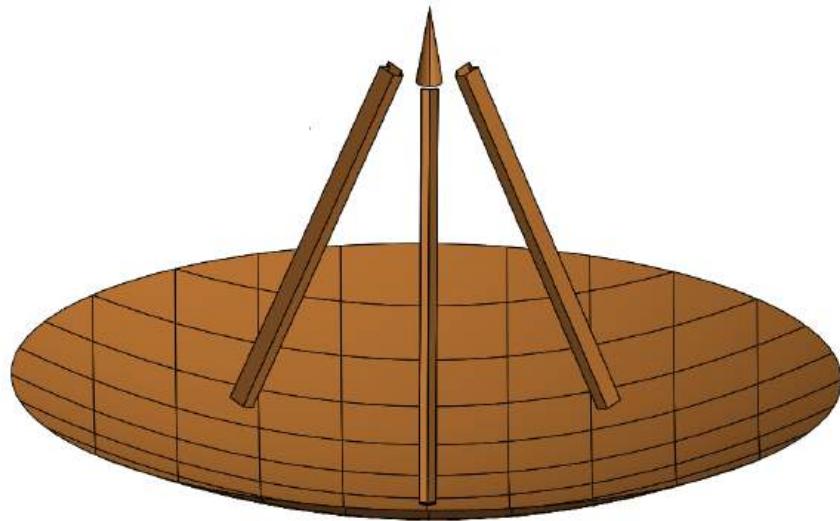


Figure 6-2 The single reflector with three struts.

In the following sections the effect of the strut scattering will be investigated. To make the analysis clear, the following computations will be presented:

1. The feed illuminates the reflector, and no struts are present (nominal case, Job_01).
2. The feed illuminates one strut. The induced currents on the strut illuminate the reflector, (spherical wave scattering, Job_02).
3. The feed illuminates the main reflector. The induced currents on the reflector illuminate one strut, (plane wave scattering, Job_03).
4. All three struts are included in the analysis. The struts are illuminated by the feed, and these induced currents together with the feed illuminate the reflector. The total field is computed, (spherical wave scattering with all struts, Job_04).
5. All three struts are included in the analysis. The currents on the main reflector computed in Job_04 illuminate the struts. The field generated by the currents induced on the struts from the reflector are added to the field computed in Job_04. The total field is computed, (spherical wave scattering and plane wave scattering with all struts, Job_05).

6.1.2 Nominal Case

The antenna geometry is defined in GRASP by means of the wizard for single reflector using the data in Section 6.1.1, as it was done in Chapter 3, see Figure 6-3.

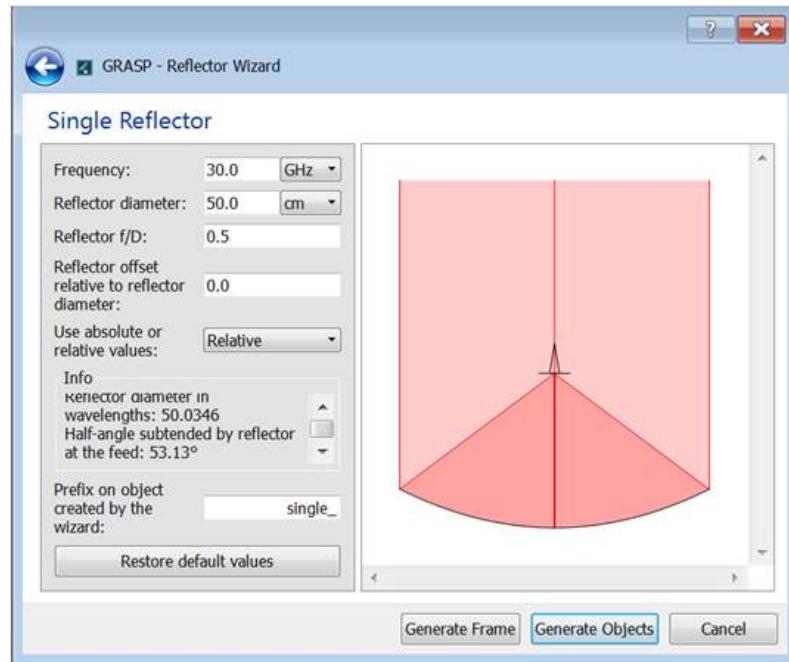


Figure 6-3 GRASP wizard for the single reflector.

After having closed the wizard and created the objects, we modify the field storage object SINGLE_CUT, in order to have $\pm 180^\circ$ in one cut at $\phi=0^\circ$ with 18001 points. The commands that carry out the required analysis were created by the wizard, i.e. a GET CURRENTS and GET FIELD command like in Section 4.2, see Figure 6-4 where the arguments have been expanded by right-clicking.

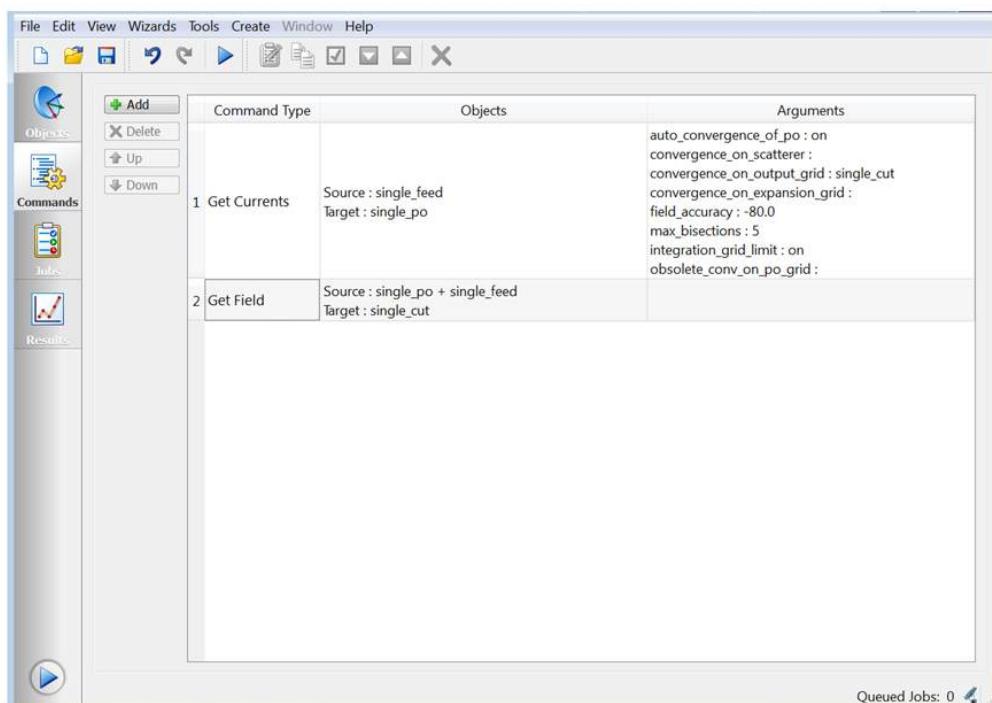


Figure 6-4 Commands created for the nominal case.

The number of PO patches are found by the automatic internal convergence procedure using the far field points in the `SINGLE_CUT` object as convergence criterion and requiring a field accuracy of -80 dB. We run these commands under Job_01 and obtain in the Results tab the pattern cut shown in Figure 6-5. We can observe that the pattern is symmetric, as expected, and that the peak directivity is at 42.8 dBi. The two peaks around $\pm 120^\circ$ are due to the feed spillover, past the reflector edge.

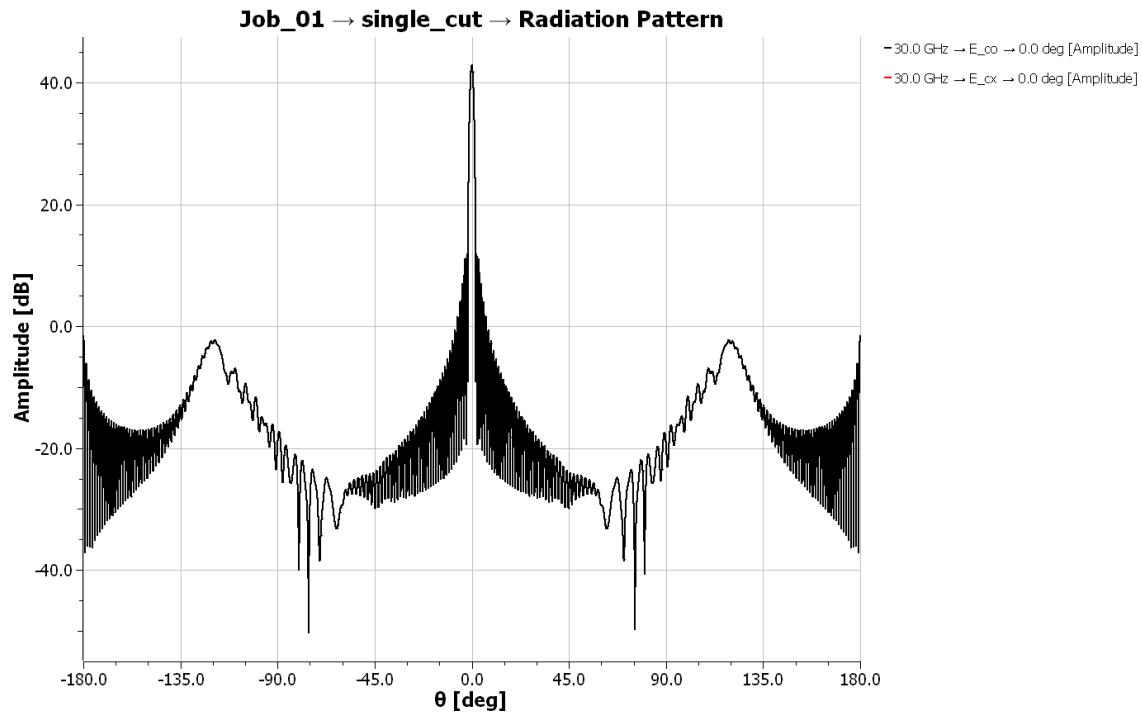


Figure 6-5 Nominal radiation pattern for the front-fed paraboloid, without struts.

6.1.3 Spherical Wave Strut Scattering

We add now one strut to the reflector geometry and study the spherical wave scattering effect. We start by defining the strut coordinate system. The strut coordinate system will be defined in the global coordinate system, which has origin at the center of the reflector and z -axis pointing towards the feed. The origin of the strut coordinate system is located at a point displaced 12 cm along the x -axis and 2.2 cm along z . The origin of the strut coordinate system coincides with the origin depicted in Figure 6-1 and is located at the bottom face of the strut. The orientation of the strut coordinate system relative to the global coordinate can be seen in Figure 6-6.

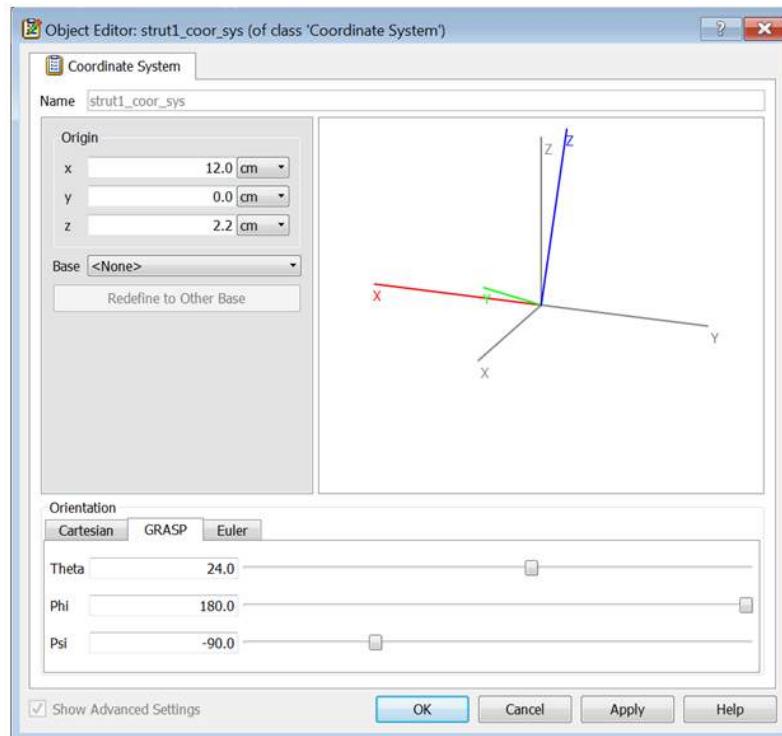


Figure 6-6 Definition of the coordinate system of the first strut.

Having defined the strut coordinate system, we can define the strut object. It is a scatter object of type POLYGONAL STRUTS which we will call STRUT1. Its POSITION is given by the STRUT1_COOR_SYS defined above, and by a z_1 and z_2 value of 0 cm and 25 cm, respectively. The input values for the strut cross-section can be seen in Figure 6-7. The values coincide with the nodes depicted in Figure 6-1.

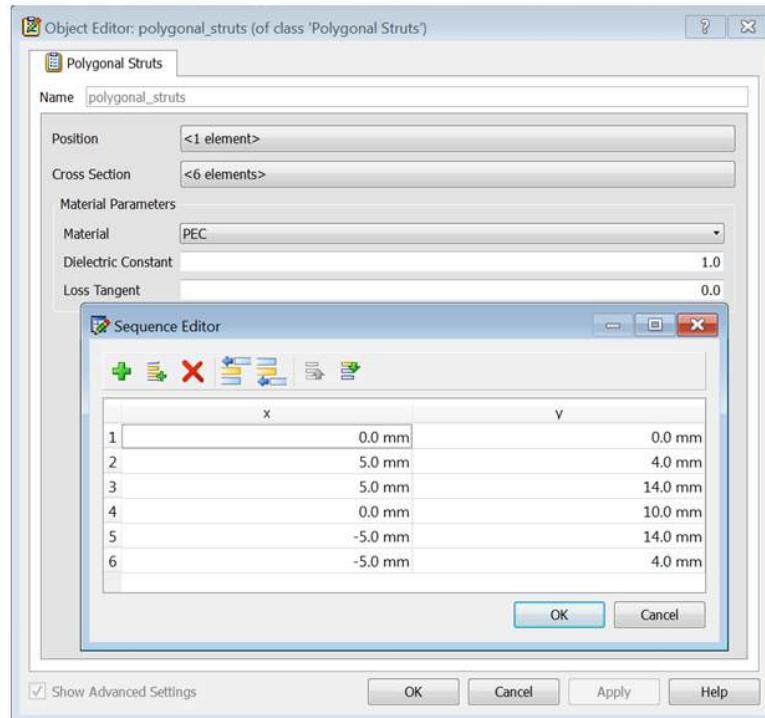


Figure 6-7 Definition of the object for the first strut.

The spherical wave strut scattering is illustrated in Figure 6-8, by adding in the 3D-view setting a VISUALIZATION OBJECT of type RAYS FROM POINT SOURCES. The field from the feed hits first the strut. Most of this scattered field hits then the reflector and is partly reflected in a direction near the main beam.

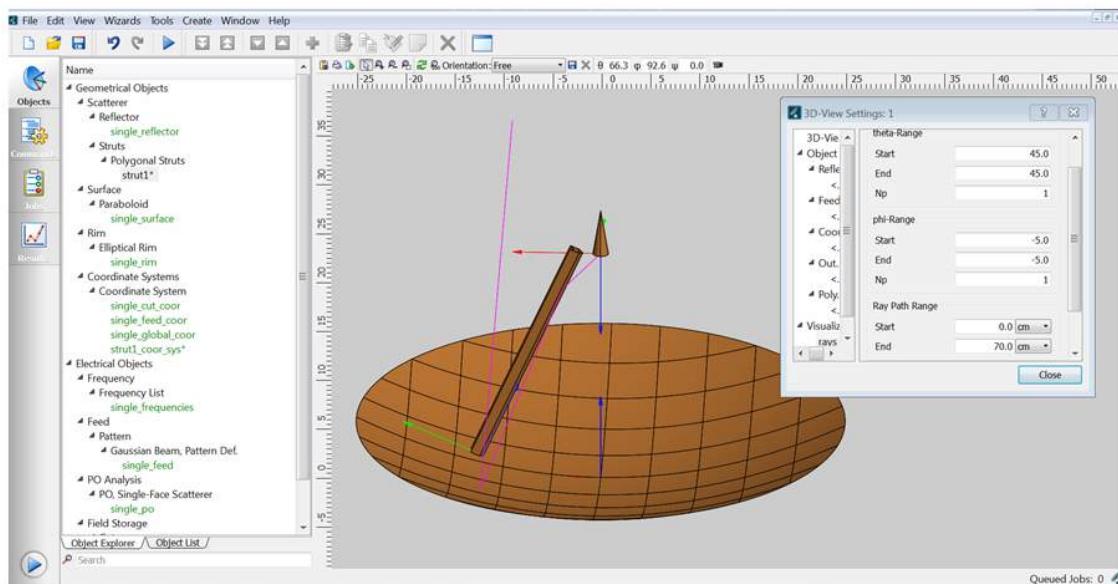


Figure 6-8 Spherical wave strut scattering.

In order to compute the scattering from the strut, we need to define an electrical object of type STRUT ANALYSIS, by choosing STRUT ANALYSIS, ARBI-

TRARY CROSS SECTION. We define the FREQUENCY and STRUT references, and accept the default values for the other entries, as shown in Figure 6-9. It is noted that the STRUT ANALYSIS object will use Physical Optics (PO) in the direction of the strut length, and Method of Moments (MoM) around the strut cross section. Automatic convergence is used for PO, while for MoM the convergence has to be checked by the user, by increasing the EXPANSION ACCURACY by one and decreasing the MAX MESH LENGTH. More can be read in the GRASP Reference Section, or by hover the mouse over the entries of the object. Since the cross section of the strut is not curved, we expect that convergence of the MoM will be reached by the default values found by GRASP. The field calculated for this case will be stored in the same spherical cut SINGLE_CUT. To compute the effect of the spherical wave scattering, the commands shown in Figure 6-10 are used.

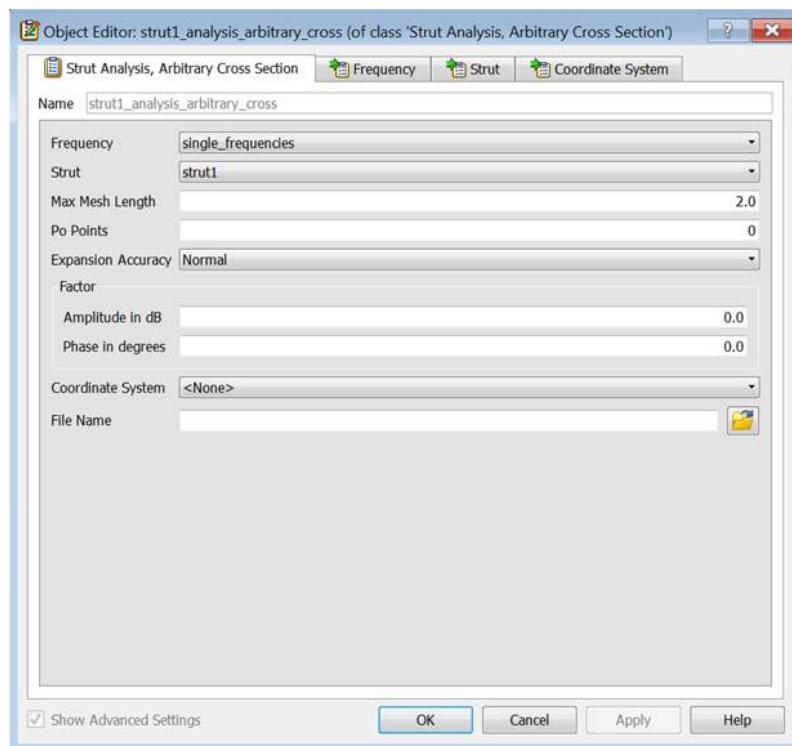


Figure 6-9 Strut analysis object for the strut.

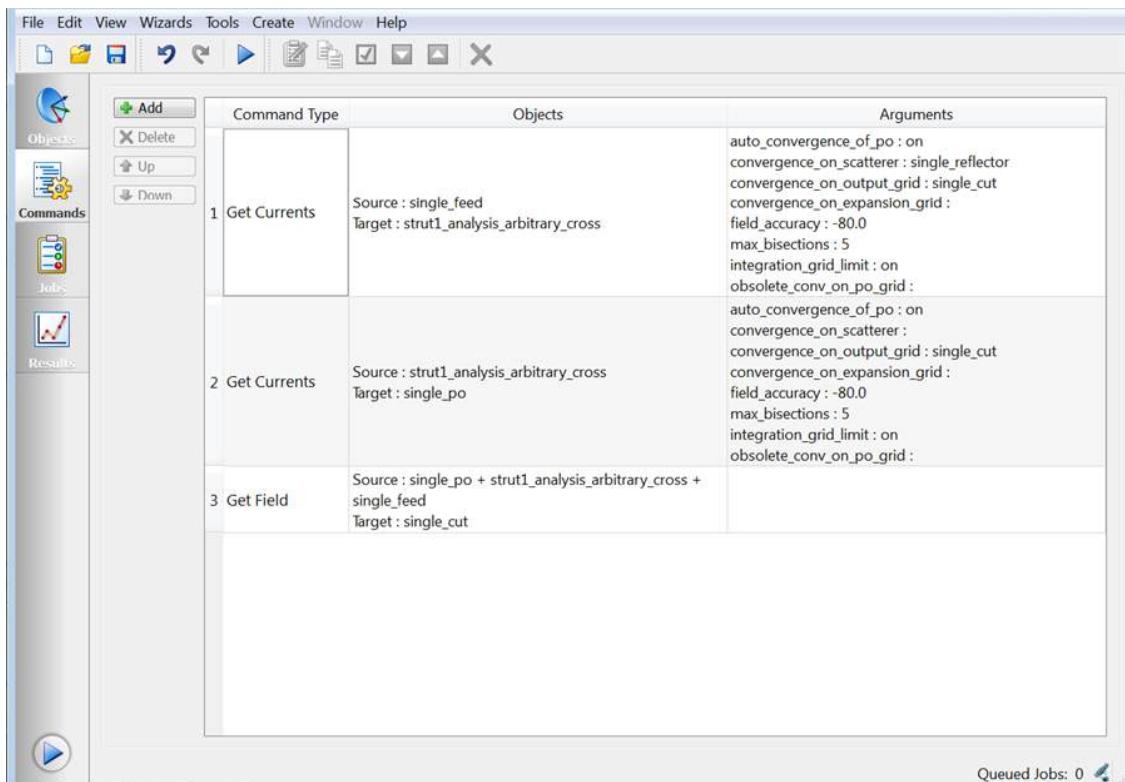


Figure 6-10 Commands for the spherical wave strut scattering.

The currents on one strut, STRUT1_ANALYSIS_ARBITRARY_CROSS, are calculated using the field from the feed as incident field. The number of PO patches are found by the automatic internal convergence procedure using the reflector and the field points in SINGLE_CUT as convergence criterion (hence, the currents will be applicable for field calculation on the reflector and for field calculation on the far field points). Remaining parameters are defaults.

The PO currents on the reflector, SINGLE_PO, are calculated using the field from STRUT1_ANALYSIS_ARBITRARY_CROSS as incident field. The number of PO patches are found by the automatic internal convergence procedure using the field points in SINGLE_CUT as convergence criterion (hence, the currents will be applicable for field calculation on the far field points).

The field given by the reflector PO currents, SINGLE_PO, is finally computed on the far field points of SINGLE_CUT, and the feed and strut field are finally added. By running these commands under Job_02 we obtain in the Results tab the plot shown in Figure 6-11

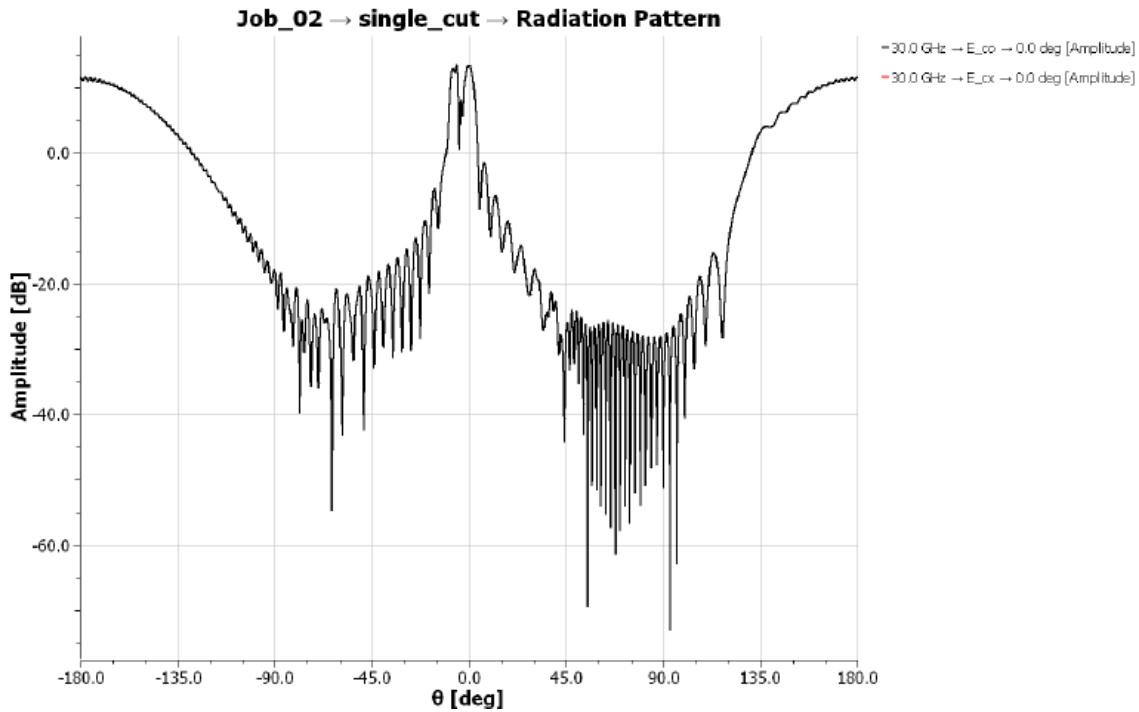


Figure 6-11 Scattered field from the main reflector due to the spherical wave strut scattering on one strut.

The figure shows a quite large disturbance, i.e. a peak around 13 dBi, relative to the nominal pattern which had a peak directivity of 42.8 dBi. There are two peaks, one at 0° and one around -7° . The latter is due to the reflection of the strut-scattered field in the main reflector, as illustrated by the ray in Figure 6-8. The peak at 0° requires a more elaborate explanation: due to the strut there will be a shadow region on the main reflector where the incident field is very low. In the present example we are interested in examining the strut effect alone and therefore we have not added the direct feed field when calculating the induced currents on the reflector. If this had been done we would have obtained a main lobe pattern similar to the one in Figure 6-5 and with increased sidelobes. Instead we observe here that the scattered field from the strut is quite large in the shadow region, comparable in size and 180° out of phase to the feed field. Consequently the reflection of this portion of the scattered field is a plane wave which peaks in the boresight direction. It can be shown that the lobe in the main beam direction is 180° out of phase relative to the nominal field of Figure 6-5 and thus will subtract from this when the total field is eventually calculated. This will be done in Job_04.

6.1.4 Plane Wave Strut Scattering

The second effect on the far field due to the struts is illustrated in Figure 6-12. After hitting the reflector the wave is scattered by the strut.

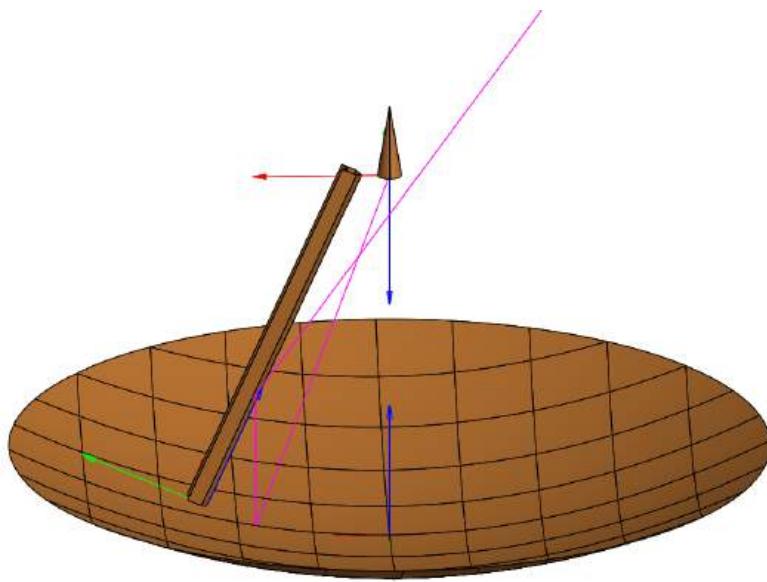


Figure 6-12 Plane wave strut scattering.

The geometry has not changed and therefore no new objects are necessary. In order to calculate the far field including this contribution, the three commands shown in Figure 6-13 shall be used.

File	Edit	View	Wizards	Tools	Create	Window	Help
	<input type="button" value="Add"/> <input type="button" value="Delete"/> <input type="button" value="Up"/> <input type="button" value="Down"/>		Command Type	Objects	Arguments		
	1 Get Currents	Source : single_feed Target : single_po	auto_convergence_of(po) : on convergence_on_scatterer : strut1 convergence_on_output_grid : single_cut convergence_on_expansion_grid : field_accuracy : -80.0 max_bisections : 5 integration_grid_limit : on obsolete_conv_on_po_grid :				
	2 Get Currents	Source : single_po Target : strut1_analysis_arbitrary_cross	auto_convergence_of(po) : on convergence_on_scatterer : convergence_on_output_grid : single_cut convergence_on_expansion_grid : field_accuracy : -80.0 max_bisections : 5 integration_grid_limit : on obsolete_conv_on_po_grid :				
	3 Get Field	Source : single_po + strut1_analysis_arbitrary_cross + single_feed Target : single_cut					

Figure 6-13 Commands for the plane wave strut scattering.

The PO currents on the reflector are calculated using the field from feed as incident field. The number of PO patches are found by the automatic internal convergence procedure using the strut and the far-field points as convergence

criterion. The PO currents on the strut are calculated using the field from the reflector as incident field and the far-field points as convergence criterion. The number of PO patches are found by the automatic internal convergence procedure using the far field points as convergence criterion.

The commands of Figure 6-13 are executed in Job_03 and the result depicted in Figure 6-14 is obtained. It is seen by comparing Figure 6-14 with the nominal field of Figure 6-5 that a sidelobe at around -48° appears, and the peak directivity decreases by 0.13 dB.

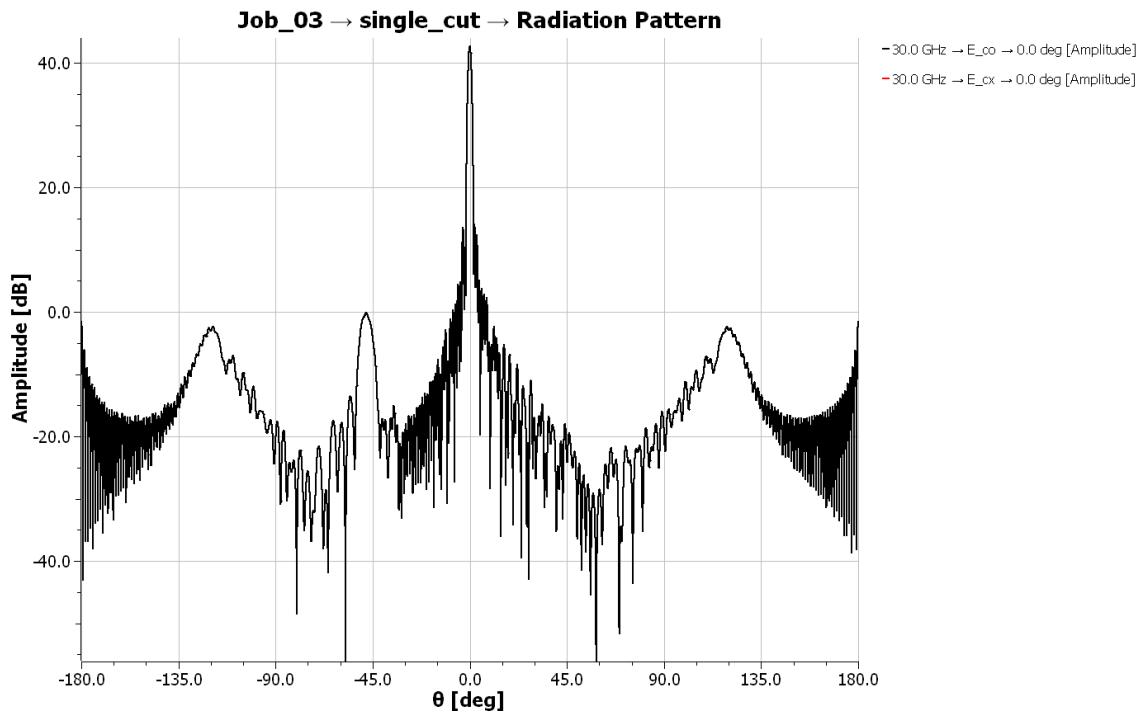


Figure 6-14 Scattered field from one strut due to the plane wave strut scattering.

6.1.5 Total Field Including Three Struts

The final step consists in calculating the total field, i.e. the field scattered by the reflector and the three struts, when both the spherical and plane wave scattering are included. For this purpose we need first of all to define the remaining two struts. Then, the coordinate systems of the two remaining struts are defined, according to Figure 6-15 and Figure 6-16. A new POLYGONAL STRUTS object including all three struts is created with the name STRUTS_ALL cf. Figure 6-17.

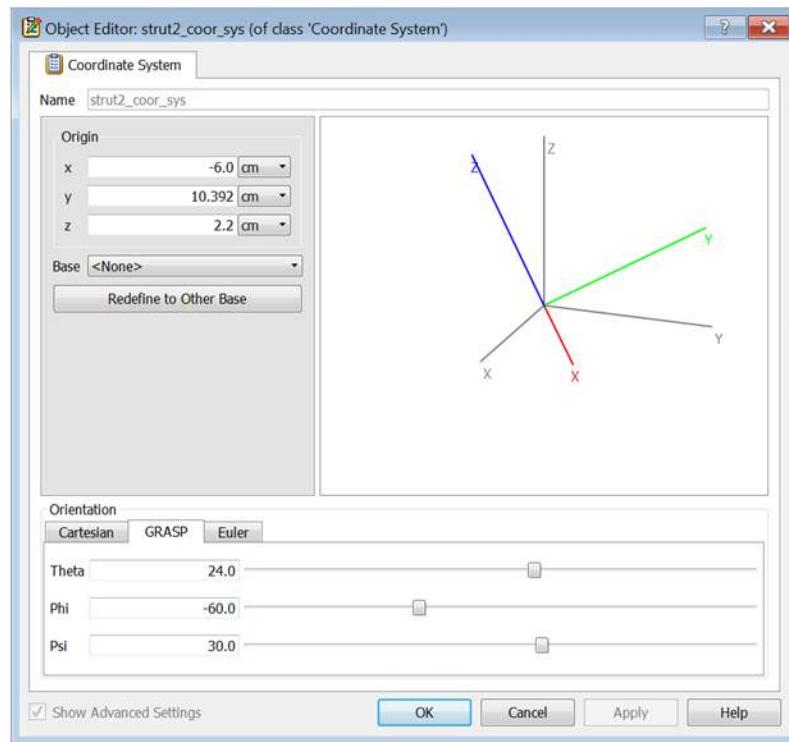


Figure 6-15 Coordinate system of the second strut.

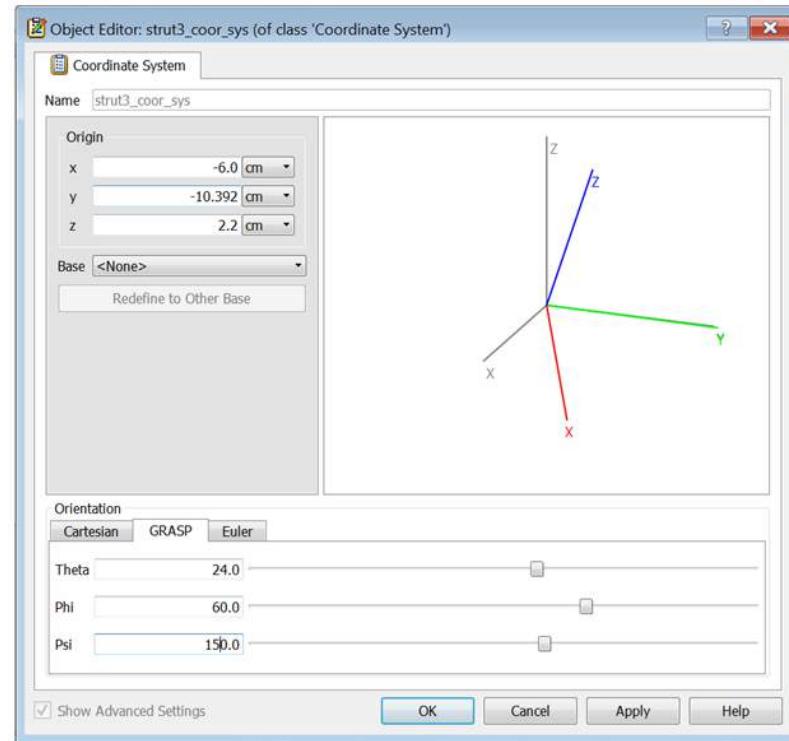


Figure 6-16 Coordinate system of the third strut.

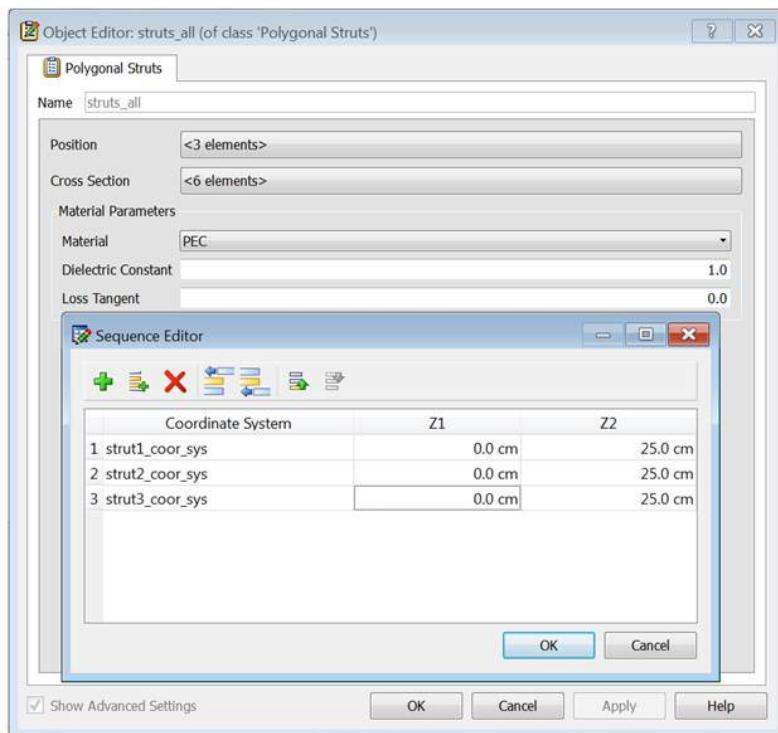


Figure 6-17 The object defining the three struts.

A STRUT ANALYSIS electrical object referring to the STRUTS_ALL object defined above is created, by renaming the existing STRUT1_ANALYSIS_ARBITRARY_CROSS with STRUTALL_ANALYSIS_ARBITRAR_CROSS and changing the STRUT reference to STRUTS_ALL.

The object SINGLE_CUT is modified, by adding two more cuts in phi, at 30° and 60° . In order to calculate the total far field, we can split the commands in two parts. First, we compute the field given by the feed, the struts illuminated by the feed, and the main reflector illuminated by the feed and struts. This corresponds to consider the so-called spherical wave scattering on all three struts, including the reflector field. The four commands shown in Figure 6-18 are therefore used and Figure 6-19 is obtained. It is noted that Figure 6-19 also contains in blue the nominal field of Job_01. We can see that the pattern is no more symmetric for $\phi=0^\circ$, and that sidelobes in the angular region $[-90^\circ : 90^\circ]$ are higher. The non-symmetry is due to the presence of the struts: the first strut creates a shadow for the field generated by the feed, which decreases the spillover for theta larger than zero. For $\phi=0^\circ$, we can still see a sidelobe around -7° , which is again due to the field scattered by the struts, and reflected by the reflector. We finally see a directivity peak loss of 0.55 dB.

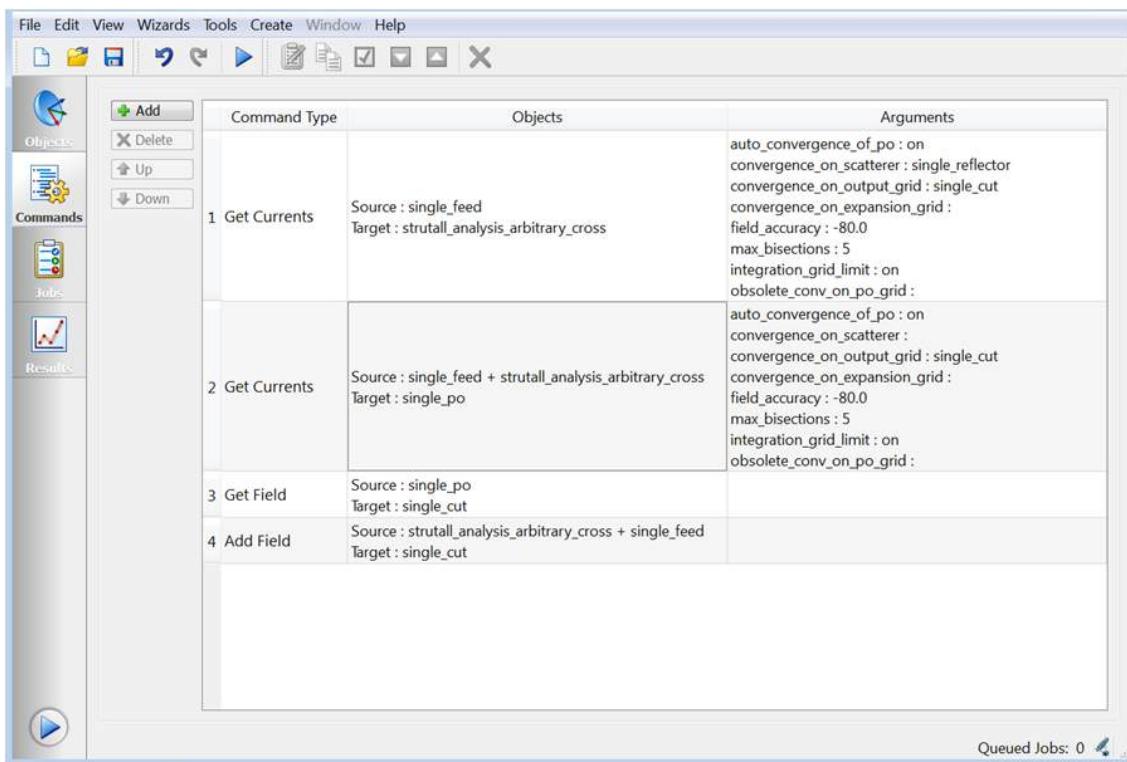


Figure 6-18 Commands for Job_04: spherical wave scattering on the three struts and field from the reflector.

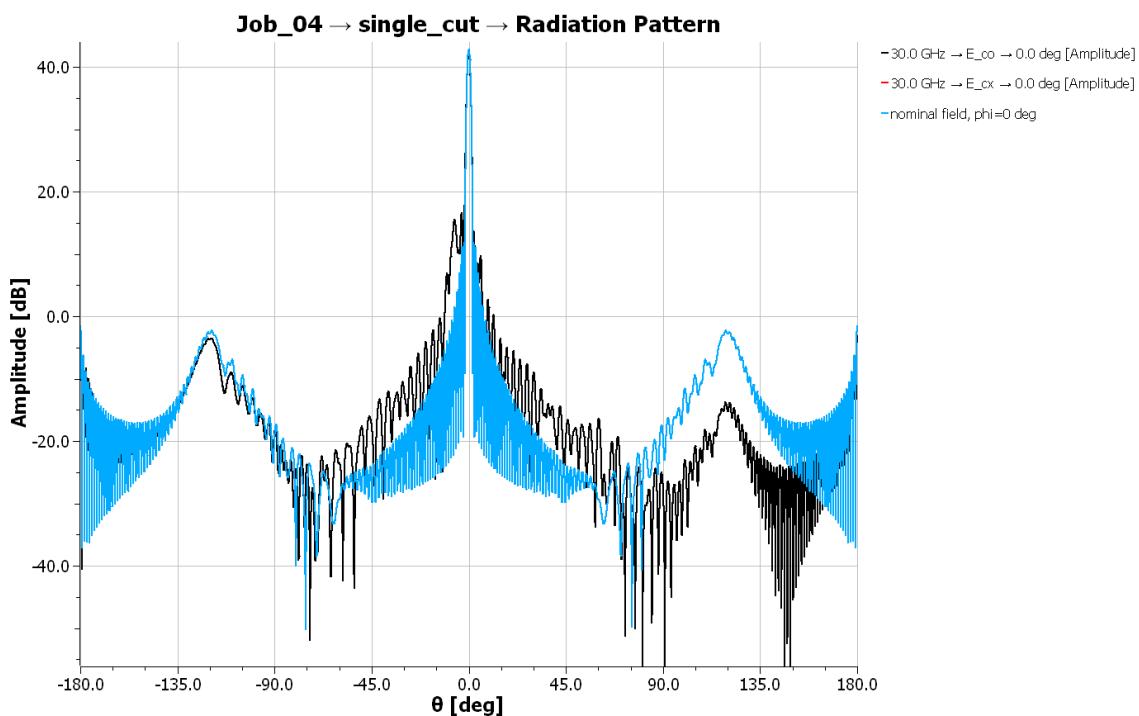


Figure 6-19 Field computed in Job_04 for $\phi=0^\circ$: spherical wave scattering on the three struts and field from the reflector.

We finally add the second part of the commands, in order to include the plane wave scattering effect, i.e. the field from the reflector to the struts. The

commands of Figure 6-20 are therefore run, and the total field is obtained in Figure 6-21. Figure 6-21 shows in green the pattern obtained in Job_04. It is seen that the plane wave scattering on the three struts clearly adds two sidelobes in the $\phi=0^\circ$ plane relative to the results of Job_04. The rest of the pattern is almost unchanged. The two sidelobes are located at $\theta=-48^\circ$ and $\theta=24^\circ$. The sidelobe at $\theta=-48^\circ$ is the same sidelobe that we observed in Figure 6-14 due to the plane wave scattering of Figure 6-12. The two sidelobes can be understood from the following. The rays from the reflector towards the struts are all parallel, if the reflector is only illuminated by the feed. This means that the scattered field from each of the struts will be mainly concentrated on a cone around the strut. The struts are making an angle of 24 degrees with the normal, and this means that the cone opening angle of the strut field will also be 24 degrees. The Figure 6-22 shows the strut field cones for the three struts and the three pattern cuts at $\phi=0^\circ$, 30 and 60 degrees. Inspecting the figure reveals that the pattern cuts intersect the strut cones at some theta angles. At these theta angles the sidelobes will appear. The strut cones intersect the $\phi=0^\circ$ plane at $\theta=24^\circ$ and $\theta=-48^\circ$.

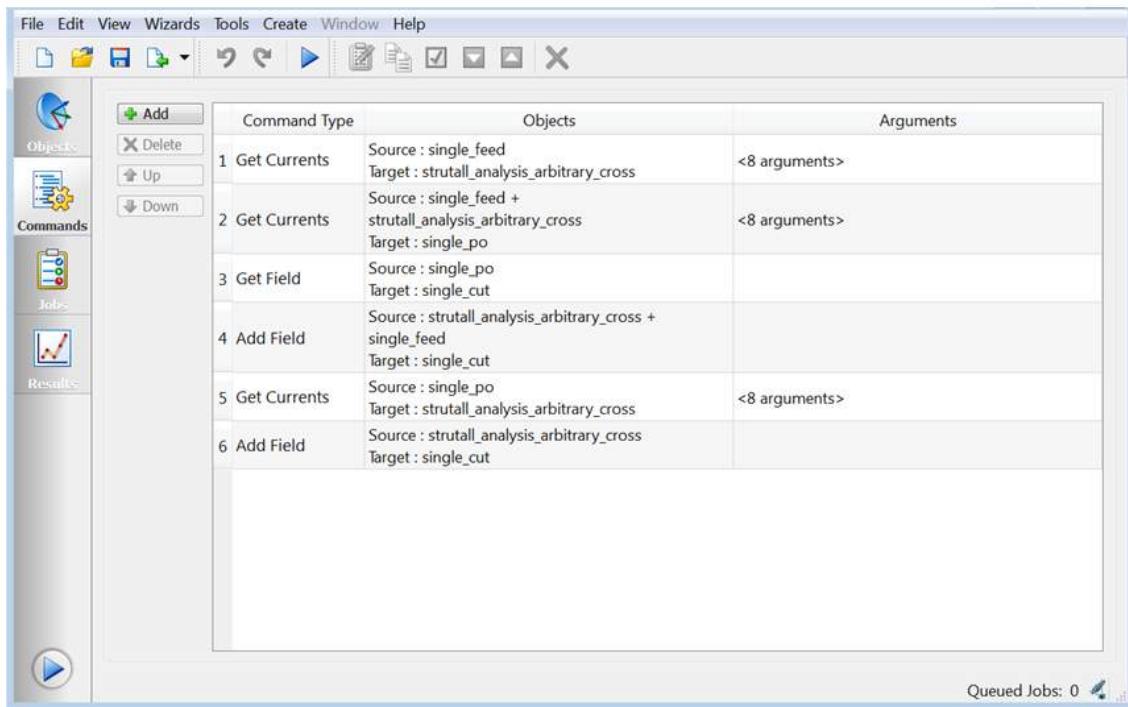


Figure 6-20 Commands for Job_05: plane wave scattering on the three struts from the reflector, total field.

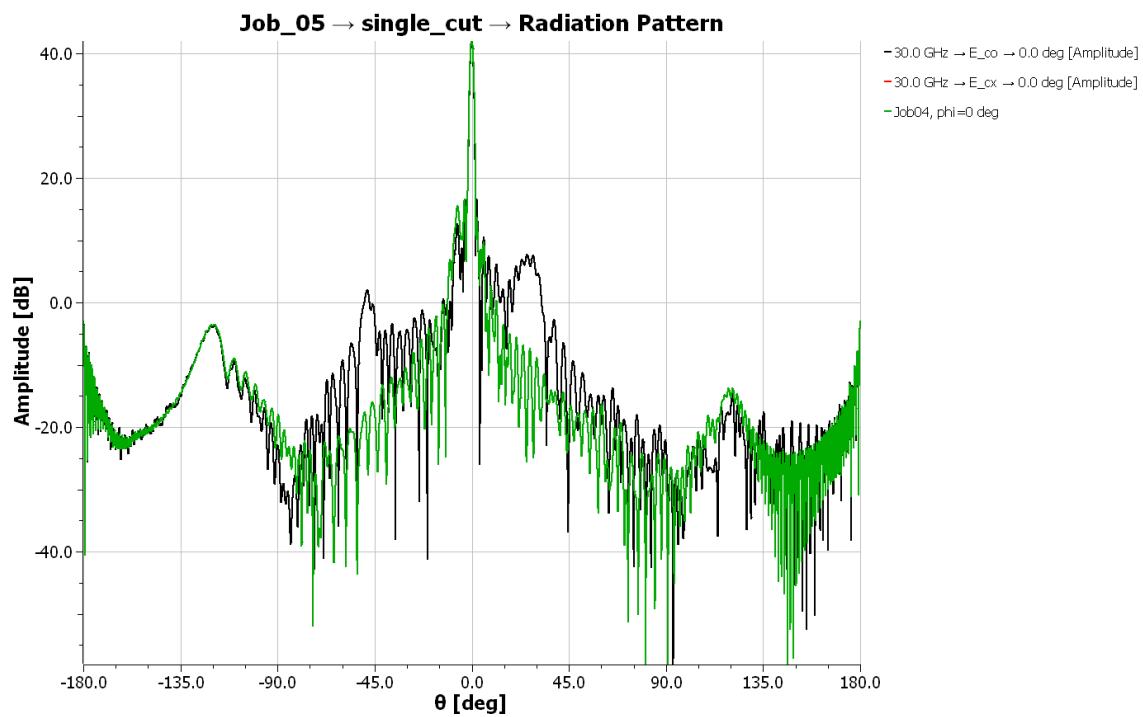


Figure 6-21 Field computed in Job_05 for $\phi=0^\circ$: plane wave scattering on the three struts, total field.

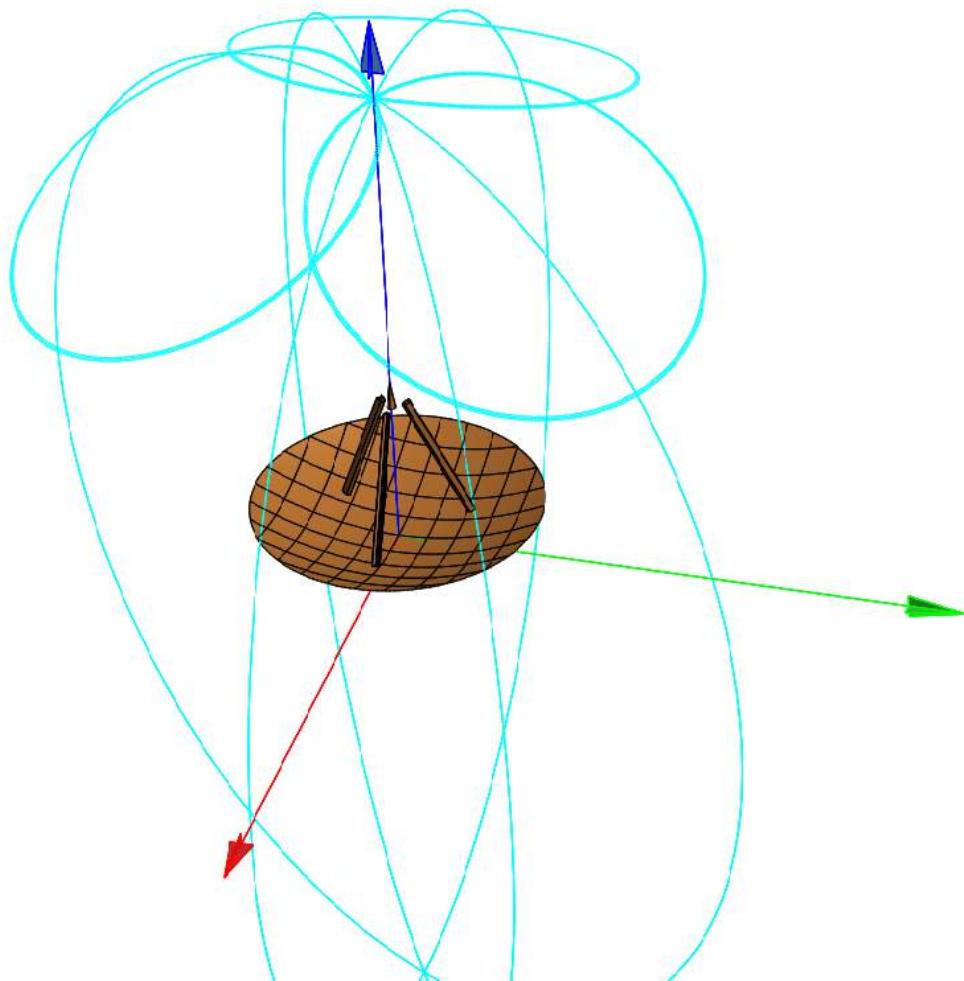


Figure 6-22 Strut diffraction cones and pattern cuts at $\phi=0^\circ$, 30° and 60° .

6.2 Dual Reflector with Blockage

The present example concerns a rotationally symmetric dual reflector antenna system. In such case the blockage due to the subreflector must be accurately modeled, since this can have significant influence on the gain and the sidelobes. Moreover, the spillover from the feed must be included in the computations because it can significantly contribute to the radiation pattern off-axis. A GRASP project can be opened from the box TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The project files are located in the TESTCASES/DUAL_REFL_WITH_BLOCKAGE subdirectory in the installation directory.

With the purpose of making the subreflector blockage and feed spillover effects easily perceived, the dimensions of the antenna are exaggerated relative to a traditional design. The main reflector aperture is only 50 wavelengths, and the feed illumination at the edge of the subreflector is just 12 dB below peak.

A real dual reflector would of course need some sort of support for the subreflector, but this is here neglected. The reader is referred to Section 6.1 for

an example on how to calculate support strut scattering.

6.2.1 Geometry

The antenna operates at 30 GHz, i.e. the wavelength is 0.01 m, and has the following geometrical characteristics:

- main reflector diameter : 0.5 m
- main reflector focal length : 0.25 m
- distance between subreflector foci : 0.15 m
- subreflector eccentricity : 3

The antenna is designed by means of the wizard for dual reflector systems using the above data, as shown in Figure 6-23.

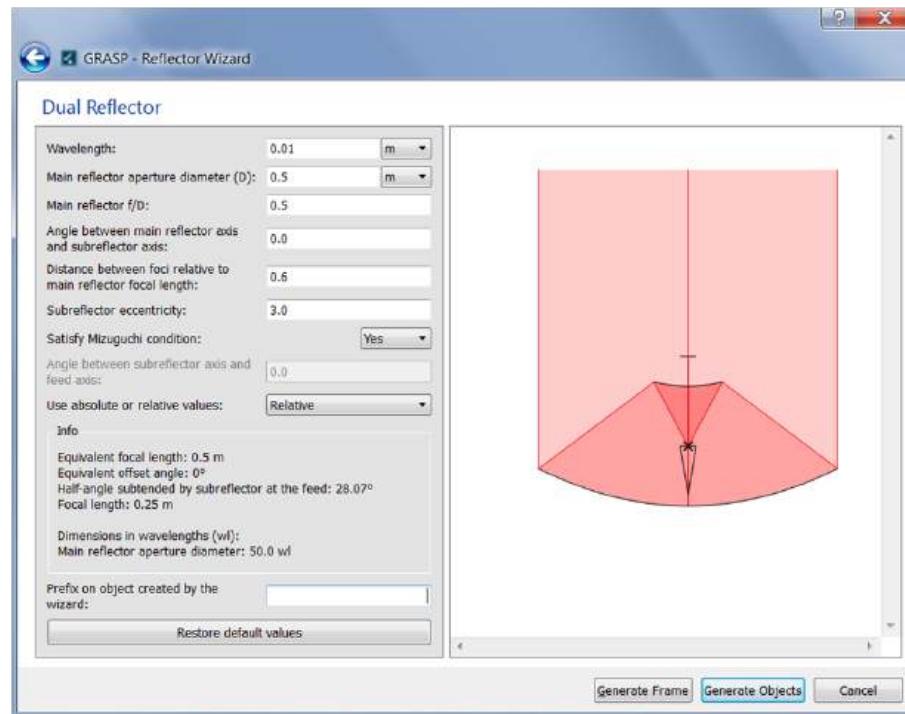


Figure 6-23 Wizard for the dual reflector antenna system.

A drawing of the geometry as generated by GRASP is shown in Figure 6-24.

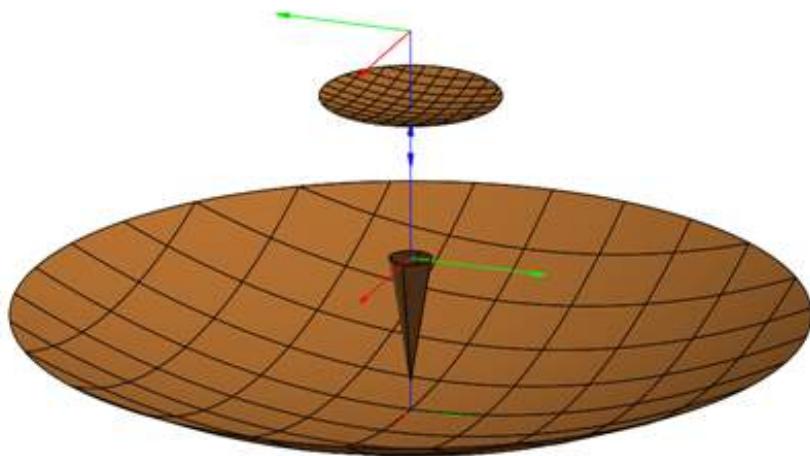


Figure 6-24 Geometry of the dual reflector antenna system.

In the following sections five cases will be considered:

1. Main reflector field
2. Feed/subreflector spillover included
3. Subreflector blockage by null field assumption
4. Subreflector blockage by accurate analysis
5. Main reflector field, subreflector calculated by GO and GTD
6. More PO iterations and comparison to BoR-MoM

6.2.2 Main reflector field

After having created the objects and closed the wizard the only object that needs to be modified is the field storage object `CUT` which assumes a calculation of the main beam and the first 3 sidelobes in 3 polar cuts. The object is opened and the start/end value of θ is changed to $0^\circ/45^\circ$ in one cut. In this cut the field shall be calculated in 451 points. Finally, the object is saved and closed.

In Figure 6-25 the command list generated by the wizard is shown. The list includes four commands:

```
get currents on the subreflector
get currents on the main reflector
get the field from the main reflector
add the field from the feed and subreflector
```

For both `GET CURRENTS` commands the number of PO patches are found by the automatic internal convergence procedure ensuring that the fields converge at the required field points. The `GET FIELD` command calculates the

field from the main reflector currents and the ADD FIELD command includes the contributions from the feed and the subreflector.

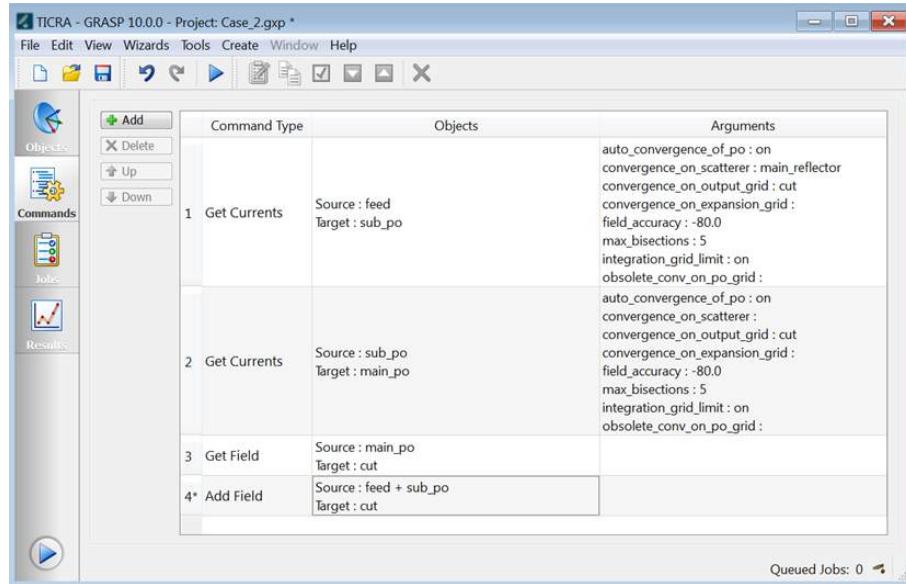


Figure 6-25 Commands generated by the wizard.

In this section we are only interested in the field from the main reflector. Thus the last command is deactivated and the remaining commands are executed under Job_01. The resulting pattern is displayed in the results section as shown in Figure 6-26 where the ordinate scale is set to run from -30 dB to 40 dB. It is seen that the sidelobe behavior is quite irregular up to about 16° from boresight. The reason to this is discussed in more detail in Section 6.2.5.

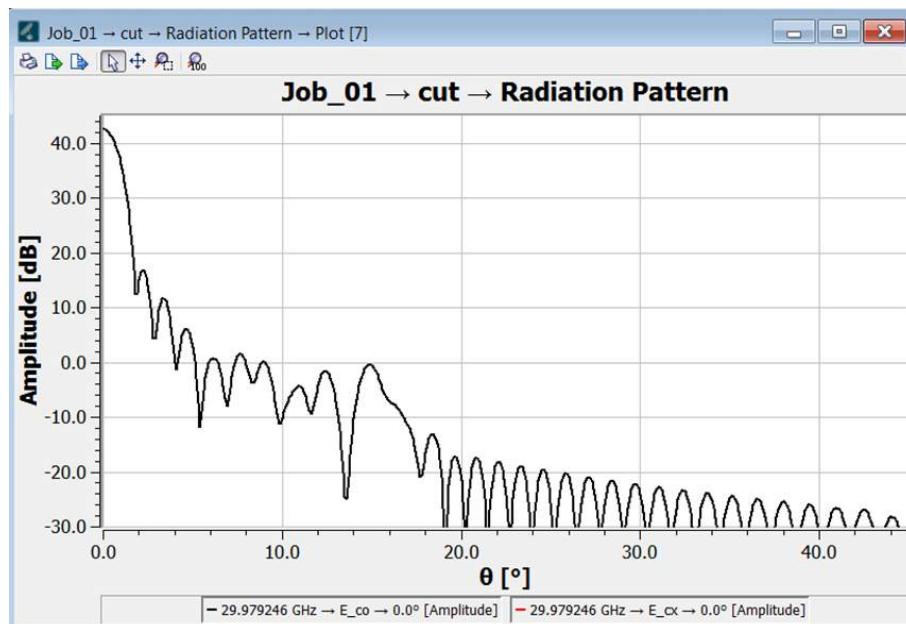


Figure 6-26 Radiation pattern generated by the main reflector currents.

6.2.3 Feed/Subreflector Spillover

A significant contribution to the far field is the direct field from the feed as this is directed towards the far field region as opposed to a single reflector geometry. Around the boresight of the antenna the feed field is shadowed by the subreflector, hence, in order to take this shadowing into account the field generated by the subreflector currents must be included together with the feed field. Both contributions are included in the ADD FIELD command generated by the wizard. This command is now re-activated and the four commands of Figure 6-25 are executed under Job_02. The resulting pattern is shown in Figure 6-27.

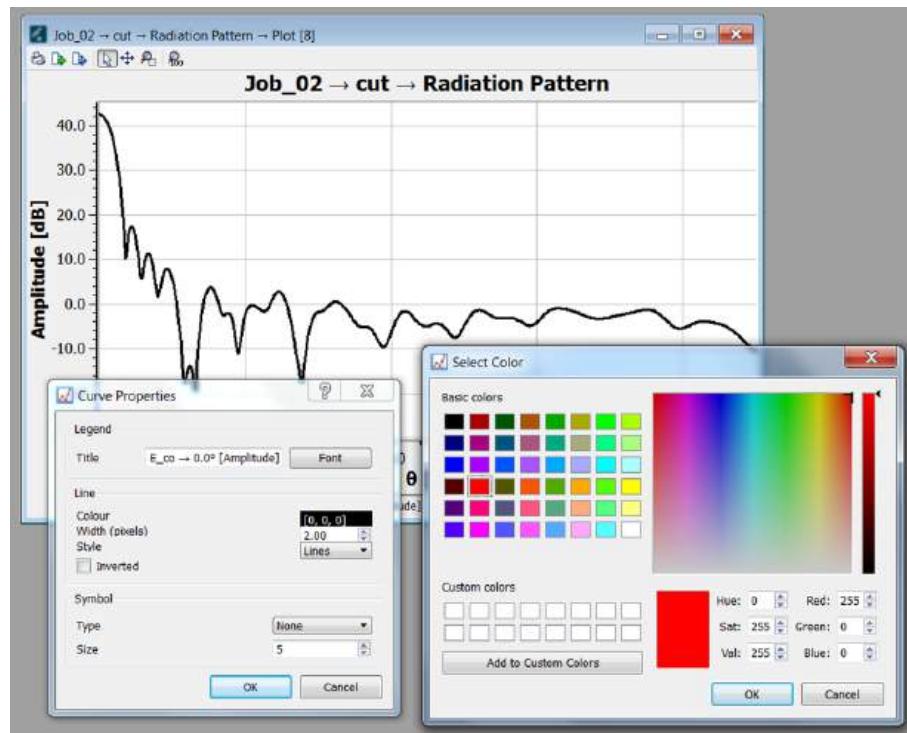


Figure 6-27 Radiation pattern including feed and subreflector contributions and menus for manipulating the curve.

By right-clicking on the curve a number of options appear and selecting PROPERTIES opens a dialog where details for the curve can be defined, such as color, line width, etc. In this case we select the color to be red and close the dialog. After right-clicking on the red curve and selecting COPY the mouse is placed over the canvas for the pattern obtained for Job_01 and after right-clicking the option PASTE is selected. The pattern for Job_02 is copied into the pattern plot for Job_01, as shown in Figure 6-28.

Figure 6-28 shows that the influence from the feed and the subreflector is very small within the main beam and the first few sidelobes, but from about 16° the spillover past the subreflector becomes the dominant contribution.

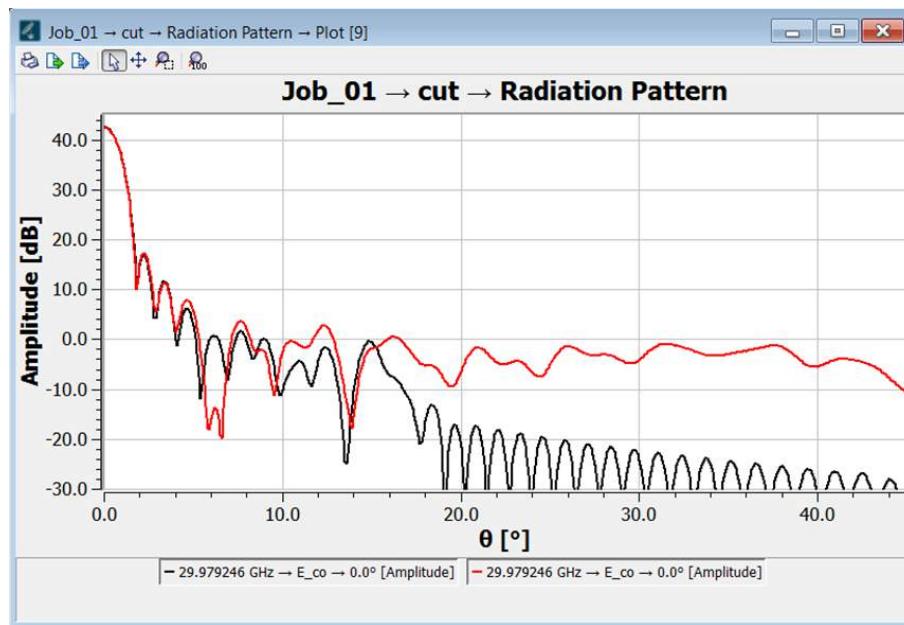


Figure 6-28 Radiation patterns. black curve: main reflector field only, red curve: including feed and subreflector.

6.2.4 Subreflector Blockage

Another contribution that needs to be considered is the subreflector blocking the nominal field from the main reflector. There are two approaches for this analysis:

1. The main reflector area “behind” the subreflector, hence, the area invisible from the boresight direction, is replaced by a hole, i.e. the currents in this area shall not contribute to the field. This is a fast approach but only accurate close to boresight.
2. The currents on the subreflector generated by letting the main reflector field illuminate the subreflector are calculated. The field from these currents are added to the far field. This approach is accurate but time consuming relative to the first approach.

To apply the 1st option the main reflector object must be modified to have a central hole of the same size as the subreflector. The object `MAIN_REFLECTOR` is opened and the attribute `CENTRE_HOLE_RADIUS` is given the value 0.057 m which is the radius of the subreflector. This value can be found from the object `SUB_RIM`. This antenna system is shown in Figure 6-29.

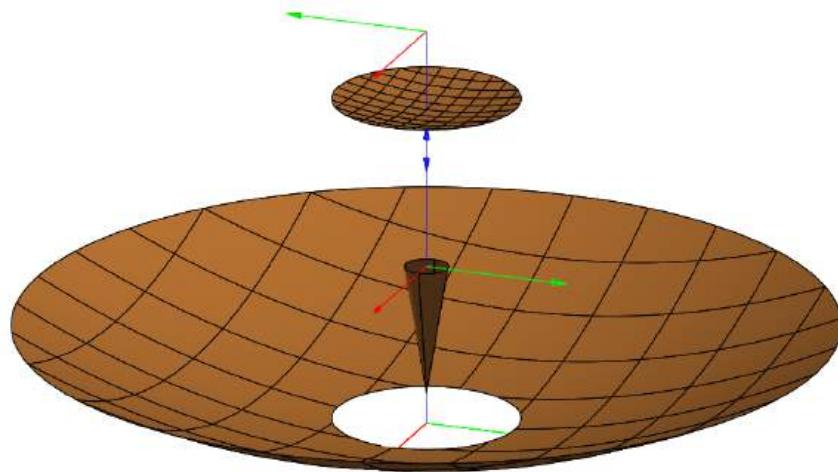


Figure 6-29 Main reflector with hole simulating the blocked area.

The calculation commands are the same as before and are run under Job_03. The calculated pattern is plotted in the usual way, the color is changed to green and the curve is copied to the two previous results as shown in Figure 6-30. It is seen that the subreflector blockage has a strong impact on the main beam and the first sidelobes. The peak directivity is reduced by about 1 dB and the near-in sidelobes are increased by about 10 dB.

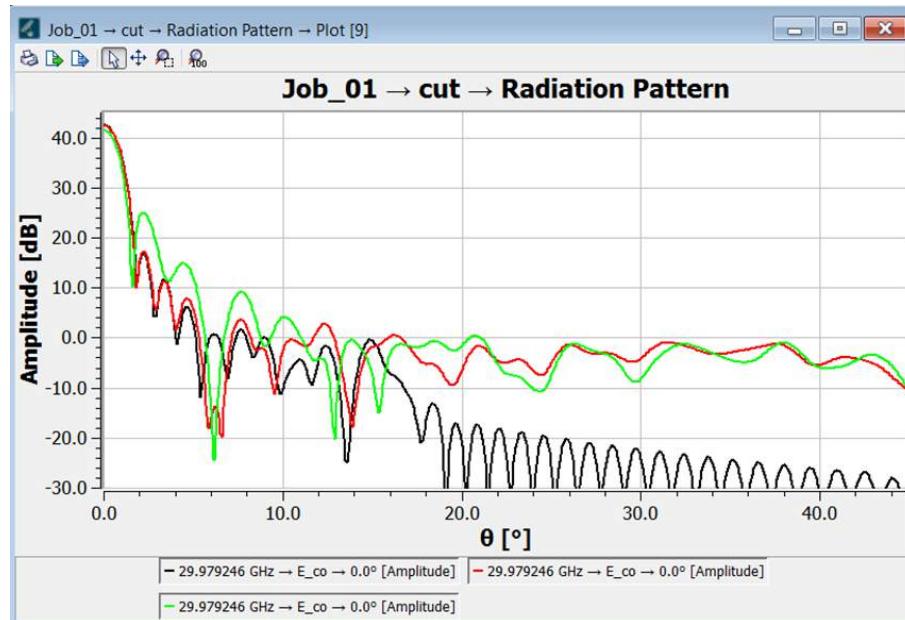


Figure 6-30 Radiation patterns. black curve: main reflector field only, red curve: including feed and subreflector, green curve: including subreflector blockage by hole in main reflector.

For the more rigorous approach the hole in the main reflector is again removed. The four commands, shown in Figure 6-25 and used until now, are

still valid but we need to make a modification to the GET CURRENTS command for the main reflector. Since we now want to calculate the field from the main reflector at the subreflector it is necessary to include this reflector as a convergence criterion for the main reflector PO currents. Two more commands are then added. One GET CURRENTS command for the subreflector with the main reflector currents as the source and one ADD FIELD command where the field from the currents on the subreflector excited by the main reflector are added to the total field. The six commands are shown in Figure 6-31 where the arguments are expanded such that all settings can be seen.

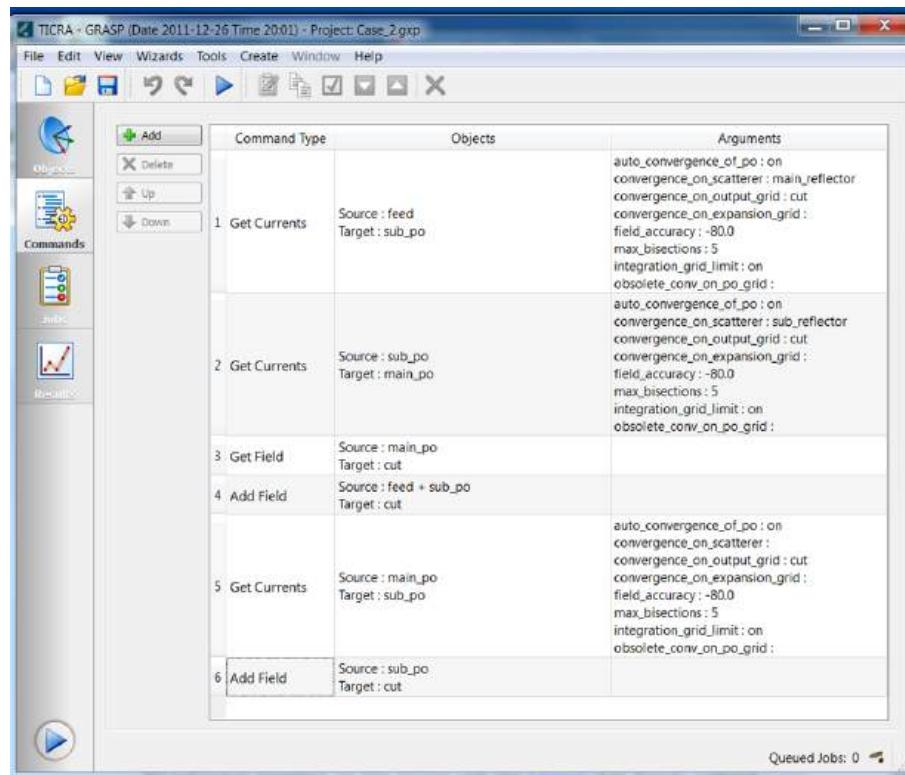


Figure 6-31 Commands including the subreflector blockage of the main reflector field.

The commands are run under Job_04 and the pattern is displayed in the RESULTS section. The color of the curve is changed to blue and copied into the pattern plot with the previous results, as shown in Figure 6-32. It is seen that in the main beam region the subreflector blockage predicted by the hole in the main reflector (green curve) is practically identical to the accurate analysis (blue curve). The two results differ slightly more far out in the sidelobe region, but here the result is dominated by the feed spillover anyway.

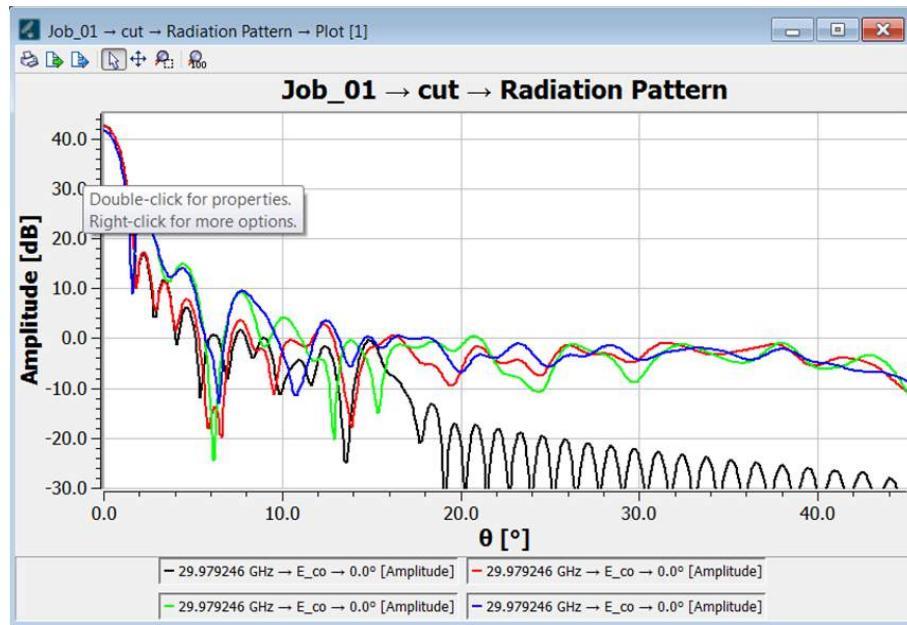


Figure 6-32 Radiation patterns. black curve: main reflector field only, red curve: including feed and subreflector, green curve: including subreflector blockage by hole in main reflector, blue curve: including subreflector blockage by accurate analysis.

6.2.5 Nominal Field, Subreflector Calculated by GO and GTD

It was pointed out in Section 6.2.2 that the sidelobes are very irregular in the angular region up to about 16° from boresight. We will show in this section that these irregularities are coming from diffractions from the subreflector. To this end an object `SINGLE_REFLECTOR_GTD` of the `SINGLE-REFLECTOR GTD` class is created for the subreflector as shown in Figure 6-33. The attribute `GO` is set to `ON` meaning that only reflected rays from the subreflector are included.

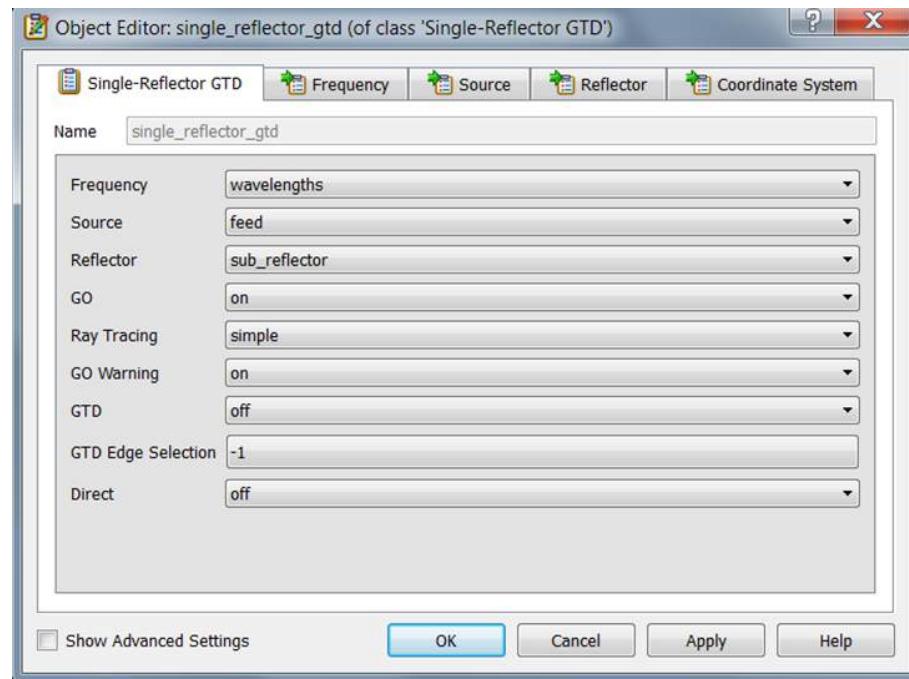


Figure 6-33 Defining Single-Reflector GTD object.

Only two commands are needed to calculate the pattern, namely a GET CURRENTS and a GET FIELD command, as shown in Figure 6-34. The commands are run under Job_05 and the pattern can be seen as the red curve in Figure 6-35 where the black curve is the results from Section 6.2.2 where the subreflector field was calculated by PO. The red curve exhibits the regular sidelobe structure typical for a circular aperture.

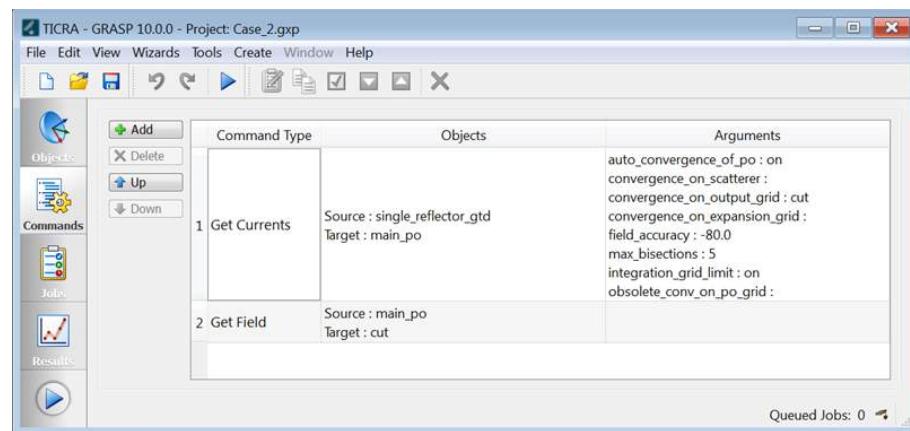


Figure 6-34 Commands for using GO on the subreflector.

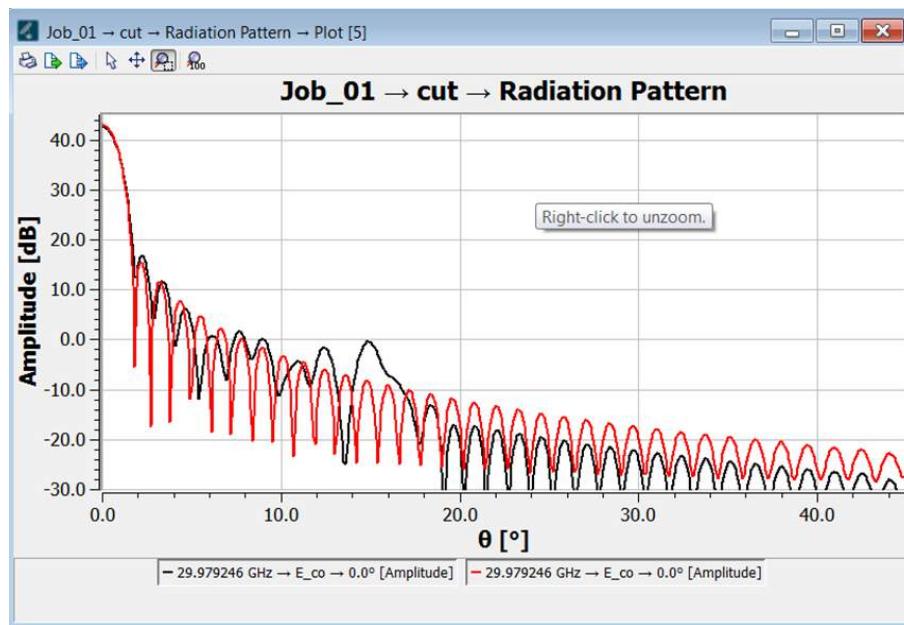


Figure 6-35 Main reflector patterns. Black curve: subreflector field calculated by PO, red curve: subreflector field calculated by GO.

Next, the attribute GTD is set to ON meaning that also diffracted rays from the subreflector edge are included. The same two commands in Figure 6-34 are run under Job_06 and the pattern is shown as the green curve in Figure 6-36. It is seen that the GO+GTD result is almost identical to the black curve using PO for the subreflector. The irregular sidelobe pattern is therefore generated by the diffractions from the subreflector edge.

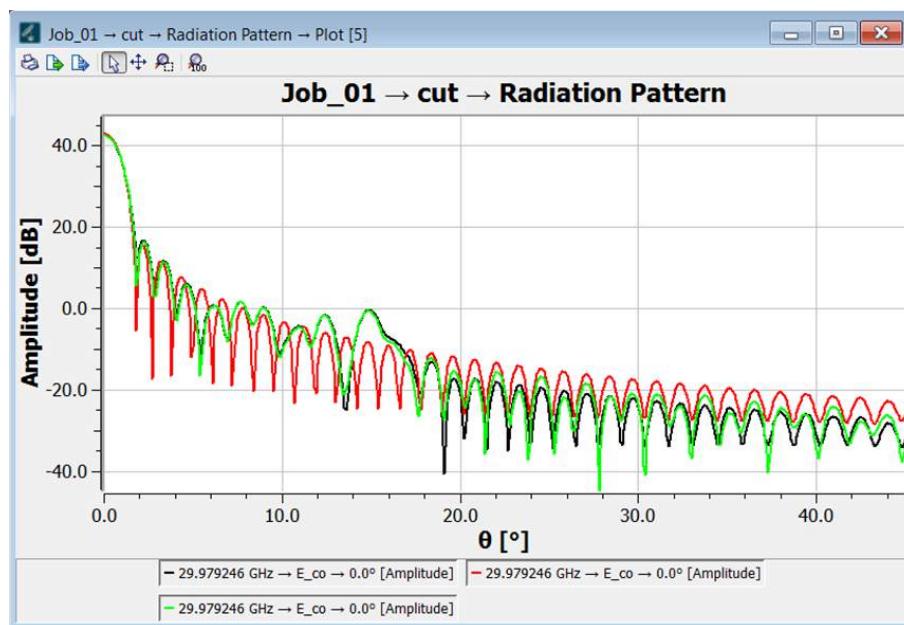


Figure 6-36 Main reflector patterns. Black curve: subreflector field calculated by PO, red curve: subreflector field calculated by GO, green curve: subreflector field calculated by GO+GTD.

6.2.6 More PO iterations and comparison to BoR-MoM

It was demonstrated in Section 6.2.4 that the subreflector blockage can be calculated from the induced PO-currents on the subreflector generated by the field from the main reflector. It was also demonstrated that this field represents a significant contribution to the total pattern. The induced currents on the subreflector will of course illuminate the main reflector once more and generate a second set of induced currents. These currents will also radiate into the far field and they will again be shadowed by the subreflector. We can therefore calculate a second iteration by the following four commands:

- get the currents on the main reflector from the main reflector induced currents on the subreflector
- add the field from the main reflector to the far field
- get the currents on the subreflector from the main reflector
- add the field from the subreflector to the far field

The effect is illustrated in Figure 6-37 where the red curve shows the result when two iterations are included. It turns out that for the present configuration five iterations are necessary before a stable result has been obtained, shown by the green curve in the figure.

The antenna system investigated in this section is rotational symmetric and it is therefore also possible to analyse it by means of the Method of Moments solution for rotational symmetric bodies, BoR-MoM. This result is also shown in Figure 6-37 and it is seen that the PO result with 5 iterations and the BoR-MoM solution are almost identical.

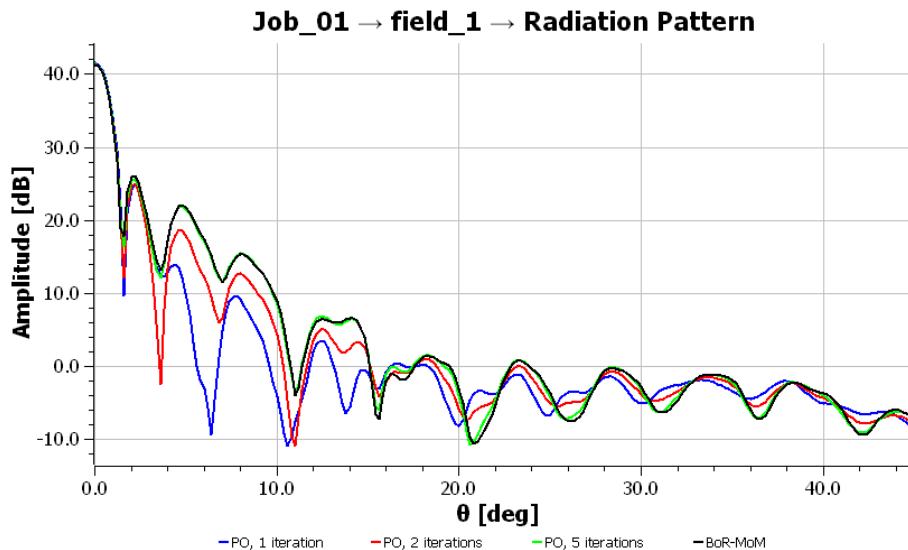


Figure 6-37 Main reflector patterns calculated with PO and BoR-MoM. Blue curve: PO, 1 iteration, red curve: PO, 2 iterations, green curve: PO, 5 iterations, black curve: Bor-MoM solution.

6.3 Single Shaped Reflector with Circular Polarisation

The following example will show the analysis of a single offset reflector antenna, shaped to produce a highly contoured beam on a certain coverage with sidelobe constraints on another coverage. Since the antenna operates in circular polarisation it is expected to produce a high degree of polarisation purity, despite the offset geometry. We will use this example to demonstrate the significance of including the PTD currents in the analysis, by looking carefully at the predicted cross polar performance of the antenna. A GRASP project can be opened from the box TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The project files are located in the TEST-CASES/SINGLE_SHAPED_REFL_WITH_CP subdirectory in the installation directory.

In general, the shaping of a reflector antenna is normally based on PO, without including the PTD contributions. The reason is that PTD tends to make the solution unstable, therefore it is recommended to use only PO for the shaping optimisation, and include PTD only afterwards, to check that the requirements are still met.

6.3.1 Shaped Reflector Geometry

The shaped reflector surface is synthesized by the POS program starting with a paraboloid with the following data:

reflector diameter : 3.3 m
 reflector focal length : 3.3 m
 Offset : 1.85 m

The user starts by opening the Single reflector wizard and providing the data above together (choosing to apply absolute values) with the wavelength of 0.075 m. The prefix of the objects is set to nil (blank). Upon finishing the wizard the objects will appear in the OBJECT EXPLORER as shown in Figure 6-38.

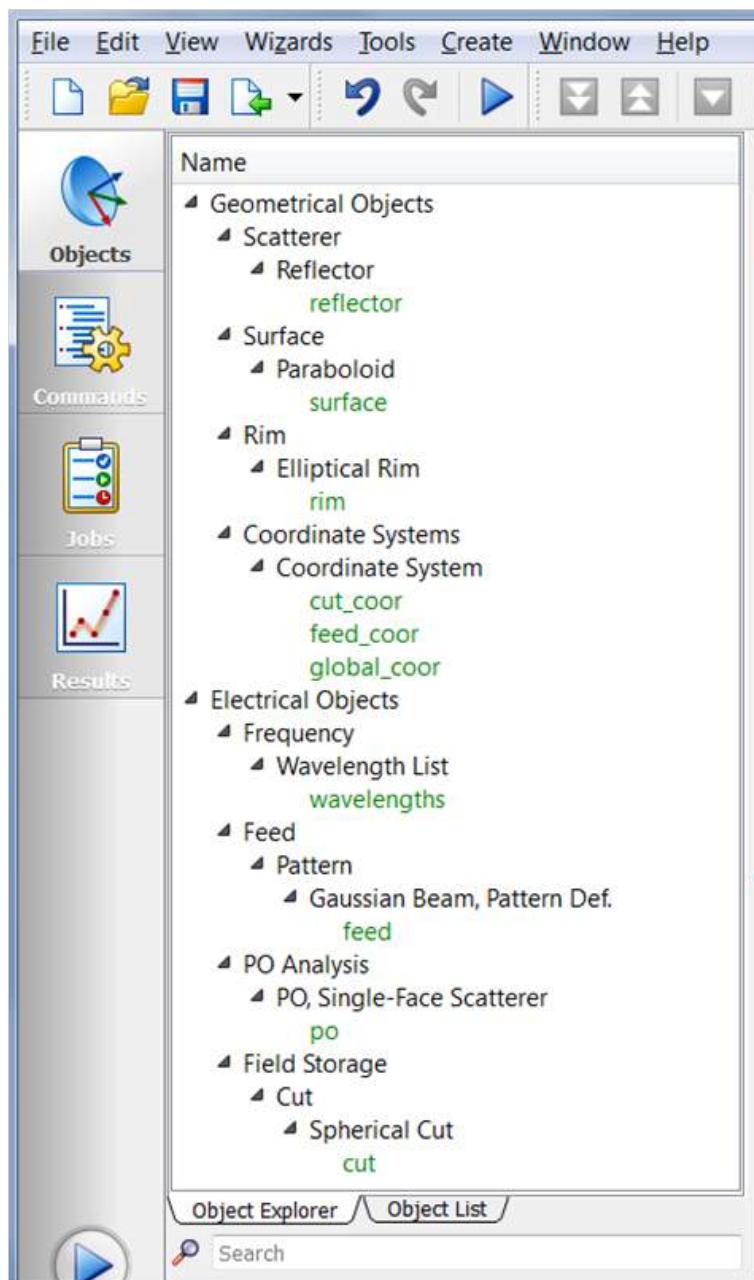


Figure 6-38 Objects generated by the wizard.

The feed used in this project is slightly different from the one generated by the wizard and the feed data must be modified. This is done by opening the object FEED. The value of TAPER ANGLE is changed from the value calculated in the wizard to 25.6° . Furthermore, circular polarisation is introduced by changing the POLARISATION attribute to LHC. Then, the object is saved.

The reflector surface data for the shaped antenna are contained in the file GR3.SFC, where values are given in meter. The file was created by the reflector shaping program POS and is provided to GRASP by creating an object called REGULAR_XY_GRID of REGULAR_XY-GRID class. In this object the name GR3.SFC is given to the FILE NAME attribute. The other attributes are left unchanged. Then the reflector object REFLECTOR is opened and the attribute SURFACE is changed from SURFACE to REGULAR_XY_GRID.

The reflector geometry is shown in Figure 6-39. The rays included in the plot clearly show that the reflector is no longer focusing.

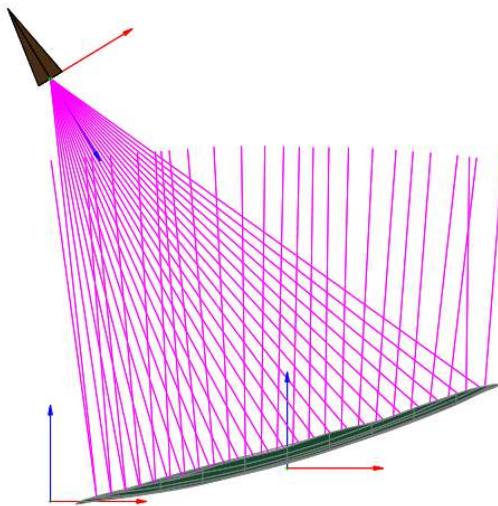


Figure 6-39 Shaped reflector generating highly contoured beam.

6.3.2 Analysis without PTD

In this section we wish to calculate the field with PO, but without PTD. However, the PTD contribution is included by default in GRASP. It is therefore necessary to open the PO object and change the attribute METHOD from PO_PLUS_PTD to PO.

As we would like to see the result in a contour plot, an object of SPHERICAL GRID class shall be created. It is shown by a listing in Figure 6-40. The uv grid window limits are -0.2 to 0.3 along u and ± 0.2 along v.

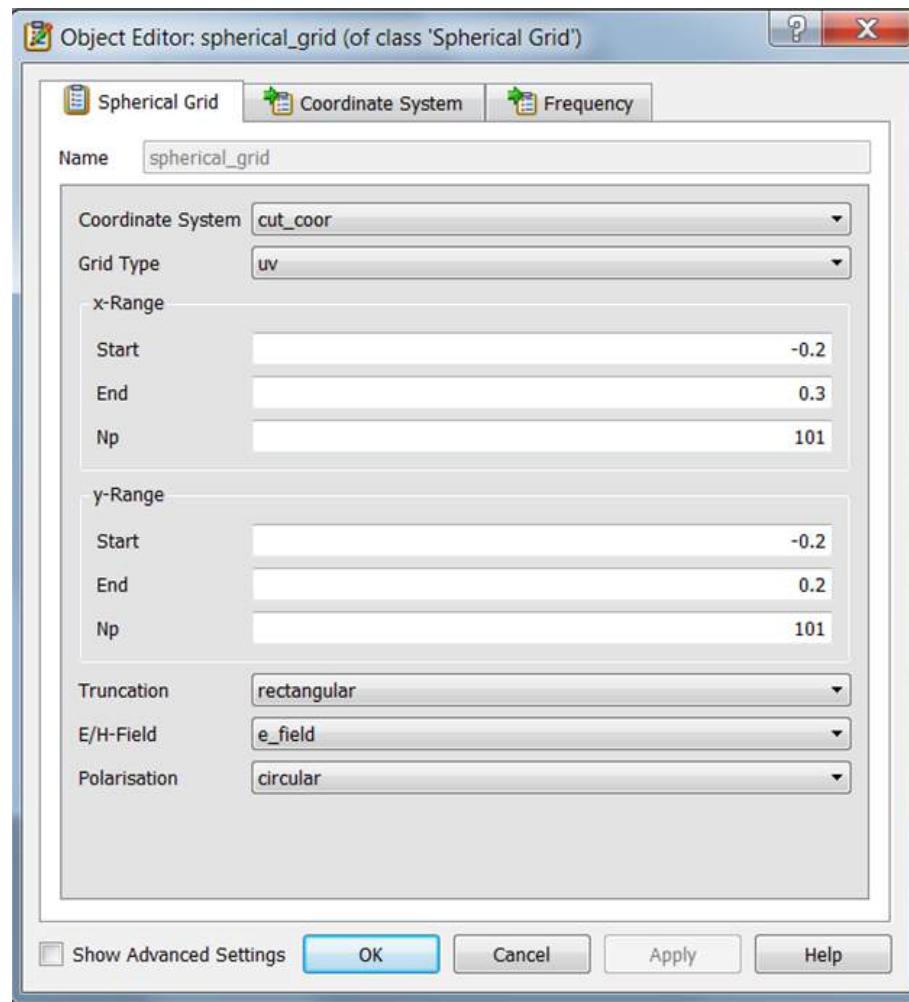


Figure 6-40 Content of spherical grid object.

The antenna can now be analysed using the two commands generated by the wizard. However, the GET CURRENTS command must be modified to assure convergence on the just defined SPHERICAL GRID. The commands with expanded arguments are shown in Figure 6-41. The commands are then executed under Job_01.

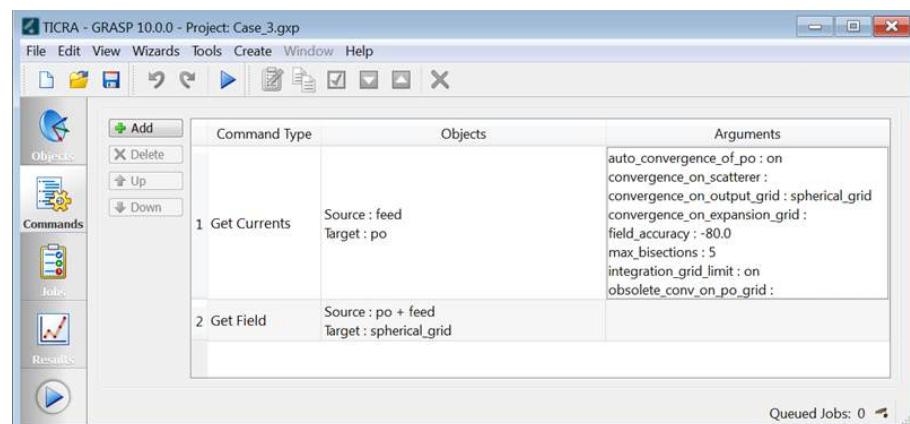


Figure 6-41 Command sequence for calculation of contour plot.

The calculated result can be displayed in the RESULTS window, as illustrated in Figure 6-42 for the co-polar component.

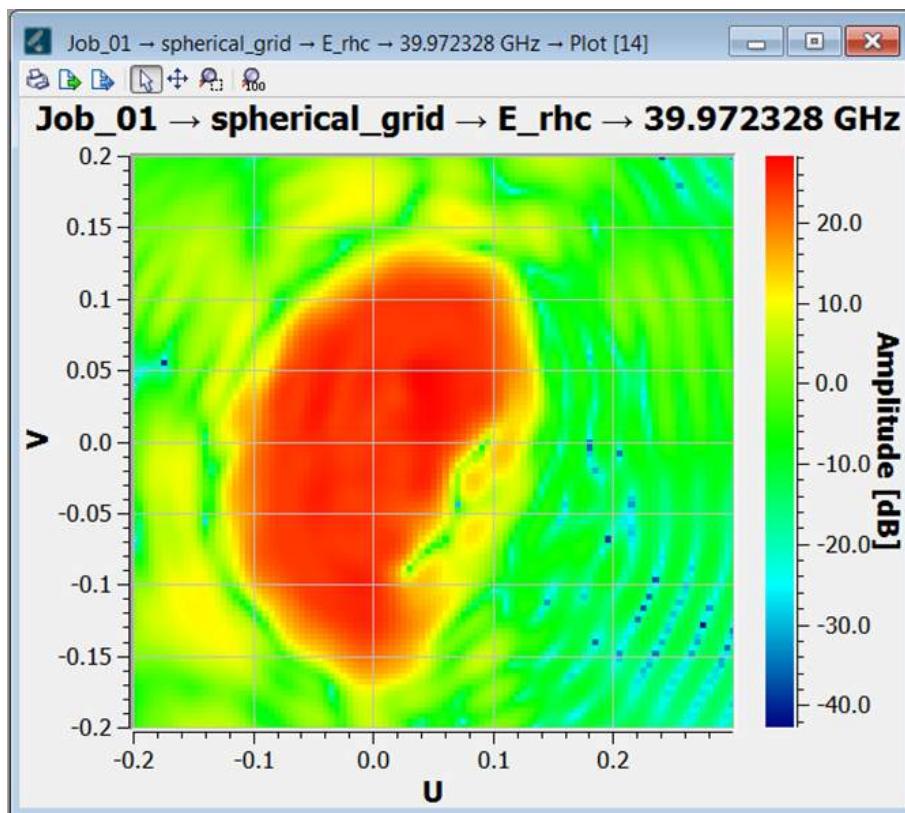


Figure 6-42 Co-polar component of the field computed in Job_01.

For shaped beams optimised for particular coverage areas the facilities available in the POSTPROCESSOR delivered with GRASP are more efficient. A project file, CASE3_JOB_01_CO.PPC, is already prepared and available in the subdirectory TESTCASES/SINGLE_SHAPED_REFL_WITH_CP/JOB_01.

In Figure 6-43 (generated by the POSTPROCESSOR using as project file CASE3_JOB_01_CO.PPC, after reading the project file press F2, F3 and F4 to generate the plot) the result in terms of the copolar field is shown superimposed on the polygons used by POS to carry out the shaping optimisation. Levels are plotted at 24.0 dBi which is the minimum coverage level, 2 dB below and at -6.0 dBi which is the maximum level in the polygon where the level is suppressed in the optimisation (the sidelobe constraint). Bearing in mind that the left polygon indicates the coverage where side lobe suppression must be guaranteed while the right polygon indicates the coverage where the shaped beam must be radiated, we can conclude that the antenna satisfies all our requirements.

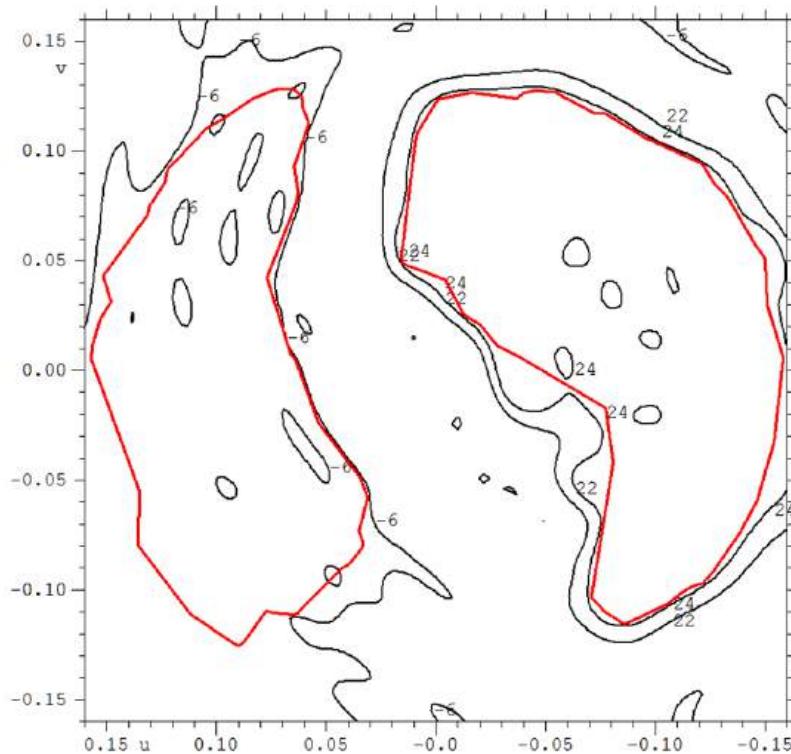


Figure 6-43 Co-polar radiation pattern from contoured beam antenna, with the East and West coverage polygons superimposed in red.

In Figure 6-44 (generated by the postprocessor using as project file CASE3_JOB_01_XPD.PPC, after reading the project file press F2, F3, click PROCESS ACTIONS->NEW COMPONENTS, press OK and finally press F4 to generate the plot) a plot of the XPD is shown with three contours.

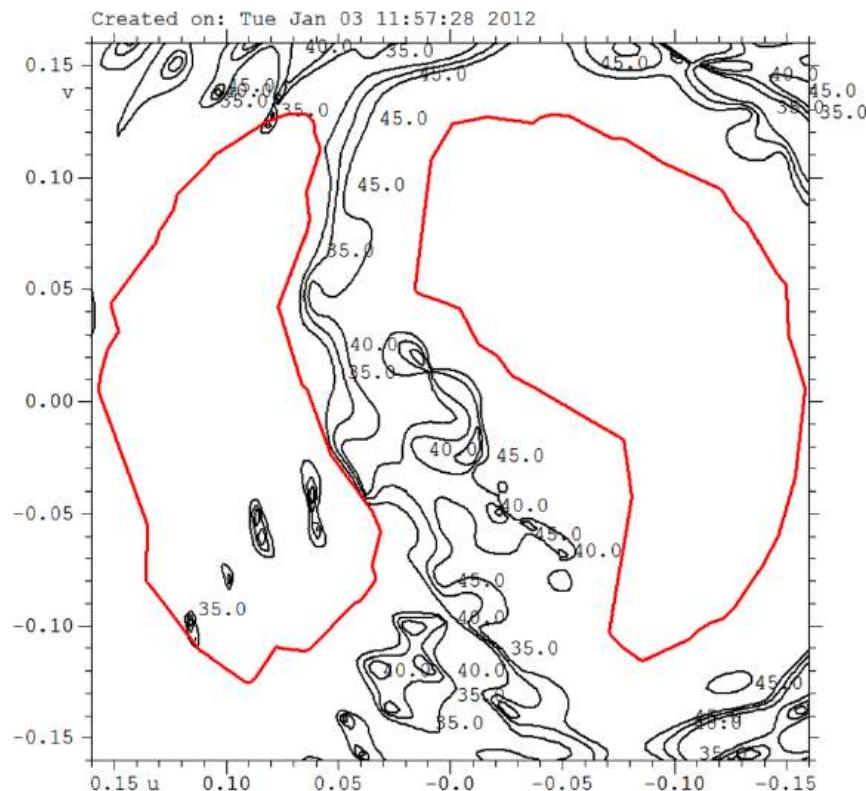


Figure 6-44 XPD contours at 35, 40 and 45 dB levels, when PTD is not included in the analysis.

6.3.3 Analysis with PTD

We wish now to include the PTD contribution. This is done by opening the PO object and change the attribute METHOD from PO to PO_PLUS_PTD.

The two commands are still valid and they are run under Job_02.

The obtained XPD can be seen in Figure 6-45 (generated by the postprocessor using as project file CASE3_JOB_02_XPD.PPC, after reading the project file press F2, F3, click PROCESS ACTIONS->NEW COMPONENTS, press OK and finally press F4 to generate the plot). The difference with Figure 6-44 is evident.

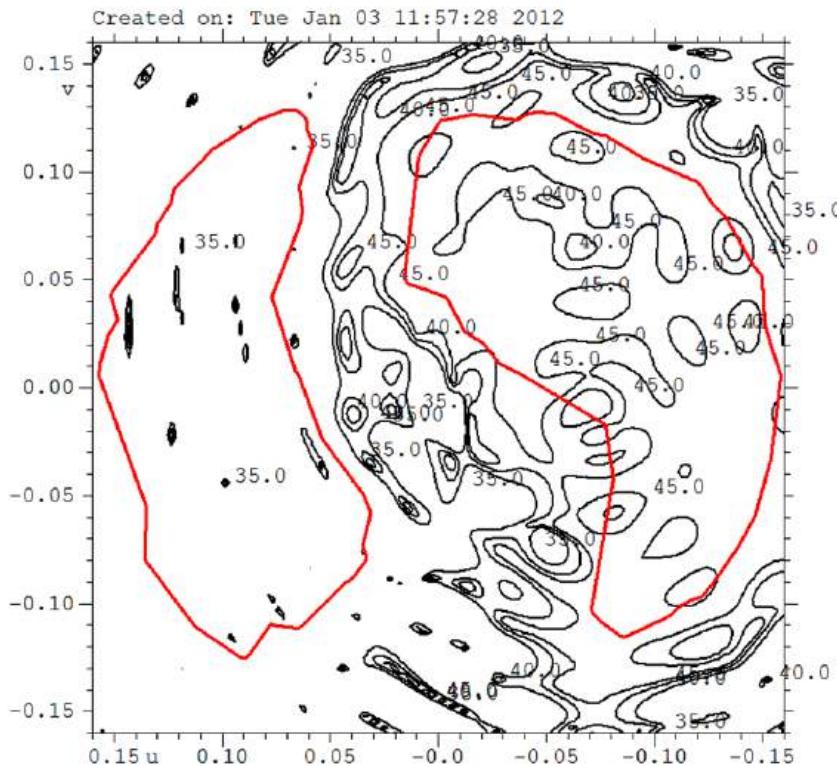


Figure 6-45 XPD contours at 35, 40 and 45 dB levels, when PTD is included in the analysis.

6.4 Dual Gridded Reflector

The present example deals with a dual gridded reflector system. A gridded reflector is a commonly used solution for providing low cross polarisation in case of linear polarisation; the reflector surface is manufactured from a polarisation sensitive material consisting of a conducting grid. Systems of this type are commonly referred to as dual gridded reflectors. The term "dual" is due to the fact that there are most often two reflectors involved, but not in the usual configuration as subreflector and main reflector. Rather, both reflectors are "main reflectors", one for each orientation of linear polarisation.

A GRASP project can be opened from the box TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The project files are located in the TEST-CASES/DUAL_GRIDDED_REFL subdirectory in the installation directory.

In the following we will first describe how to set up the geometry, and then the commands necessary to run an analysis.

6.4.1 Geometry

We consider the dual gridded reflector shown in Figure 6-46.

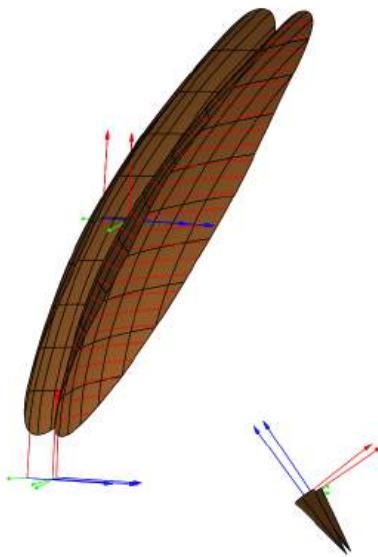


Figure 6-46 Dual gridded reflector geometry. The rear reflector is solid and the front reflector is a grid mounted on a dielectric layer with lines perpendicular to the offset plane.

The antenna consists of two shells, one behind the other, in such a way that they share the same circular output aperture when seen from the boresight direction. The front reflector is made from a grid of conducting strips aligned with the desired polarisation orientation of the far field and mounted on a dielectric layer. There is one feed for each reflector, in order to achieve a linear dual polarized field.

We look at a design made for a 12 GHz coverage of Australia.

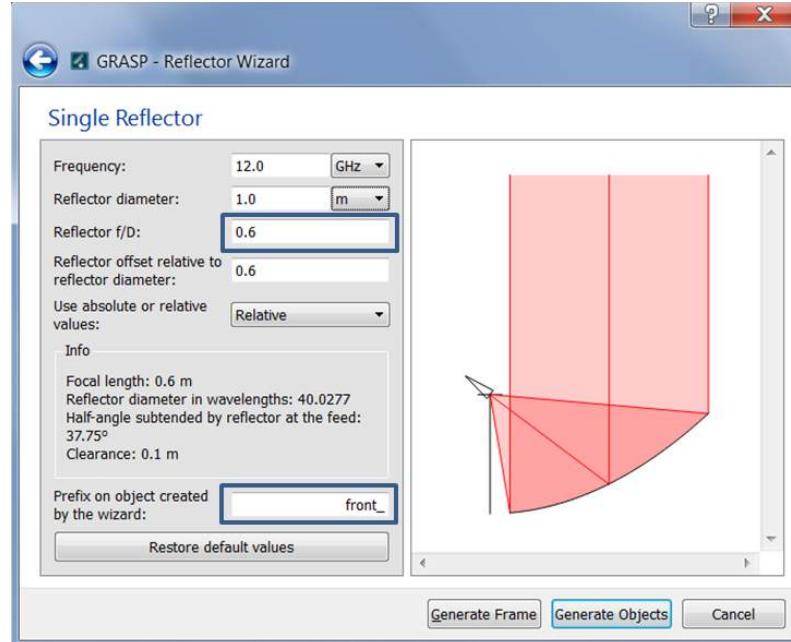
The diameter of both shells is 1.0 m and the focal lengths (before shaping) are 0.65 m for the rear and 0.6 m for the front reflector. They are rotated 3.5° making it possible to place them behind each other and still have the feeds nearly in the same plane, a desirable feature when integrating the antenna on a spacecraft. The front antenna is made from conducting strips with a width of 0.25 mm and a periodicity of 1.25 mm mounted on a dielectric layer with a thickness of 0.4 mm and a dielectric constant of 2.5. These parameters are such as to ensure good reflection properties for the co-polarised field at 12 GHz whilst transmitting the orthogonal field with limited reflection loss. The surface material properties are described in the STRIP GRID object of the electrical properties class.

6.4.2 Procedure for setting up dual reflector geometry

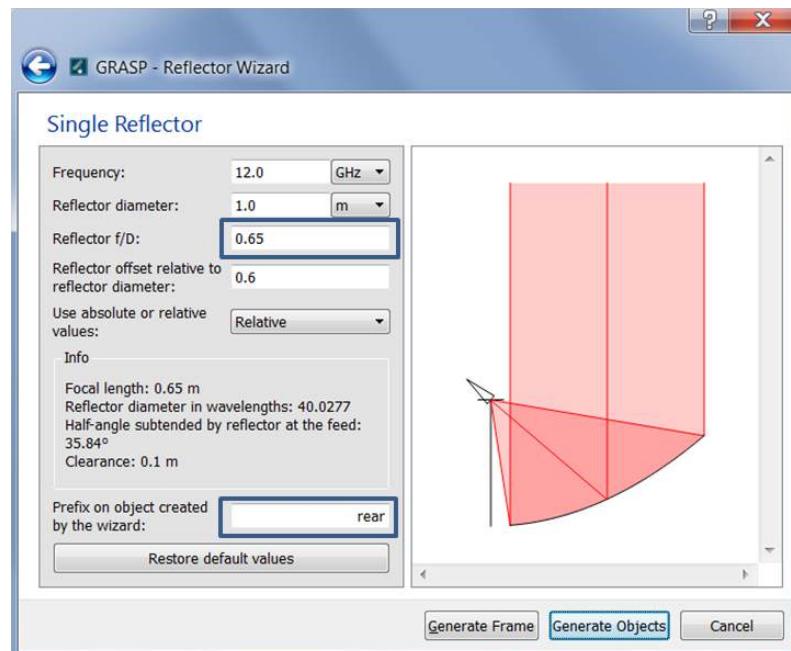
The procedure for accommodating two main reflectors in the way described above is demonstrated step by step in the following:

After starting GRASP the single reflector wizard is opened twice and the data for the rear reflector and the front reflector are specified as shown in Figure

6-47.



(a) Front reflector



(b) Rear reflector

Figure 6-47 Single reflector wizard with data for initial reflector geometries.

The only difference between the two data sets are the f/D and the prefix (blue frames). The result of the wizard definitions is visualised in Figure 6-48.

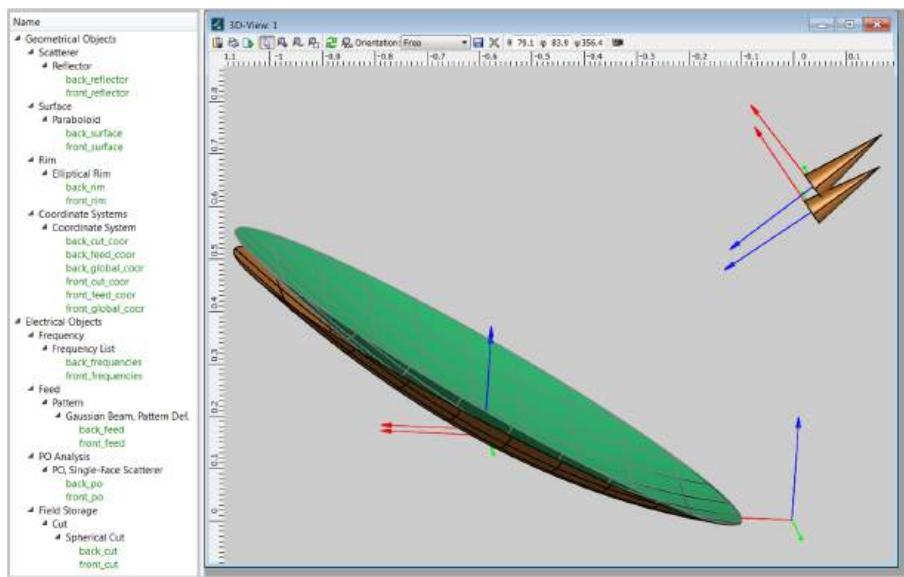


Figure 6-48 The two single reflector geometries as created by the wizard.

In Figure 6-48 there is no clearance between the surfaces and the feeds. The task is now, by means of coordinate system manipulation to establish the necessary clearance between the front and the rear single reflector. The steps are the following:

FRONT_CUT_COOR renamed to FRONT_ROT_COOR_SYS

REAR_CUT_COOR renamed to REAR_ROT_COOR_SYS

The CUT coordinate systems are located at the centre of the reflector apertures with the z-axes in the boresight direction. This is the axis around which we want to rotate the front and rear reflectors. This is the reason for the name change.

FRONT_GLOBAL_COOR duplicated to GLOBAL_COOR_SYS

Two GLOBAL coordinate systems are created by the wizard, one for the front and one for the rear reflector. It is reasonable to define one common GLOBAL coordinate system for the total geometry so one of the GLOBAL coordinate systems is duplicated as shown above.

FRONT_GLOBAL_COOR renamed to FRONT_COOR_SYS

REAR_GLOBAL_COOR renamed to REAR_COOR_SYS

The wizard generated GLOBAL coordinate systems should not be considered as GLOBAL but as "main" coordinate systems for each part of the reflector combination.

FRONT_ROT_COOR_SYS base redefined from FRONT_COOR_SYS to GLOBAL_COOR_SYS

REAR_ROT_COOR_SYS base redefined from REAR_COOR_SYS to GLOBAL_COOR_SYS

We want to keep the ROT coordinate systems in the same position but the reference should be the GLOBAL_COOR_SYS. Figure 6-49 shows how this is done.

This base change ensures that the ROT coordinate systems are directly linked to the global reference of the whole geometry.

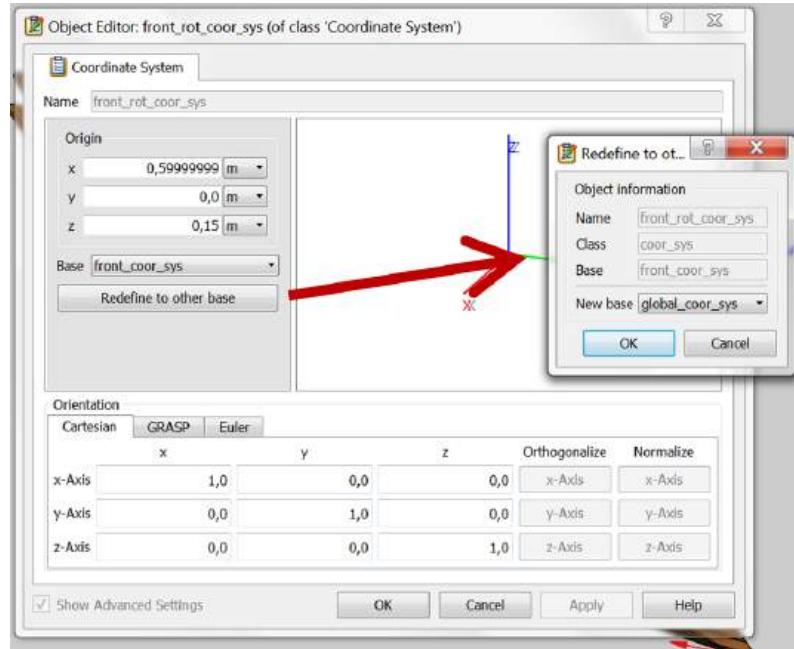


Figure 6-49 The coordinate system dialog showing how the base is changed.

FRONT_COOR_SYS base changed from NONE to FRONT_ROT_COOR_SYS

REAR_COOR_SYS base changed from NONE to REAR_ROT_COOR_SYS

This change of base ensures that when the ROT coordinate systems are moved the reflectors and the feeds are moved with them (the FRONT_REFLECTOR has reference to the FRONT_COOR_SYS which has FRONT_ROT_COOR_SYS as base and FRONT_FEED has reference to FRONT_FEED_COOR which has FRONT_COOR_SYS as base).

We are now ready to apply the rotation of $\pm 3.5^\circ$ to the front and rear reflectors and this is done by applying a ψ rotation to the associated coordinate systems FRONT_ROT_COOR_SYS and REAR_ROT_COOR_SYS. For FRONT_ROT_COOR_SYS the change is shown in Figure 6-50. For REAR_ROT_COOR_SYS the ψ value is -3.5° .

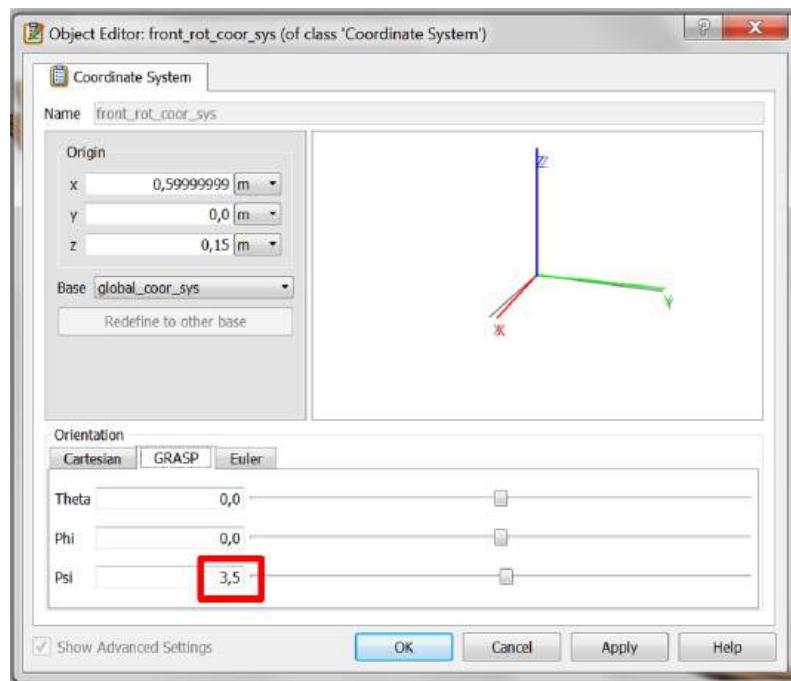


Figure 6-50 The coordinate system dialog showing how the rotation around the z -axis is specified.

In order to ensure clearance between the reflector surfaces after the rotation the rear reflector shall be translated backwards. This translation was accounted for in the wizard procedure by giving the rear reflector a focal length 5 cm larger than the focal length of the front reflector. We will then move the rear reflector backwards by 5 cm and this is done, as shown in Figure 6-51, by subtracting 5 cm from the z-coordinate of REAR_ROT_COOR_SYS, hence changing 0,13846154 m to 0,08846154 m.

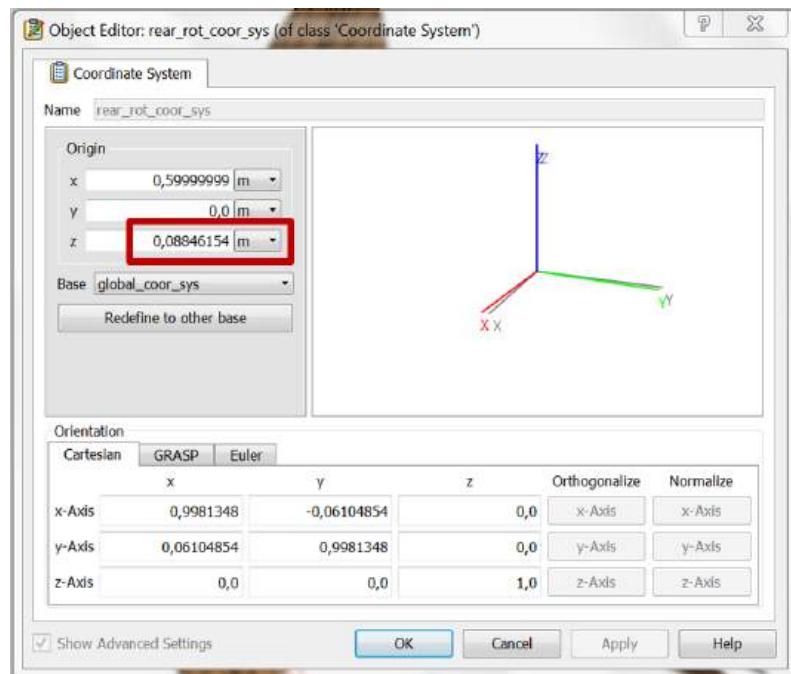


Figure 6-51 The coordinate system dialog showing how the translation along the z -axis is specified.

We have now arrived at a geometry in which clearance between the reflectors and the feeds is ensured. The geometry is shown in Figure 6-52.

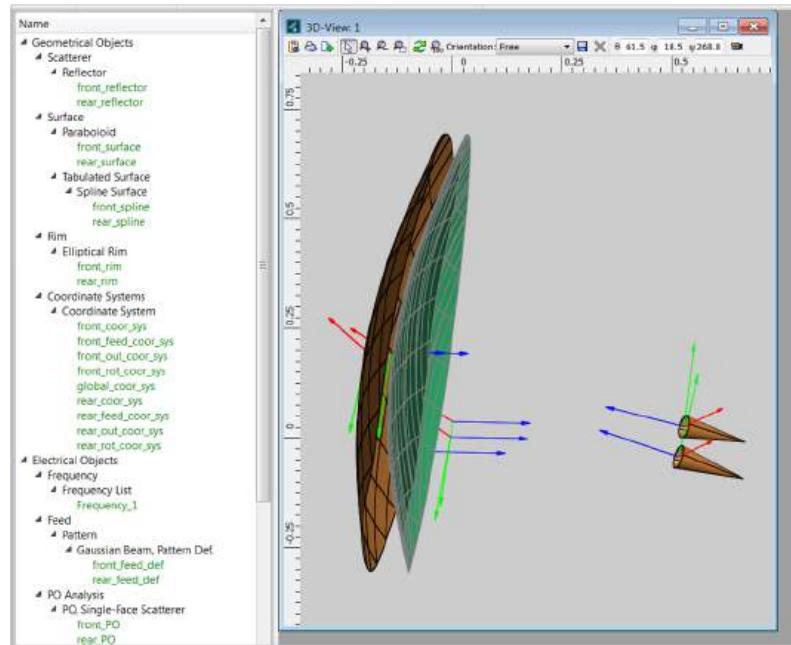


Figure 6-52 The two single reflector geometries after adjustment of coordinate systems.

As we have rotated the front and rear reflector and thereby also the linearly polarised feeds by $\pm 3.5^\circ$ around the antenna z -axis also the polarisation orientation has been rotated with the same amount. We want to recover the

polarisation parallel and orthogonal to the (initial) symmetry plane and this must be done by rotating the feeds around their axes by applying a ψ rotation in the associated coordinate systems. In FRONT_FEED_COOR the ψ angle is changed from 180° to -176.5° and in REAR_FEED_COOR the ψ angle is changed from 180° to $+176.5^\circ$.

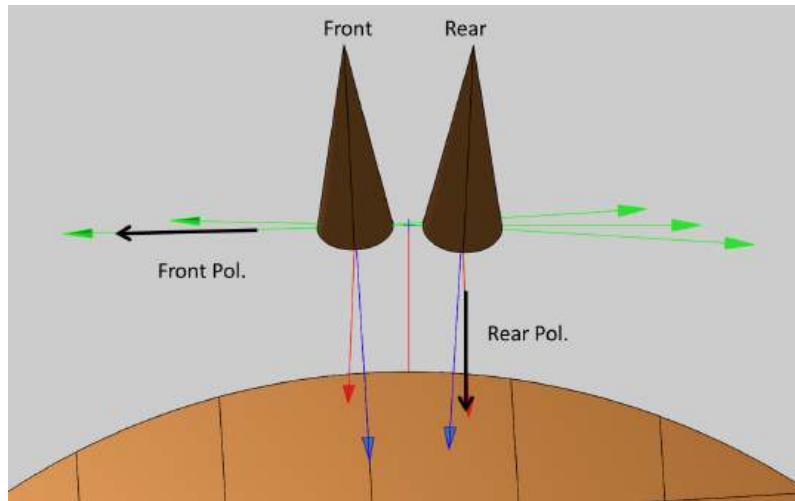


Figure 6-53 Front view of feeds after adjustment of feed coordinate systems.

Figure 6-53 shows a front view of part of the reflector surface and the feeds. The feed rotation of $\pm 3.5^\circ$ appears as the angle between the x -axes (red) and the z -axes (blue). The thick arrow indicates the polarisation direction of the rear and front feed.

This concludes the procedure for setting up the geometry of the reflector surfaces in a dual gridded antenna. To arrive at the data in the TESTCASES/DUAL_GRIDDED_REFL subdirectory a number of additional modifications are necessary:

- On the front shell ELECTRICAL_PROPERTIES objects of STRIP_GRID and DIELECTRIC_LAYER class shall be added.
- The shaping of the surfaces are added by specifying an object of SPLINE_SURFACE class in the DISTORTIONS attribute in the Reflector objects.
- FIELD_STORAGE objects shall be created having reference to GLOBAL_COOR_SYS.

6.4.3 Analysis of the Geometry

The general working principle of the antenna is best described by assuming that the antenna transmits. One of the feeds, the front feed, is oriented such that its linearly polarised field is aligned with the grid on the front reflector, to the greatest possible extent. Thus the feed co-polar component will be reflected by the front shell, whereas the cross-polar component will be transmitted and will hit the rear shell. This is normally made from normal,

conducting surface material, and will reflect the feed cross-polar component. However, due to the asymmetric arrangement, the cross-polar component will be reflected in a direction which can be significantly different from the bore-sight direction of the front reflector, and will thus appear at locations in the far field where it has no impact on the performance of the antenna. Finally, the field reflected by the rear shell is filtered by the grid on the front shell.

The rear feed works in a similar way, except that the primary component of this feed is transmitted through the front shell and reflected by the rear.

We choose to demonstrate the steps of the gridded reflector analysis by looking at how the field from the rear feed is scattered by the front and back reflectors. The basic steps are exactly the same for the front feed, but whereas the significant contribution to the scattered field comes from the rear reflector in the example, it would come from the front reflector if the front feed was considered.

The detailed analysis a dual gridded reflector is slightly more complicated than what we have described above and can easily be understood by looking at Figure 6-54.

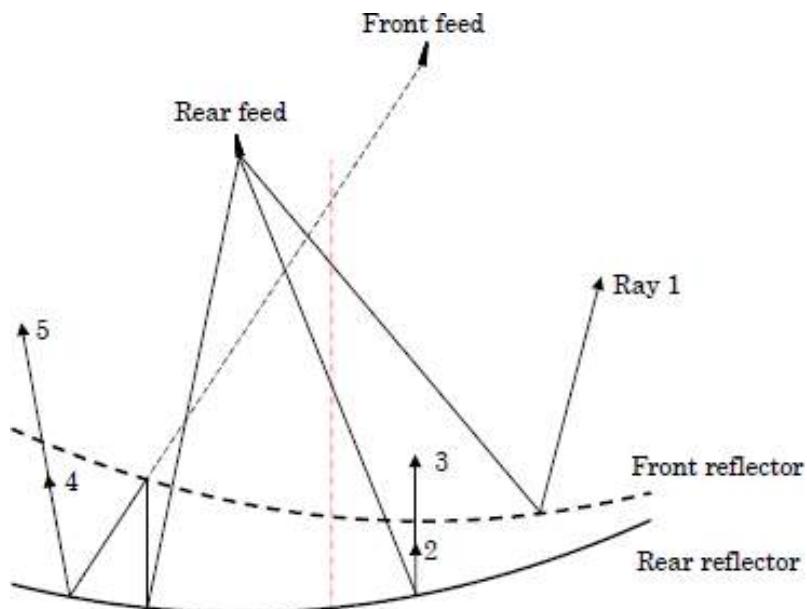


Figure 6-54 The rays from the rear feed in a dual gridded reflector antenna.

The rear feed illuminates the front reflector and a small part is reflected, as indicated by ray 1. This represents most of the cross-polar component of the antenna. The major part of the field from the rear feed is transmitted through the front reflector and hits the rear reflector where it is totally reflected, ray 2. This field passes again through the front reflector, ray 3. The fields associated with the rays 2 and 3 form the normal main beam from the antenna. The reflected ray from the rear reflector, represented by ray 2, is parallel to the antenna axis. It will be reflected again in the rear surface of the front reflector and will appear as coming from the front feed. The field will be reflected again in the rear reflector, ray 4, and will pass through the front reflector, ray 5.

Note that rays 4 and 5 appear on the other side of the main beam compared to ray 3. The field from rays 4 and 5 is smaller than the field from ray 3, since the field has to pass twice through the front reflector, and is responsible of an additional side lobe outside the expected coverage region. The level of this additional side lobe is normally 20 dB lower than the main beam.

The detailed analysis is thus done in five steps:

1. Calculation of the field coming from the rear feed and scattered from the front (gridded) reflector, field_1.
2. Calculation of the field coming from the rear feed, transmitted through the front reflector and scattered by the rear reflector, field_2.
3. Calculation of the field computed in step 2 and transmitted through the front reflector, field_3.
4. Calculation of the field computed in step 2, reflected by the front reflector and later by the rear reflector, field_4.
5. Calculation of the field computed in step 4 and transmitted through the front reflector, field_5.

To get a prediction of the total behaviour of the dual gridded reflector with the rear feed active, the five contributions above must be added. In the following sections it will be described how each contribution is calculated and how the total field is obtained.

6.4.3.1 Field from the rear feed scattered by the front reflector, field_1

The first contribution to the far field is the field from the rear feed reflected in the front reflector. This is an undesired contribution. Since the feed polarisation is perpendicular to the front reflector grid direction, the copolar component will pass the front reflector and the cross-polar component will be reflected.

The antenna shall illuminate Australia, and thus a contour plot of the field is necessary. For this purpose, a new object of SPHERICAL GRID class is defined, as shown in Figure 6-55:

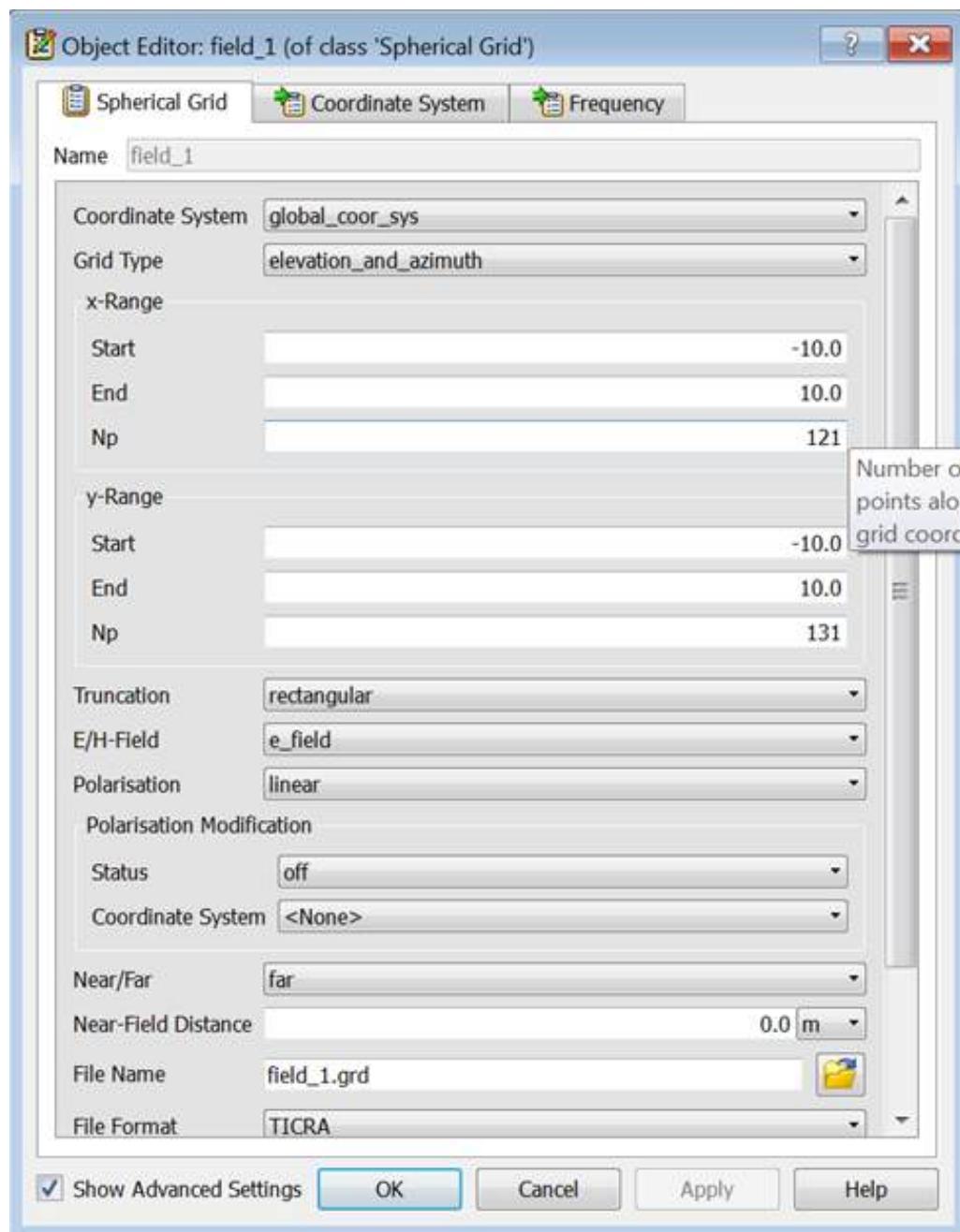


Figure 6-55 Inputs for the spherical grid.

Similar objects are generated for field_2, field_3, field_4, field_5 and field_total.

To compute field_1 the commands shown in Figure 6-56 must be defined.

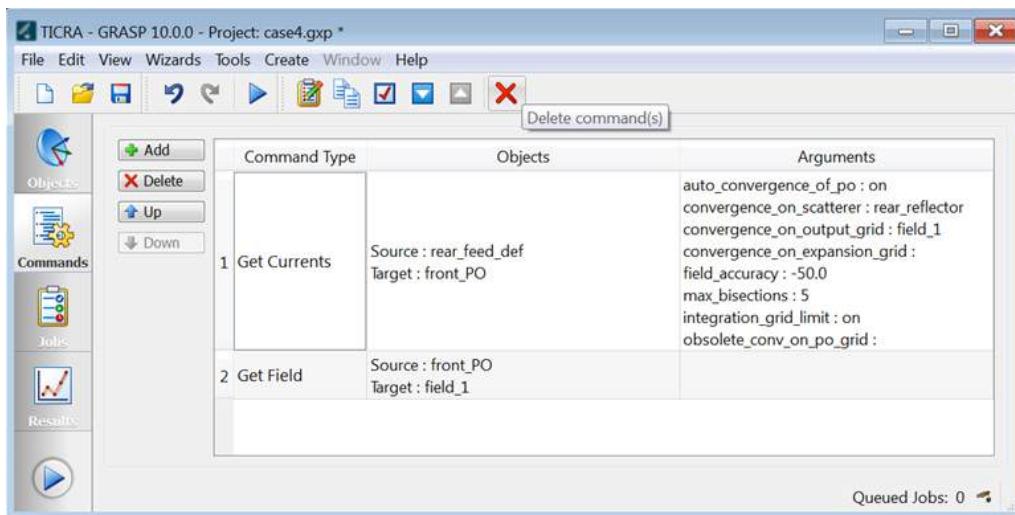


Figure 6-56 Commands to compute field_1.

The PO currents on the front reflector are calculated when illuminated by REAR_FEED_DEF. The number of PO patches is found by the automatic internal convergence procedure using both the far field points in field_1 as well as test points on the rear reflector as convergence criterion (hence, the front reflector currents will be applicable both for calculating the field in a far field grid but also for calculating the induced currents on the rear reflector in the following step). Since the field from the rear feed scattered in the front reflector will be small, only a modest accuracy is required, -50 dB.

The above commands are run under Job_01. Launching these two commands will create a file, FIELD_1.GRD, in which the field data are stored. The field can be displayed in the Results window or in more detail by the Postprocessor delivered with GRASP. Project files are already prepared and can be found in the Job_01 folder, with the names FIELD1_CO.PPC and FIELD1_CR.PPC.

These project files can be read from the Postprocessor after selecting "Contour" presentation. Then simply press F2, F3 and F4 to obtain the field contours.

There are similar project files in the Job_02, Job_03 and Job_06 folders.

The results for both the co- and the cross-polar components are shown in Figure 6-57, together with the coverage polygon (Australia). The contour levels on these plots (as well as all the following contour plots) are 1, 3, 5, 7, and 9 dB below the peak. The figures show that the field levels are low and, since the rear feed is offset relative to the front reflector, the beams are tilted away from the coverage region.

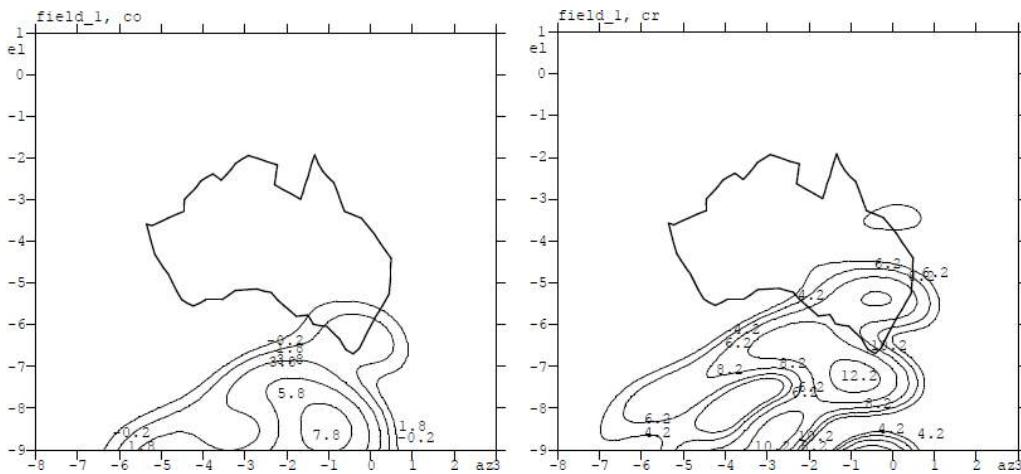


Figure 6-57 Co-and cross-polar components of the field from the rear feed scattered by the front reflector. Contour levels are 1, 3, 5, 7, and 9 dB below the peak.

6.4.3.2 Field from the rear feed passing through the front reflector and scattered by the rear reflector, field_2

The rear reflector is illuminated by two sources, the rear feed and the currents induced by the rear feed on the front gridded reflector. To compute field_2 the commands shown in Figure 6-58 must be defined.

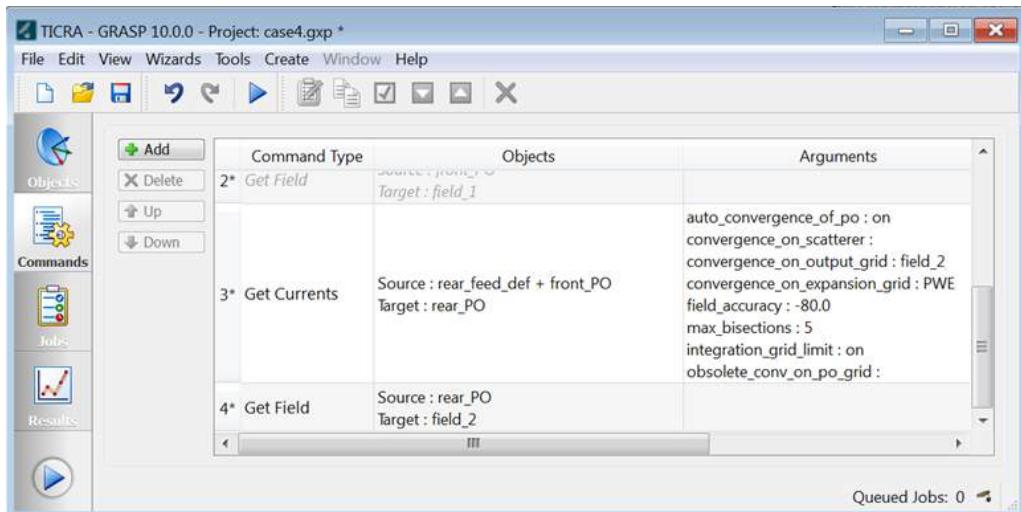


Figure 6-58 Commands to compute field_2.

The PO currents on the rear reflector are calculated when illuminated by REAR_FEED_DEF and the currents on the front reflector calculated in the previous step. The number of PO patches are found by the automatic internal convergence procedure using the far field grid as convergence criterium (hence, the rear reflector currents will be applicable for calculating the far field, field_2, in the following command). The currents on the rear reflector represent the major contribution from the antenna and therefore the accuracy is set to -80 dB. For reasons to be described later the field from the rear reflector will be expanded in plane wave modes and, consequently, the

currents are also required to converge on the plane wave expansion object, PWE, see definition below.

The above commands are run under Job_02. Launching these two commands will create a file, FIELD_2.GRD, in which the field data are stored. The result is shown in Figure 6-59, together with the coverage polygon (Australia).

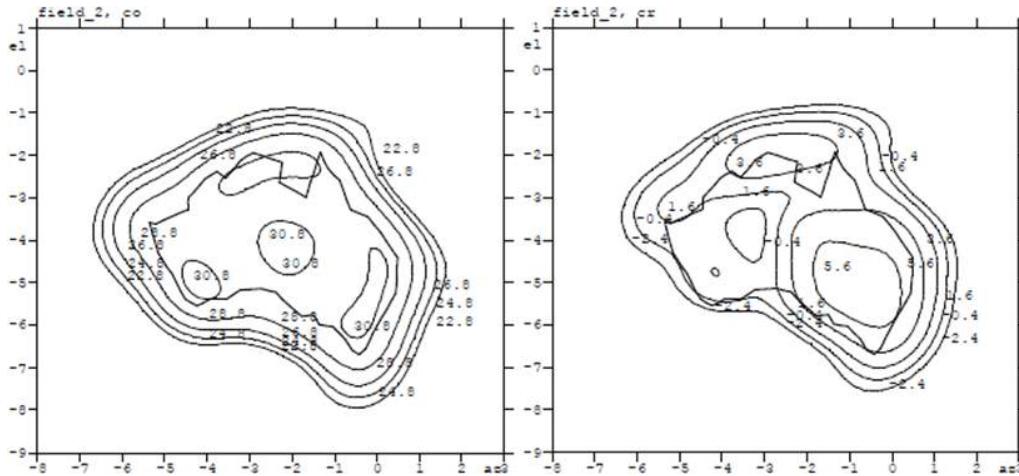


Figure 6-59 Co-and cross-polar components of the field from the rear reflector illuminated by the rear feed and the currents on the front reflector.

6.4.3.3 Transmission of the rear reflector field through the front reflector, field_3

The field scattered by the rear reflector, field_2, is then transmitted through the front reflector.

This analysis requires a calculation of the induced currents on the front reflector generated by the currents on the rear reflector. Using PO for this type of analysis, and especially for closely spaced reflectors, can be very time consuming. We therefore introduce an intermediate step, a plane wave expansion. The field from the rear reflector is a focused field with an almost constant phase. This means that it can be represented by a relatively small number of plane wave modes. These modes are then used to calculate the currents on the front reflector.

The new object with the name PWE and belonging to the PLANE WAVE EXPANSION class is defined as shown in Figure 6-60:

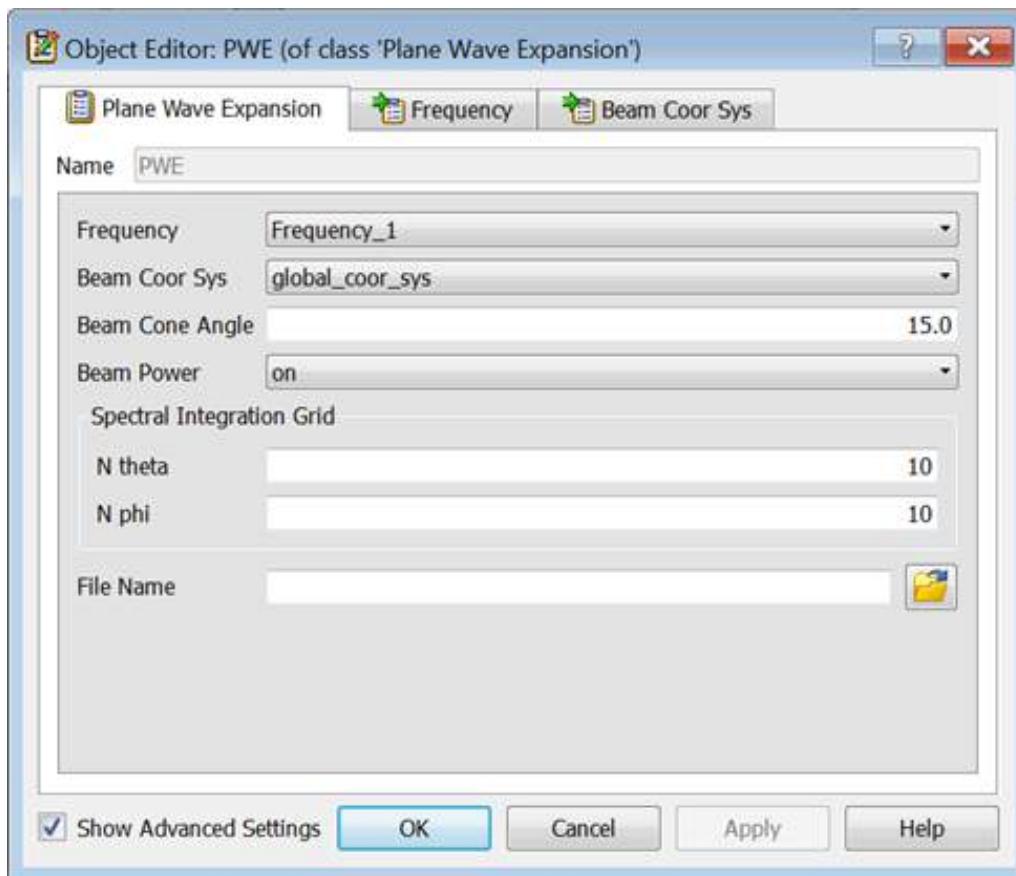


Figure 6-60 Plane wave expansion object definition.

The BEAM_CONE_ANGLE is the angular extent of the spectrum used in the plane wave expansion. It shall include all power from the source to be expanded and the previous results show that an angle of 15° is sufficient.

To compute field_3 the commands shown in Figure 6-61 must be defined.

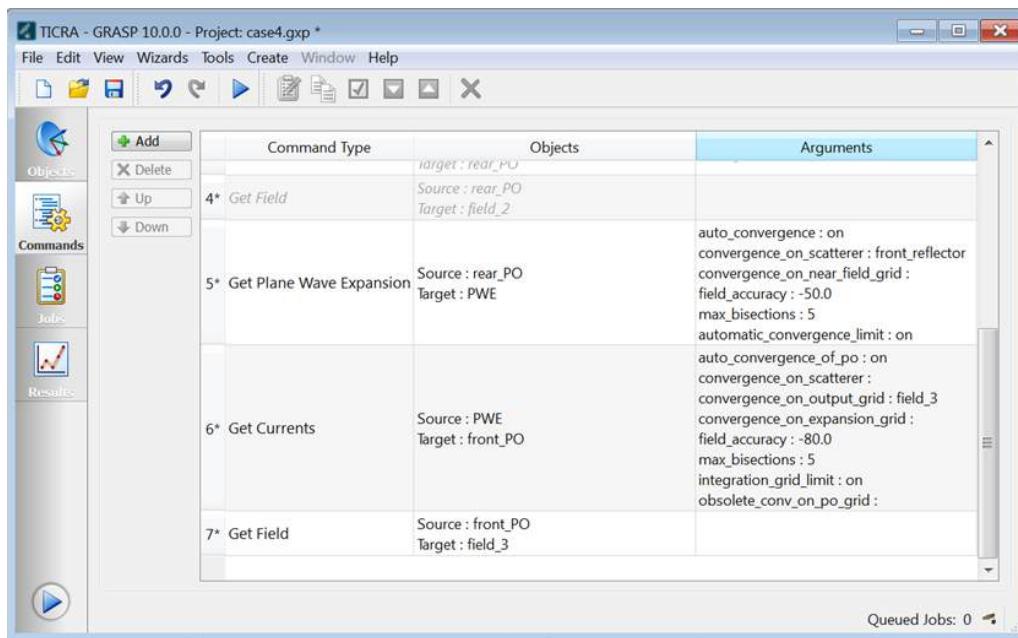


Figure 6-61 Commands to compute field_3.

The plane wave expansion of the field from the rear reflector is calculated. The number of plane wave modes is found by the automatic internal convergence procedure using test points on the front reflector as convergence criterion. The PO currents on the front reflector are then calculated when illuminated by the plane wave expansion of the rear reflector currents. The number of PO patches is found by the automatic internal convergence procedure using the far field points in field_3 as convergence criterion.

The above commands are run under Job_03. The result is shown in Figure 6-62, together with the coverage polygon (Australia).

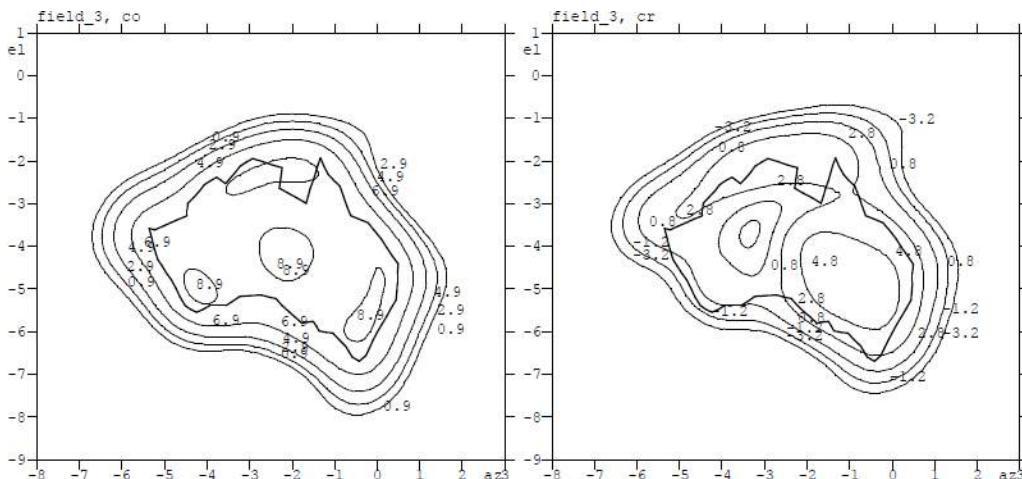


Figure 6-62 Co-and cross-polar components of the field from the front reflector illuminated by the plane wave expansion of the currents on the rear reflector.

6.4.3.4 Second reflection by the rear reflector, field_4

The field scattered by the rear reflector, field_2, is transmitted through the front reflector, see the computation above, but also partly reflected. This reflected field is reflected again by the rear reflector and produces field_4. To compute field_4 the commands shown in Figure 6-63 must be defined.

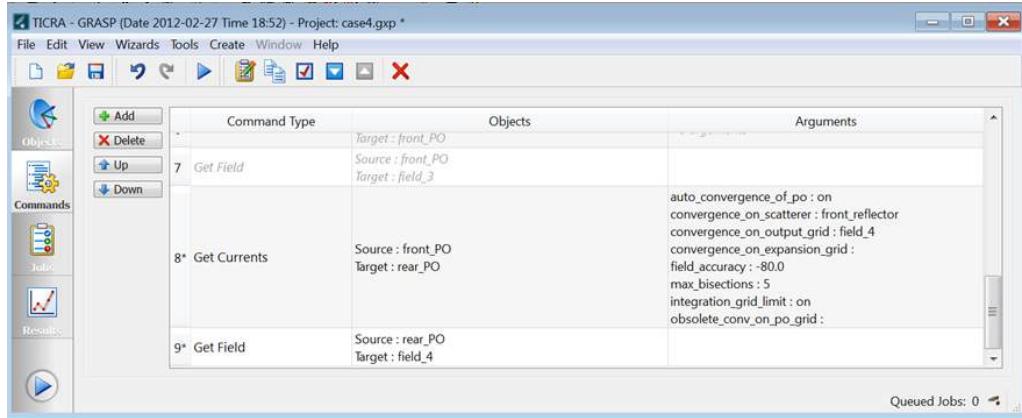


Figure 6-63 Commands to compute field_4.

The PO currents on the rear reflector are calculated when illuminated by FRONT_PO. The number of PO patches are found by the automatic internal convergence procedure using both the far field grid and the front reflector as convergence criterium. The above commands are run under Job_04.

6.4.3.5 Second transmission through the front reflector, field_5

Finally, the field scattered by the rear reflector, field_4, is transmitted through the front reflector and produces field_5. This analysis requires a calculation of the induced currents on the front reflector generated by the currents on the rear reflector and is time consuming. A plane wave expansion could be used, but the beam cone angle should now be much larger than the one considered before, since the field is not focused, and thus will not allow us to save time. We use therefore the commands shown in Figure 6-64 and run Job_05.

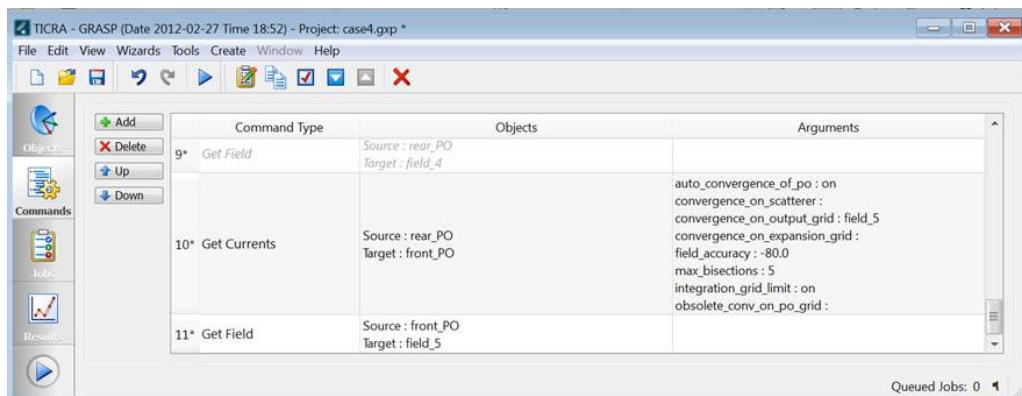


Figure 6-64 Commands to compute field_5.

6.4.3.6 Total field

The contributions calculated in the five preceding sections shall now be added to find the total field from the dual gridded reflector with rear feed in operation. The necessary commands are found in Figure 6-65 and run under Job_06.

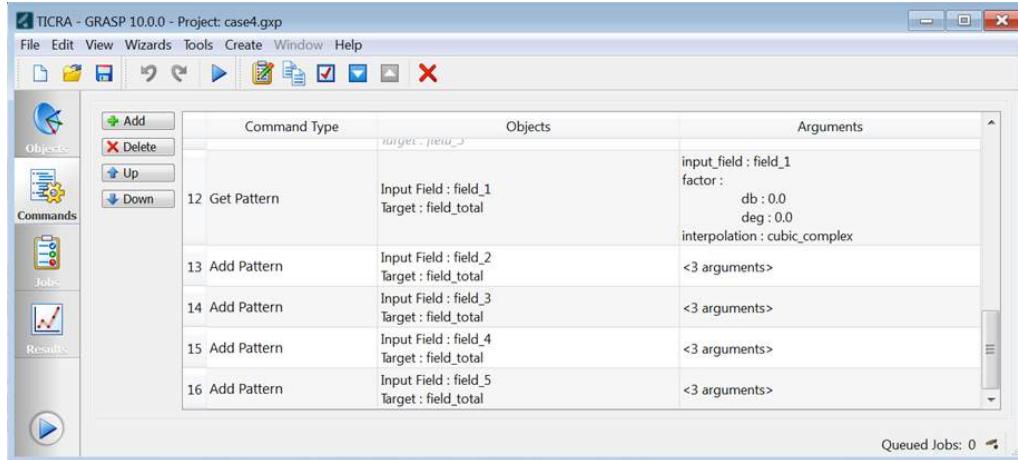
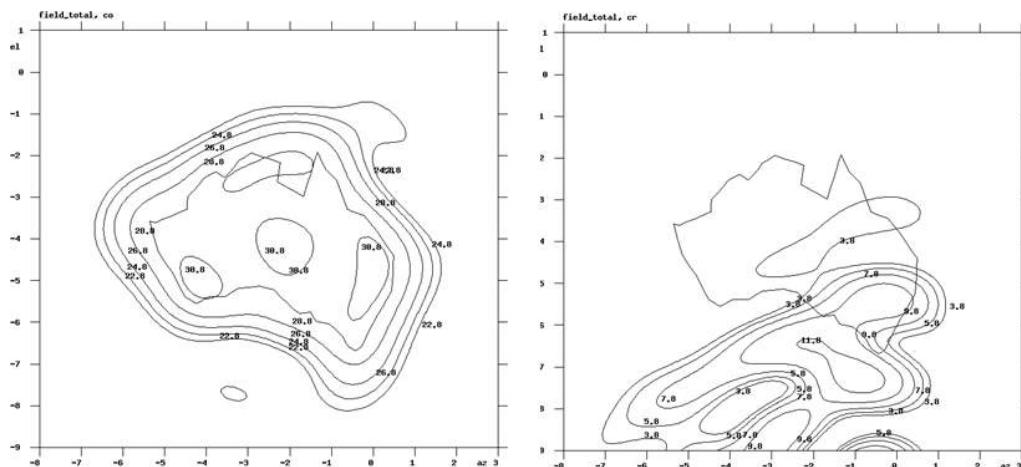


Figure 6-65 Commands to compute the total field.

The total co- and cross-polar components are shown in Figure 6-66. By the comparison with Figure 6-57 it is seen that the cross polarisation is almost exclusively generated by the first reflection in the front reflector (field_1).



illumination is obtained by optimising the amplitude and phase of the feeds. As in the previous section this is done by the POS program.

A GRASP project can be opened from the box TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The project files are located in the TEST-CASES/SINGLE_REFL_WITH_FEED_ARRAY subdirectory in the installation directory.

6.5.1 Creation of the Geometry

In Figure 6-67 the antenna geometry is shown. In the following the steps for obtaining this geometry will be described.

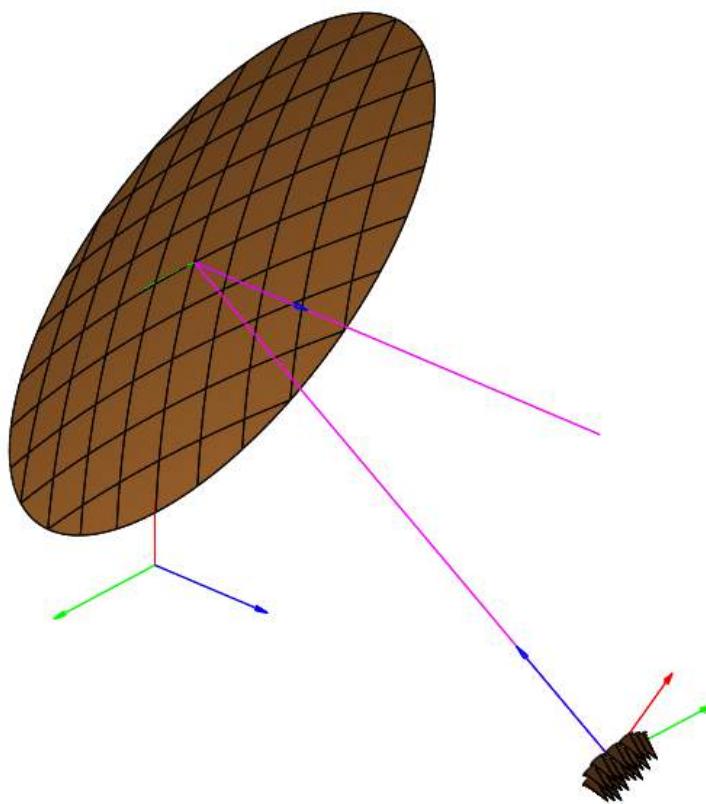


Figure 6-67 Drawing of a multifeed, parabolic reflector.

First, the Single reflector wizard is opened and the data shown in Figure 6-68 are provided as an initial definition of the reflector. The corresponding objects are created by pressing the button GENERATE OBJECTS. In order to obtain the geometry in Figure 6-67 a number of additional objects is necessary. The creation of the objects is automatically done if the user has the POS program available, however, in this section the steps will be described as if they were done manually.

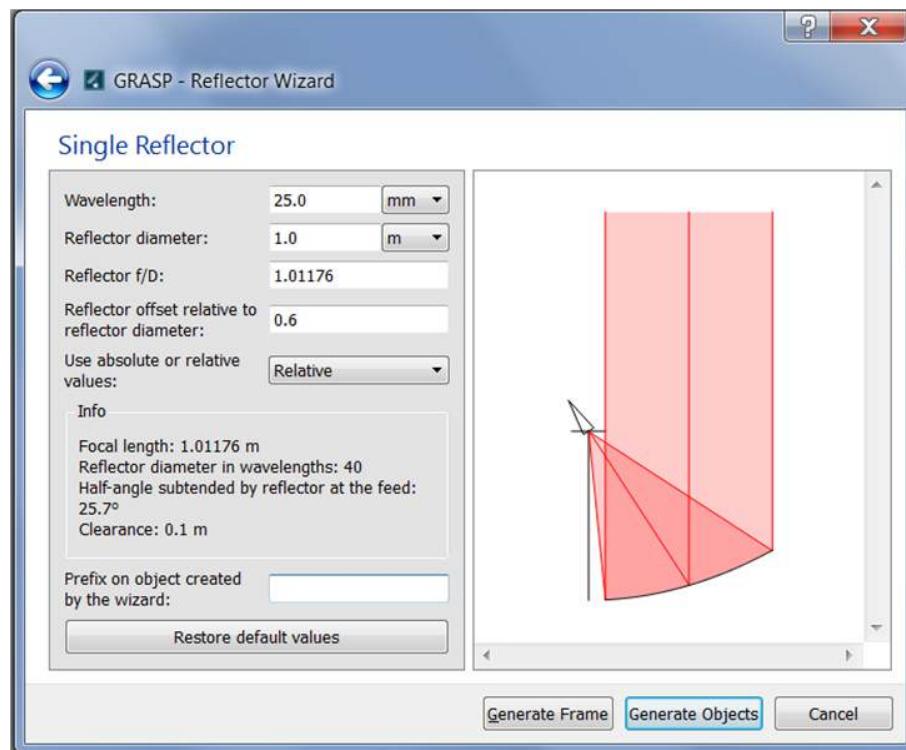


Figure 6-68 Wizard with initial data for single reflector.

Since feeds are arranged in an hexagonal grid and illuminate the reflector, circular aperture horns are the most appropriate feeds. A new object of CONICAL HORN class is thus opened and the attributes are given the values shown in Figure 6-69. The object is given the name ELEM_FEED_TYPE and the dialogue is closed by clicking OK.

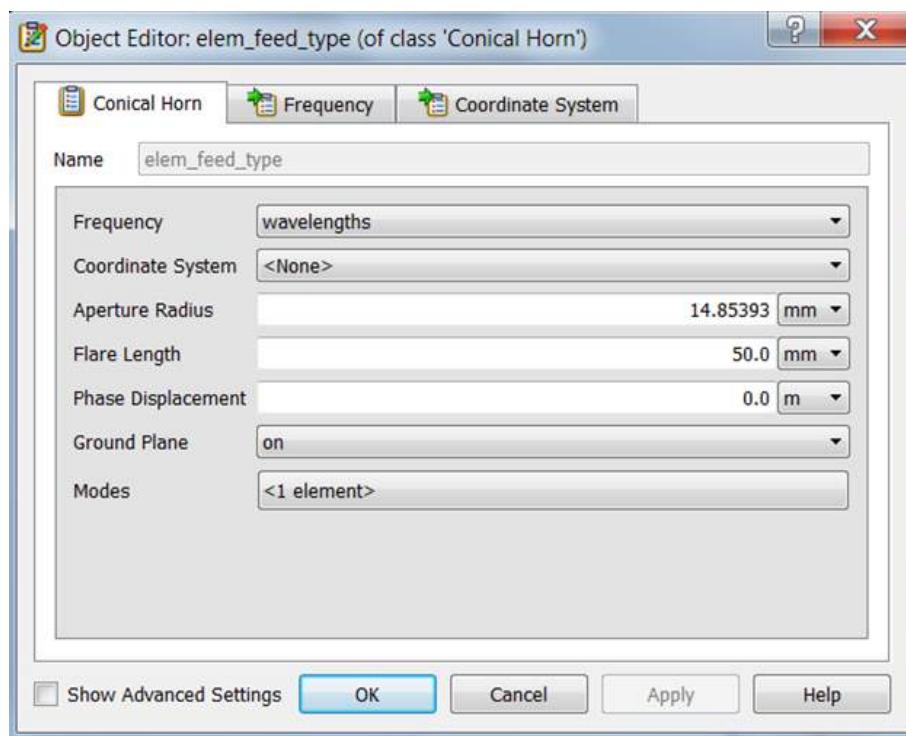


Figure 6-69 Data for the conical horn feed element.

This object will now be the feed element in the array. The next step is to define the array itself. A new object of TABULATED PLANAR GRID ARRAY class is opened and the attributes are given the values shown in Figure 6-70.

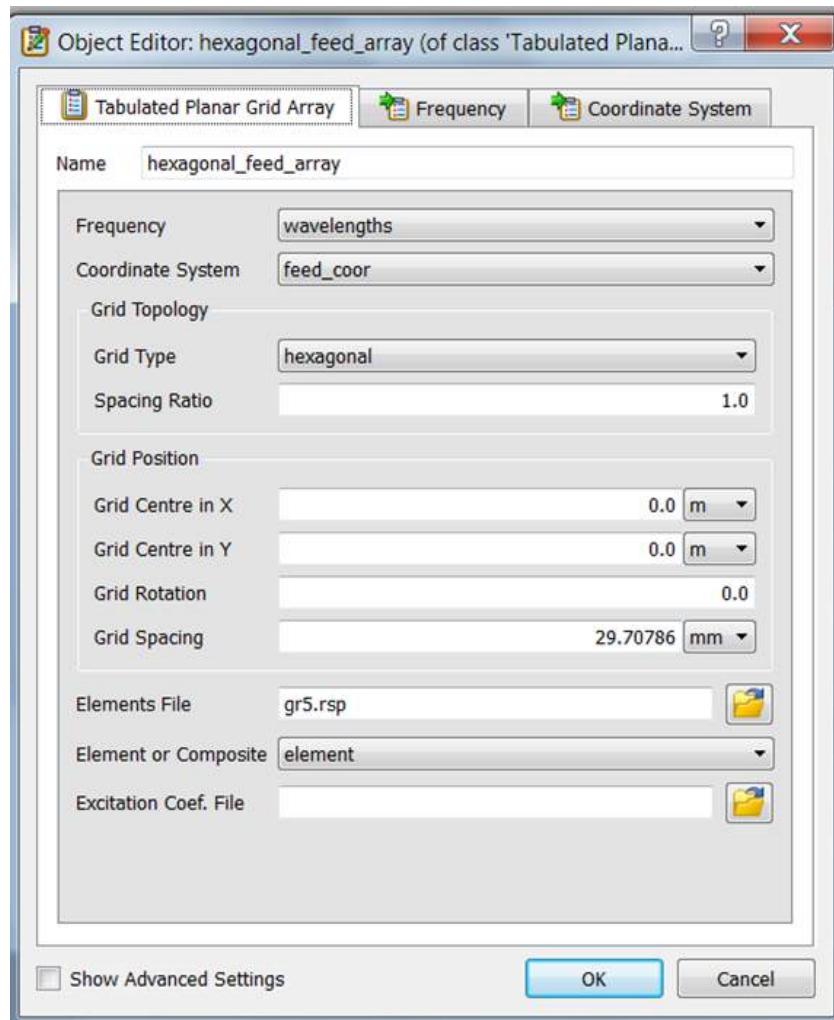


Figure 6-70 Data for the tabulated, planar grid array.

The file, GR5.RSP, contains the positions of the elements in the array. This file is very conveniently generated by POS. The object is given the name HEXAGONAL_FEED_ARRAY and the dialogue is closed by clicking OK. This object will be the source for calculating each beam separately (ELEMENT_COMPOSITE attribute).

This completes the definition of the multifeed reflector geometry shown in Figure 6-67.

6.5.2 Analysis of the Geometry

In this section the analysis of the multifeed reflector antenna is presented. The wizard (Figure 6-68) generated a FIELD_STORAGE object of SPHERICAL_CUT class, hence assuming a field calculation in a cut. However, in order to analyse a multifeed reflector it is necessary to calculate the field in a grid on the far field sphere. For this reason a FIELD_STORAGE object of SPHERICAL_GRID class must be created. In Figure 6-71 a listing of the appropriate

object is shown. To the attributes which are not shown in the listing default values are assigned.

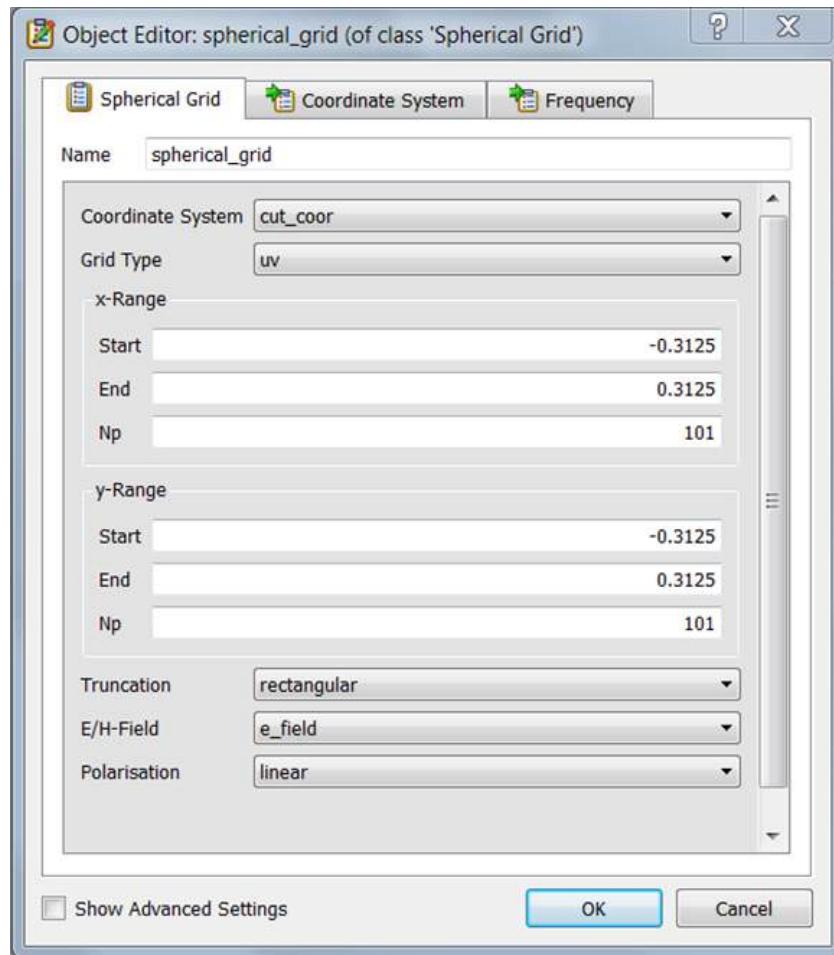


Figure 6-71 Data for the spherical output grid.

The field is calculated in 101 by 101 points in a uv-grid (default) with limits ± 0.3125 ($\pm 18.2^\circ$).

To carry out the analysis the commands generated by the wizard must be modified as shown in Figure 6-72.

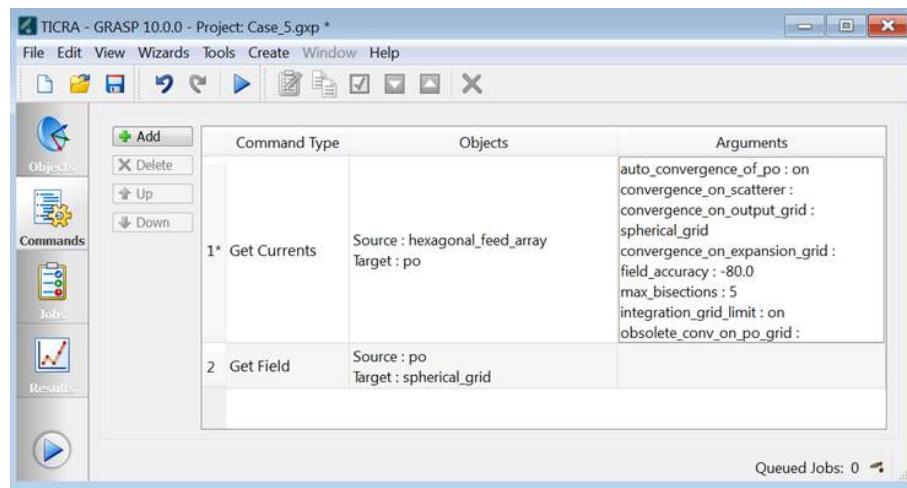


Figure 6-72 Command sequence for the multifeed reflector antenna.

The two commands are executed under Job_01 and the generated beams can be visualized in the RESULTS window. However, here only one beam can be displayed at a time and it is therefore more convenient to use the POSTPROCESSOR for which a project file, CASE5_JOB_01_ELEMENTS.PPC, is prepared in advance (in the subdirectory JOB_01).

The field data can be displayed by opening the project file in the POSTPROCESSOR (press F2 (takes a few seconds), F3 and F4 after reading the project file). The result is Figure 6-73. The element beams are shown in black, the polygon that will be input to POS for optimising the feed excitation coefficients in red and the map and the edge of the Earth in blue.

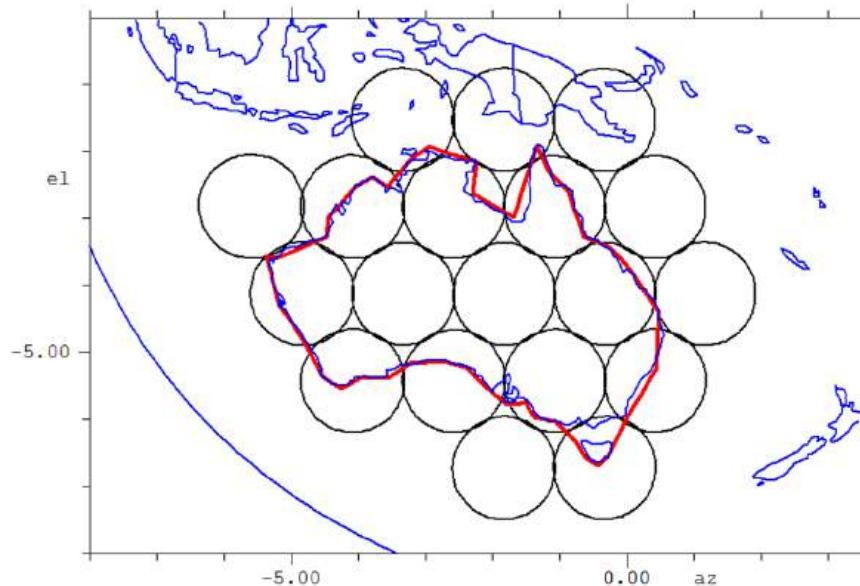


Figure 6-73 Footprint of element beams on Australian coverage.

It is also of interest to calculate the composite beam. For this purpose HEXAG-

ONAL_FEED_ARRAY is modified such that the attribute ELEMENT or COMPOSITE is set to COMPOSITE and the attribute EXCITATION COEF. FILE is set to GR5.EXI. The file GR5.EXI contains the array excitation coefficients and has been obtained by POS. The object HEXAGONAL_FEED_ARRAY can now be used to do the analysis of the composite beam.

The same two commands as before are run under Job_02 and the result can be displayed by the POSTPROCESSOR using the project file CASE5_JOB_02_COMPOSITE.PPC. The composite beam is shown in Figure 6-74.

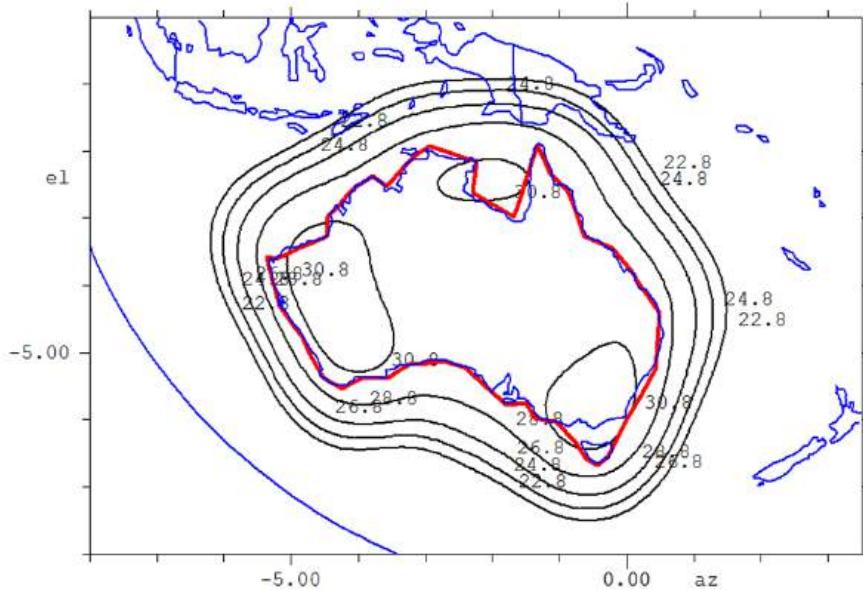


Figure 6-74 Optimised contoured beam. Contours at 1, 3, 5, 7 and 9 dB below maximum (31.7 dBi).

6.6 Compact Antenna Test Range with Serrated Edges

In this chapter a dual reflector geometry operating as a Compact Antenna Test Range (CATR) is analysed. The requirement of low edge diffraction necessitates the use of the serrated edge option in GRASP. A GRASP project can be opened from the box TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The project files are located in the TESTCASES/CATR_WITH_SERATED_EDGES subdirectory in the installation directory.

6.6.1 Creation of the Geometry

Apart from the serrated edge, the CATR can be considered as a usual dual reflector geometry. Hence, the first step is to open the dual reflector wizard. In Figure 6-75 the values to be provided in this case are shown. We have here zoomed in on the sketch of the reflector system (by the mouse scroll wheel or by the +/- buttons of the keyboard).

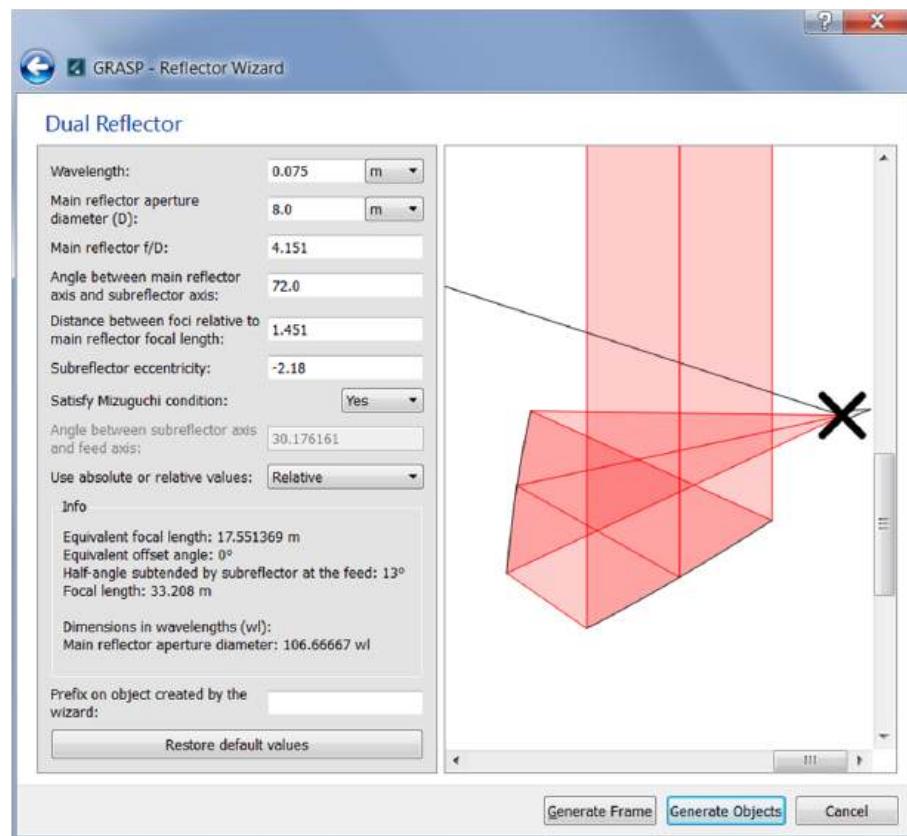


Figure 6-75 Wizard for creation of a dual, compensated reflector geometry.

This geometry is the so-called Side Fed Offset Cassegrain (SFOC) geometry. It is cross polar compensated which ensures low cross polarisation in the quiet zone.

As in the previous cases some objects must be modified and more objects must be created in order to enable us to analyse the CATR. The wizard automatically generates a source of GAUSSIAN BEAM, PATTERN DEF. class, however, in this case it is decided to use a source of SIMPLE TAPERED PATTERN ($M=1$) class. The attributes in this object must be defined as shown in Figure 6-76. At the angle 14.289109° from feed axis a taper of 2 dB is specified in both E- and H-plane.

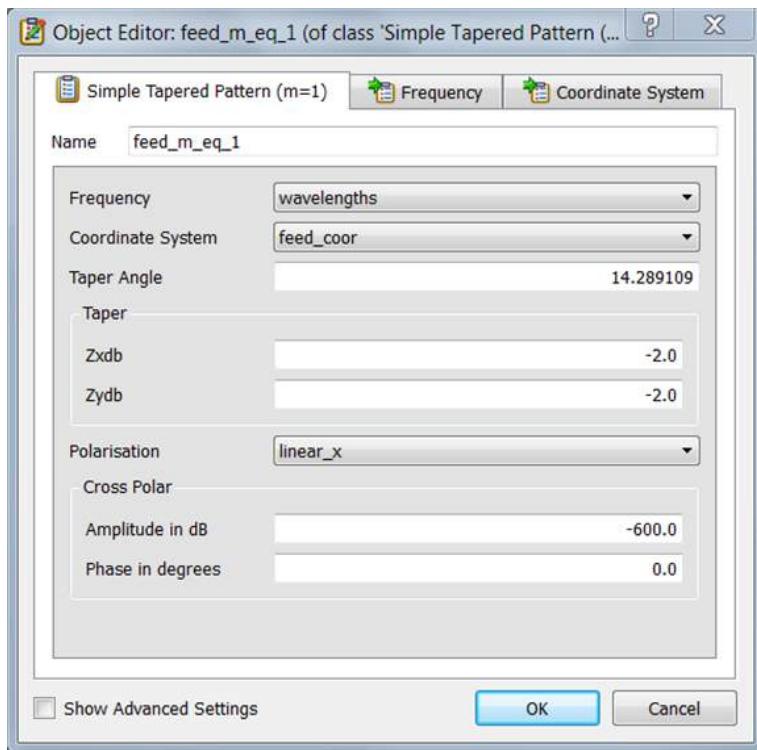


Figure 6-76 Definition of feed object for the CATR.

From the wizard two objects of ELLIPTICAL RIM class (one for the subreflector and one for the main reflector) have been created. As the reflector apertures in the CATR are rectangular, it is necessary to apply the RECTANGULAR RIM class for the definition of the sub- and main reflector rim. As shown in Figure 6-77 four new rim objects of this class are generated: two for the main reflector inner and outer rim (bottom and tip of serrations) and two for the subreflector inner and outer rim.

The centre of the main reflector rims is located on the x -axis of the basic coordinate system (position -38.79 m) and the side length of the inner/outer rim is 7.3/8.8 m along the x -axis and 6.0/7.5 m along the y -axis.

The centre of the subreflector rims is located on the x -axis of the coordinate system SUB_COOR (position 7.14 m) and the side length of the inner/outer rim is 8.14/9.16 m along the x -axis and 6.87/7.62 m along the y -axis.

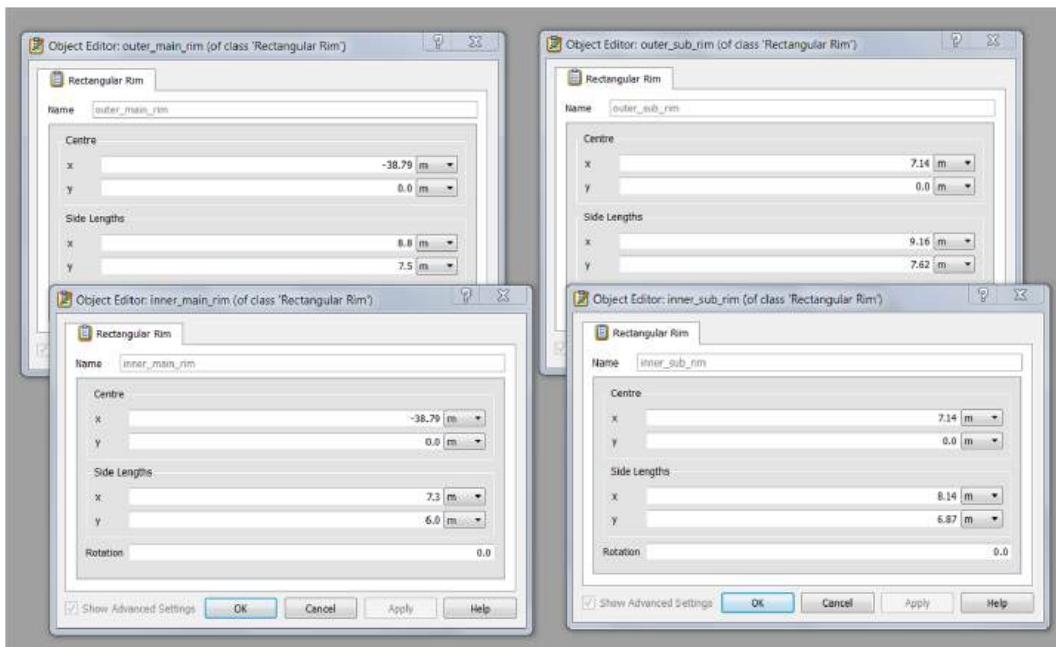


Figure 6-77 Definition of rectangular rim objects for the CATR.

The reflector objects, as generated by the wizard, do not contain any serration model. In order to analyse the influence of the serrations we define two sets of reflectors, one with and one without serrations. Thus the reflector objects have to be modified as follows: The RIM attribute in the MAIN_REFLECTOR object is changed to OUTER_MAIN_RIM. This object now defines a main reflector with rectangular but solid rim. To define a main reflector with a serrated edge, the object MAIN_REFLECTOR is duplicated as MAIN_REFLECTOR_SERRATED which is opened and under the SERRATION attribute the INNER_RIM struct member is set to the INNER_MAIN_RIM object. The SHAPE attribute is set to LINEAR.

Similarly an object SUB_REFLECTOR_SERRATED is created and modified to include the serrations in the same way as for the main reflector.

The definition of the CATR is hereby completed and a drawing of the geometry will result in Figure 6-78.

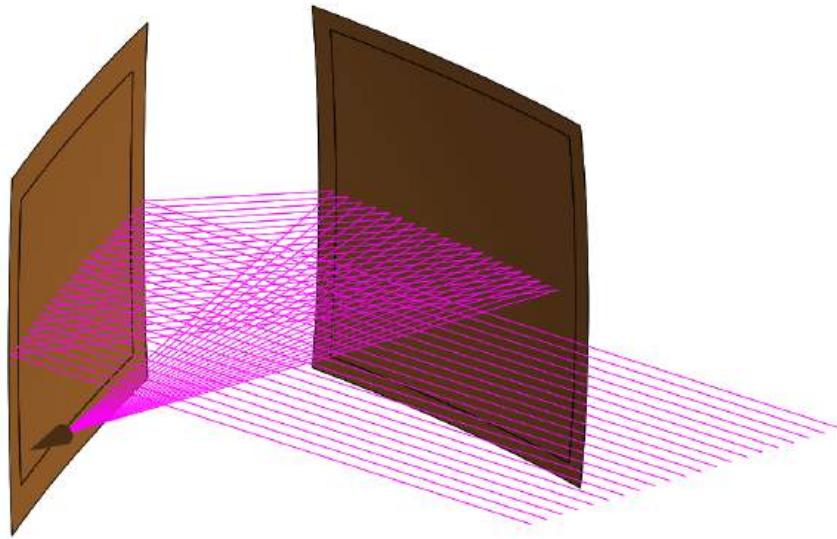


Figure 6-78 Example of a compact range with rays shown in the horizontal plane.

6.6.2 Analysis of the Geometry

The wizard automatically generates commands which carry out a far field (cut) analysis of the geometry assuming the use of PO on both sub- and main reflector. Using PO on both reflectors turns out to be very time consuming, due to the electrical dimensions of the reflectors and the short distance between the subreflector and the main reflector. Since in this example we are mainly interested in the influence of the main reflector serrations, we will use PO for the main reflector and GO for the subreflector. It is noted that using GO or GTD on the subreflector means that the serrations (attribute INNER RIM in the REFLECTOR specifications) are not taken into account, hence, they are here set only to give the user the possibility of making a PO analysis of the subreflector, if necessary. A new object of the *Single-Reflector GTD* class is defined as shown in Figure 6-79 .

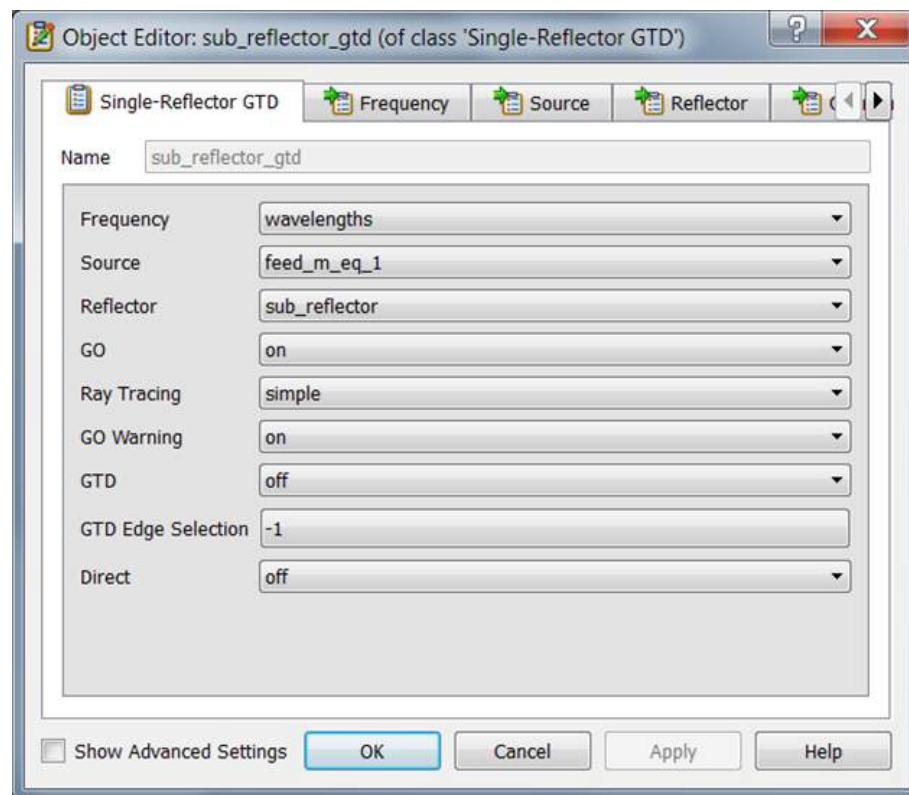


Figure 6-79 Definition of the subreflector GTD object.

As the purpose of a CATR is to generate a plane wave in the near field (quiet zone) it is most interesting to do an analysis in the near field. The object CUT generated by the wizard defines an analysis in the far field. Hence, a new object of the PLANAR CUT class, QUIET_ZONE_CUTS, is defined located at the position of the quiet zone which is 27 m in front of the main reflector. This object is shown in Figure 6-80.

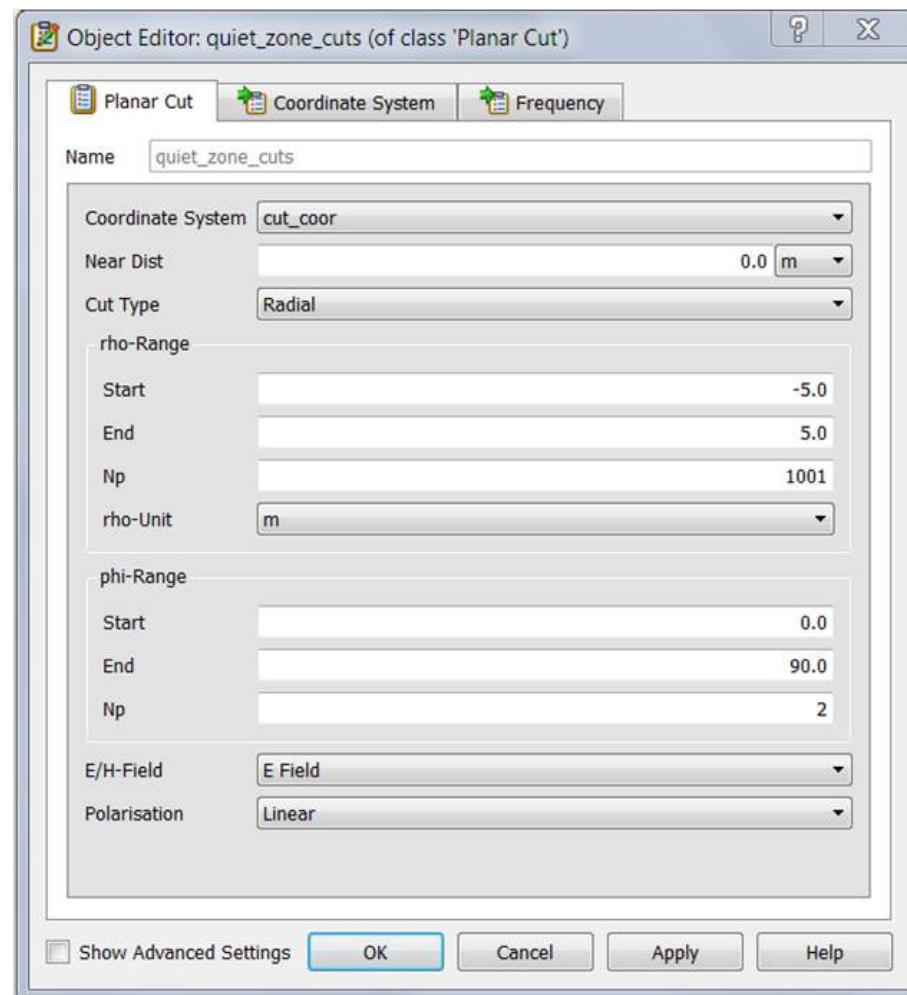


Figure 6-80 Definition of the planar cuts in the quiet zone.

The commands generated by the wizard shall be modified as shown in Figure 6-81.

	Command Type	Objects	Arguments
1*	Get Currents	Source : sub_reflector_gtd Target : main_po	auto_convergence_of_po : on convergence_on_scatterer : convergence_on_output_grid : quiet_zone_cuts convergence_on_expansion_grid : field_accuracy : -80.0 max_bisections : 5 integration_grid_limit : on obsolete_conv_on_po_grid :
2	Get Field	Source : main_po Target : quiet_zone_cuts	

Figure 6-81 Commands to calculate the field in the quiet zone.

The first command calculates the PO currents on the main reflector and the second command determines the field in the quiet zone generated by these currents. For the first run, the main reflector PO object MAIN_PO refers to the scatterer object MAIN_REFLECTOR which is the main reflector without serrations. Running the two commands (as Job_01) therefore calculates the field in the quiet zone when there are no serrations on the main reflector. Next, the

attribute SCATTERER in MAIN_PO is changed to MAIN_REFLECTOR_SERRATED and the same two commands are run as Job_02 now giving the field in the quiet zone when the main reflector serrations are included. The results of the calculations are presented in Figure 6-82. In these plots the curves for Job_02 are changed to red and copied into the results from Job_01 in order to facilitate the comparison.

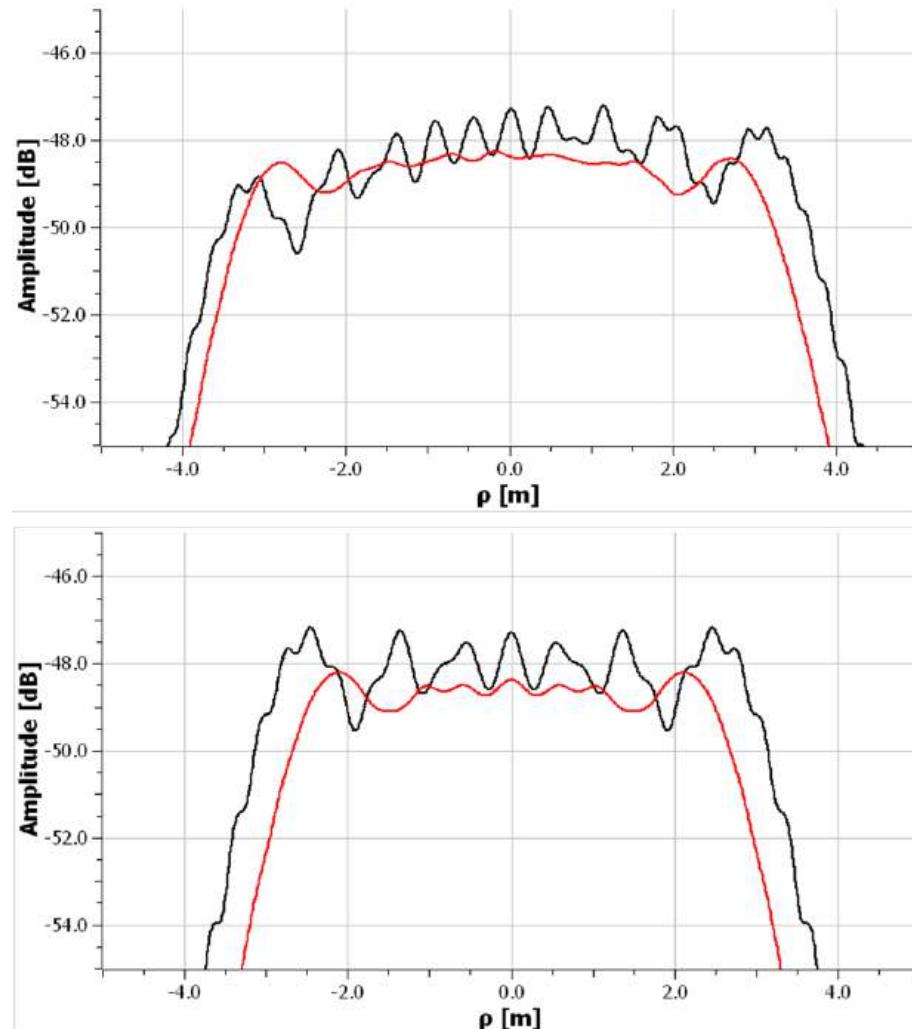


Figure 6-82 Fields in quiet zone of the CATR.
Top figure: horizontal plane.
Bottom figure: vertical plane.
Black curves: no serrations.
Red curves: with serrations.

6.7 Plane Wave Expansion

This section contains a short description of the theory behind the plane wave expansion and some examples of its use. The headings of the sections are:

- - Theory of plane wave expansion
- - Near field from a feed

- - Polarisation grid
- - Near field from a reflector

6.7.1 Theory of Plane Wave Expansion

According to e.g. Collin & Zucker, "Antenna Theory"¹, an arbitrary electric field \vec{E} can be expanded in a spectrum of plane waves of the form

$$\vec{E}(r) = \frac{1}{4\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \vec{f}(k_x, k_y) e^{-j\vec{k}\cdot\vec{r}} dk_x dk_y, \quad (6.1)$$

where \vec{k} is the vector

$$\vec{k} = k_x \hat{x} + k_y \hat{y} + k_z \hat{z} = k_x \hat{x} + k_y \hat{y} + \sqrt{k^2 - k_x^2 - k_y^2} \hat{z} \quad (6.2)$$

where k is the wave number and $\vec{f}(k_x, k_y)$ is the amplitude density function, which is related to the far field by

$$\vec{E}_{far}(k_x, k_y) = j \frac{kk_z}{2\pi} \vec{f}(k_x, k_y). \quad (6.3)$$

The variables $k_x k_y k_z$ are related to the standard spherical angles θ and ϕ by

$$\begin{aligned} k_x &= k \sin \theta \cos \phi \\ k_y &= k \sin \theta \sin \phi \\ k_z &= k \cos \theta. \end{aligned} \quad (6.4)$$

In this way the near field (6.1) can be calculated from the far field (6.3) by numerical integration. The representation (6.1) of the near field is only exact if the integral includes the visible $k_x^2 + k_y^2 \leq k^2$ as well as the invisible area $k_x^2 + k_y^2 > k^2$ of the spectral domain, but since the spectrum $\vec{f}(k_x, k_y)$ is normally not known outside the visible area it is not included in the present software implementation. As a consequence, the plane wave expansion (6.1) is only accurate if the far field is reasonably directive and has decreased to a low level at $\theta = 90^\circ$.

A further truncation is possible by means of the attribute BEAM CONE ANGLE which limits the spectrum to the far-field cone $\theta \leq \theta_o$, θ_o being the BEAM CONE ANGLE, see Figure 6-83. Such a truncation will improve the computational speed and is convenient if it is known that nearly all power is contained within $\theta \leq \theta_o$. The angle θ_o is measured from the z -axis of the coordinate system specified in BEAM COOR SYS. For that reason the orientation of this coordinate system is important and the z -axis should point towards the beam maximum of the far-field pattern to capture as much power as possible within the cone $\theta \leq \theta_o$.

¹ R.E. Collin and F.J. Zucker, "Antenna Theory, part 1", McGraw-Hill Book Company, New York 1969

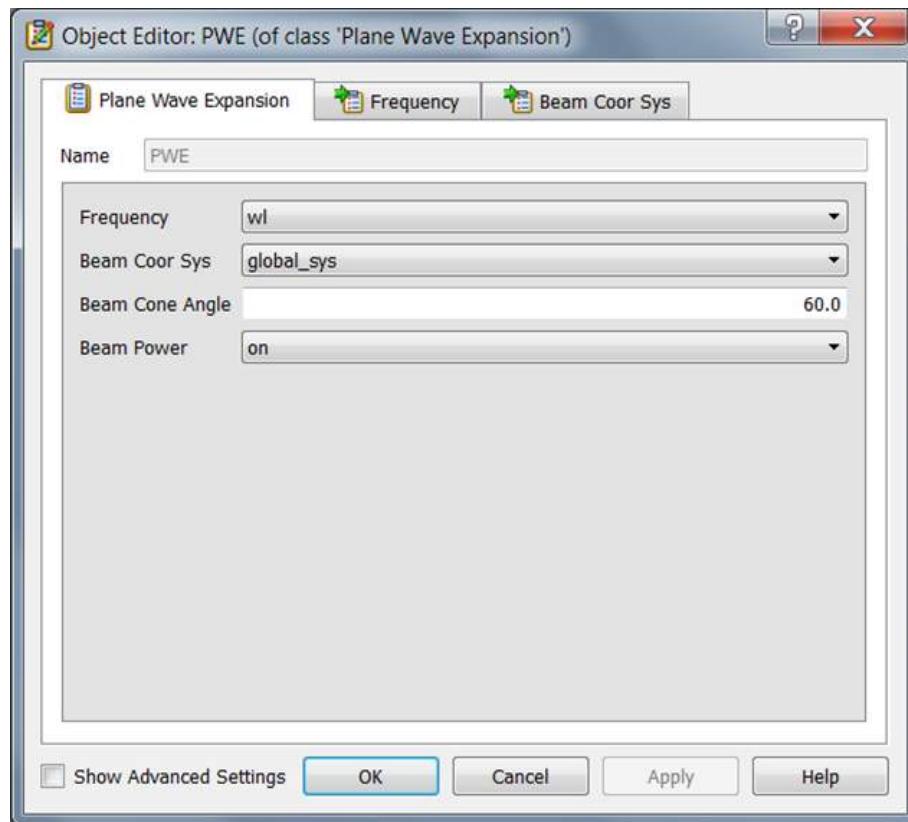


Figure 6-83 Editor for the Plane Wave Expansion object.

The convergence properties of a plane wave expansion is rather different from that of induced or equivalent currents. When a field is represented by currents, the field can be calculated in the far field as well as in the near field except on the surface on which the currents are located. If the observation point approaches this surface very many current elements will be needed and the convergence becomes very slow. As opposed to the currents, the plane wave expansion is most rapidly converging close to the source, and it cannot be used in the far field. An increasing number of samples is needed in the numerical integration of (6.1) as the observation point moves away from the source. The plane wave expansion is thus a useful tool if the near field is needed very close to a source. The following examples illustrate the most important properties of the plane wave expansion.

6.7.2 Near Field from a Feed

This example illustrates some basic properties of the plane wave expansion. A GRASP project can be opened from the box TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The project files are located in the TEST-CASES/PWE_NEAR_FIELD_FROM_FEED subdirectory in the installation directory.

A feed radiating an ideal Gaussian beam is shown below in Figure 6-84. The beam has a far-field taper of -12 dB at $\theta = 30^\circ$ corresponding to a waist radius of $w_0 = 0.704\lambda$. $\lambda = 1 \text{ cm}$ in this example.

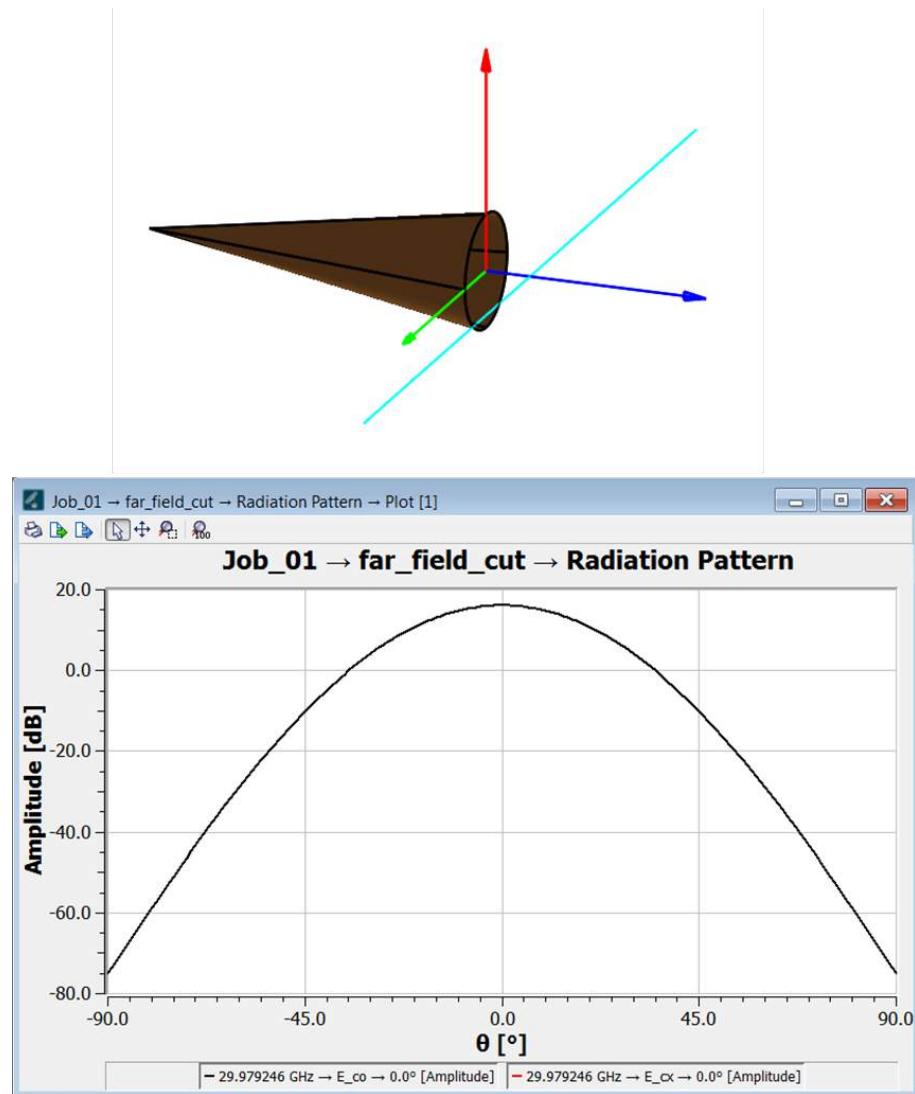


Figure 6-84 Feed horn and the near field cut shown as a line in light blue (top) and far-field pattern (bottom).

The near field cut is defined as shown in Figure 6-85.

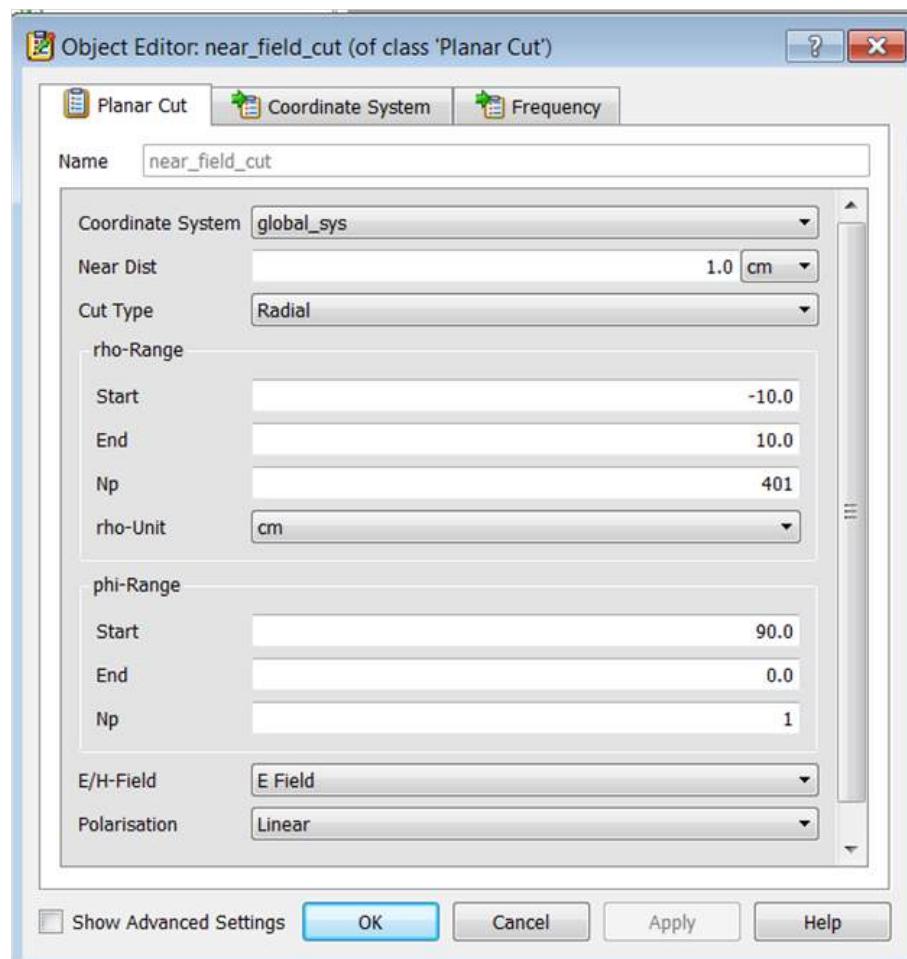


Figure 6-85 Near field cut definition.

The near field is now calculated by means of the plane wave expansion. Since the feed pattern is a simple Gaussian beam the exact near field is also known and can be compared to the result of the plane wave expansion. The commands needed to calculate the near field from the plane wave expansion are shown in Figure 6-86. These commands are run three times with the BEAM CONE ANGLE equal to 40°, 50° and 60°.

The results are compared in Figure 6-87 with different truncation angles θ_o of the plane wave expansion. It is seen that a truncation of the far field (spectrum) at $\theta_o = 40^\circ$ gives an accurate reconstruction of the near field down to approximately 35 dB below peak and that larger truncation angles rapidly improves the accuracy.

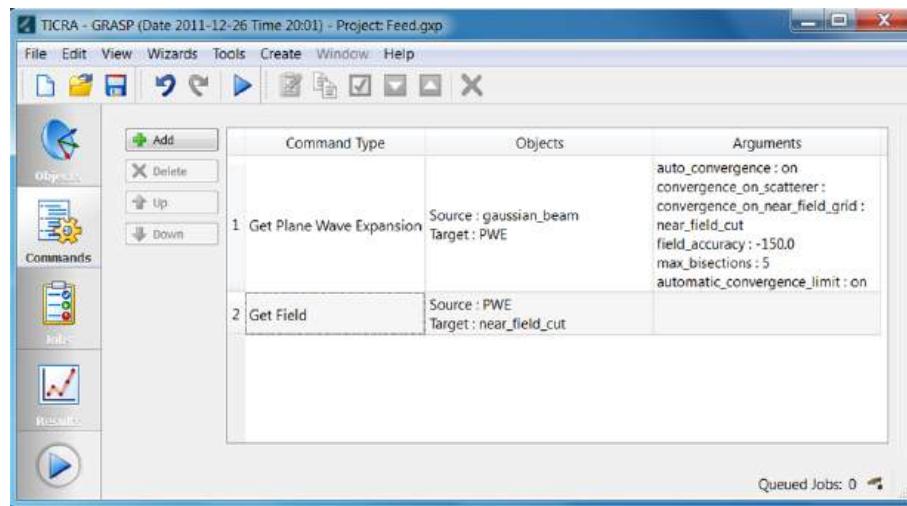


Figure 6-86 Commands for generating the plane wave expansion and the associated near field.

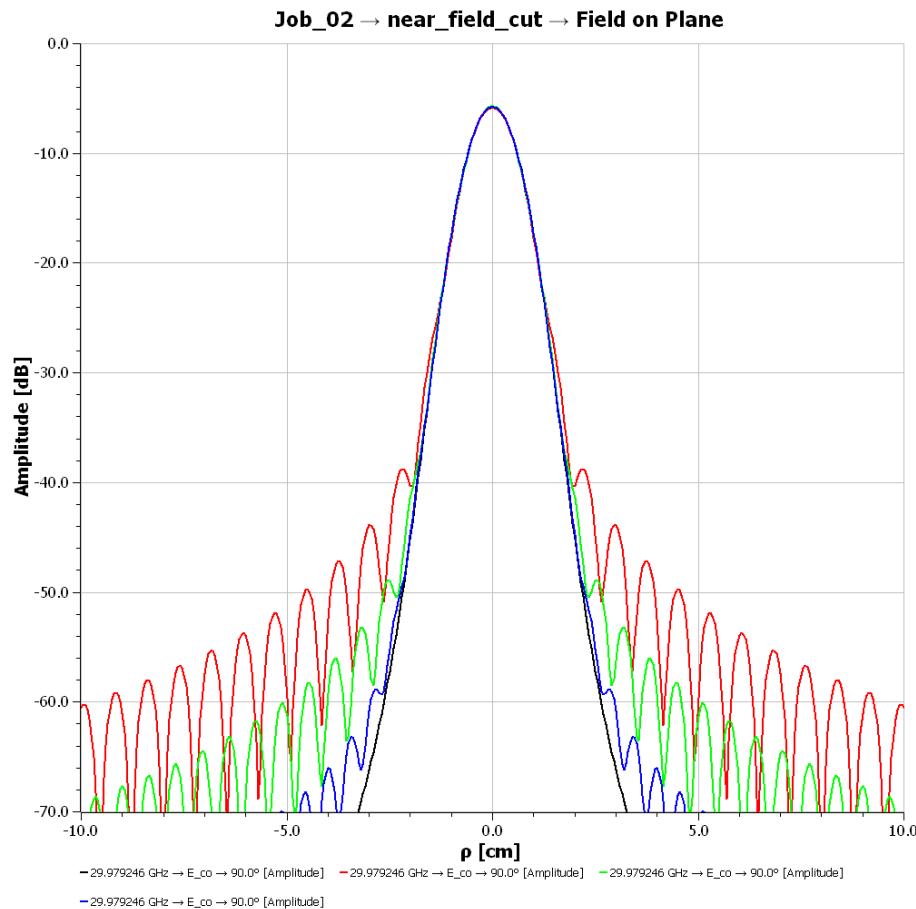


Figure 6-87 Near field from a Gaussian beam feed
 Black curve: exact Gaussian beam near field
 Red curve: PWE with $\theta_o = 40^\circ$
 Green curve: PWE with $\theta_o = 50^\circ$
 Blue curve: PWE with $\theta_o = 60^\circ$

The power of the plane wave spectrum is output when BEAM POWER is set

to ON. This power is given relative to the full power (4π watt) of the horn in the following table. The power value is a useful tool in the evaluation of the accuracy of the spectrum and thus of the determined near field.

θ_o	40°	50°	60°
Relative power	0.9921	0.9994	0.99997

Table 6.1 The power in the plane wave expansion for different truncation angles relative to the total power of the feed.

6.7.3 Polarisation Grid

In this example the plane wave expansion is used to improve the accuracy of the PO-analysis of non-perfectly conducting reflector surfaces.

A GRASP project can be opened from the box TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The project files are located in the TEST-CASES/PWE_POLARISATION_GRID subdirectory in the installation directory.

The analysis by PO of scatterers with special surface materials (e.g. polarisation grids and dielectric layers) is based on reflection and transmission coefficients given for an infinite planar surface. The approximations introduced are that the surface can be considered to be locally planar and that the incident field locally is a plane wave. The first approximation is normally satisfied because the radius of curvature of the surface is normally much larger than the wavelength. The second approximation is more problematic, because the incident field may have a complicated behaviour as e.g. close to a beam waist. In this case it is important to expand the incident field in plane waves which are then treated separately in the PO analysis of the surface.

An example is shown below in Figure 6-88. A Gaussian beam is radiated by the feed and reflected by a solid ellipsoidal mirror. Hereafter the beam reaches an ideal polarisation grid where most of the power is reflected, since the polarisation of the Gaussian beam and the direction of the wires in the grid are both orthogonal to the plane of symmetry of the system (i.e. the plane of the beam path).

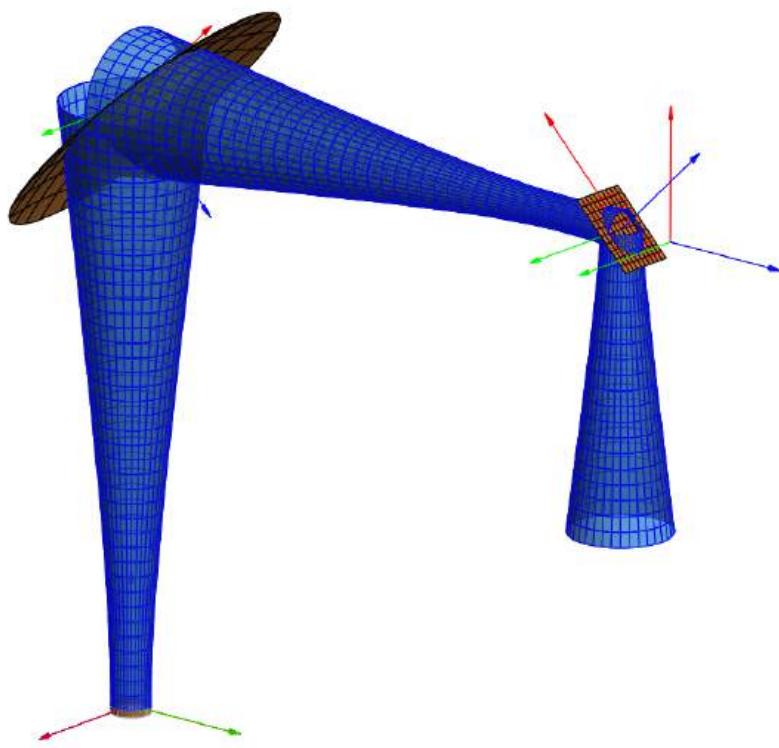


Figure 6-88 An ideal polarisation grid located close to a beam waist.

A minor part of the field is, however, transmitted through the grid. Ideally, this field should be zero, but since the incident field is not a single perfect plane wave and since the solid reflector generates some cross-polarisation the grid is not fully able to block for transmission. The results for the transmitted field are shown in Figure 6-89, computed with the standard PO method and with an expansion of the incident field in plane waves. The polarisation component shown is parallel to the wires of the grid and is thus not caused by the cross-polarisation of the solid reflector. It is seen that the correct result obtained by the plane wave expansion is about 30 dB lower than by the standard PO method. If the grid is placed so far away from the waist that the incident field can be considered as a spherical wave, the agreement between the two methods becomes much better. In general we recommend using a plane wave expansion of a source when the source illuminates a reflector with non perfectly conducting surface materials. The standard PO method is, however, adequate when the reflector is clearly in the far field of the source.

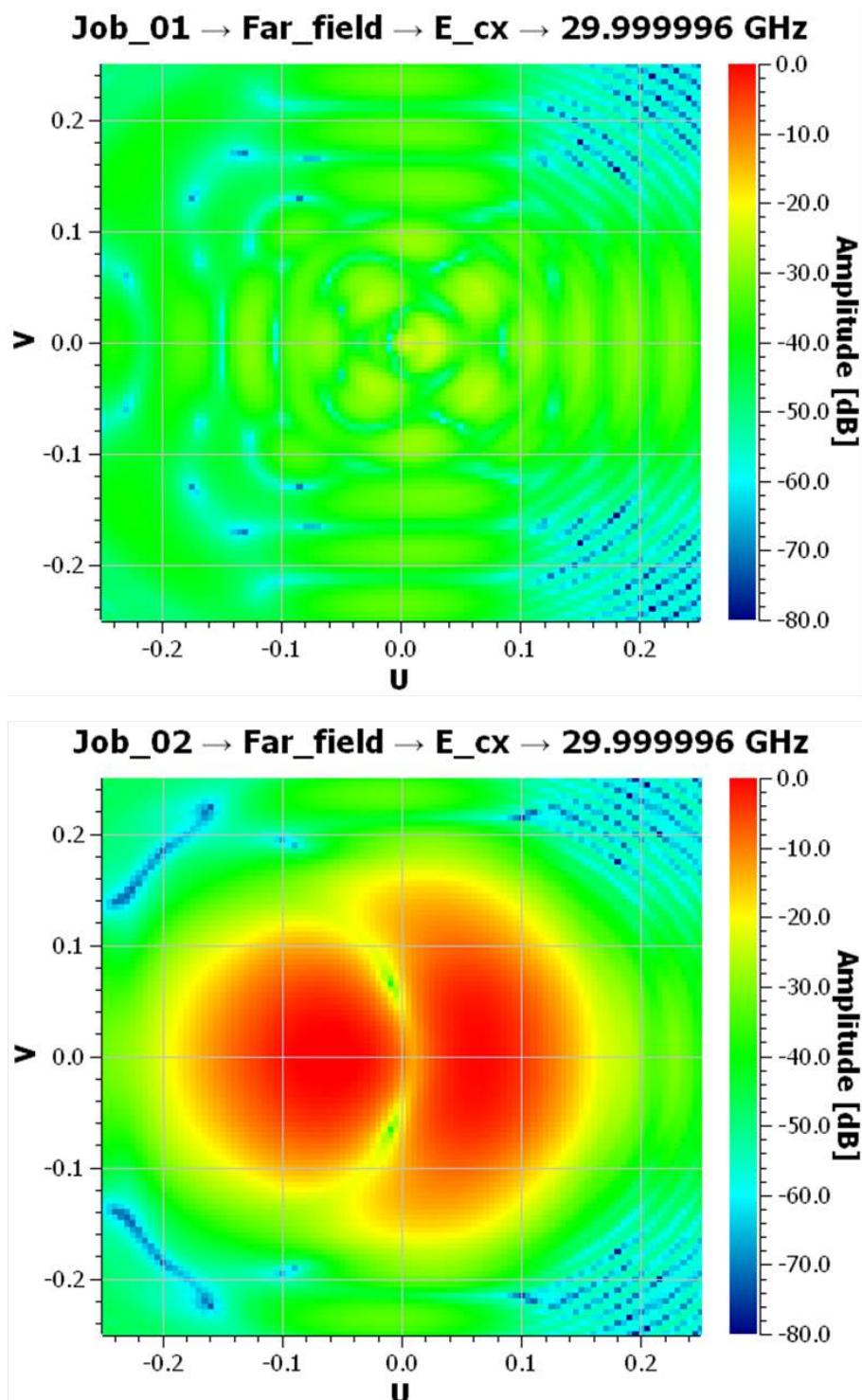


Figure 6-89 Transmitted field computed by plane wave expansion (top) and by standard PO (bottom).

The beam in Figure 6-88 is reflected at a right angle at the ellipsoidal mirror and the feed and polarisation grid are located in the two foci of the ellipsoid. These foci are both located at a distance of 200 cm from the centre of the mirror. The feed is an ideal Gaussian beam with waist radius of $w_o = 4.17$ cm radiating with a wavelength of 1 cm. For the plane wave expansion the BEAM CONE ANGLE is selected as 15° which captures the fraction 0.999997 of the radiated power from the reflector.

The commands needed to perform the calculations including the plane wave expansion are shown in Figure 6-90. The first command calculates the currents on the ellipsoidal reflector. The source is the FEED and convergence is requested on the FAR_FIELD grid points and on the plane wave expansion Pw_EXP. This means that the field from the reflector must converge both in the requested far-field grid and also within the cone in the far field determined by the BEAM_CONE_ANGLE attribute in Pw_EXP. It is often useful to include the convergence target CONVERGENCE_ON_EXPANSION_GRID because it is then ensured that the field from the reflector is converged in far-field region needed for the following plane wave expansion.

The second command determines the plane wave expansion of the field from the ellipsoidal reflector. The third command calculates the induced currents on the polarisation grid using the plane-wave expansion as the source. The two last commands add the contributions from the reflector and the polarisation grid.

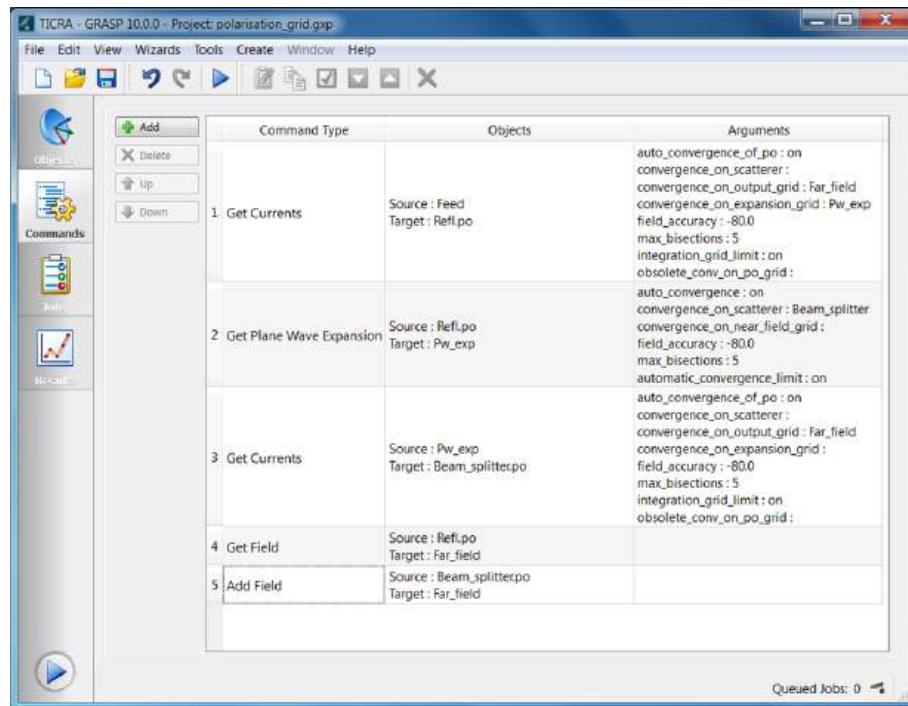


Figure 6-90 Commands including the plane wave expansion.

The same commands are used for the standard PO calculation, except that the second command is deactivated and the currents from the ellipsoidal reflector, REFL.PO, are used directly as source for the third command.

6.7.4 Near Field from a Reflector

This example illustrates that the near field very close to a large reflector can be computed efficiently by a plane wave expansion.

A GRASP project can be opened from the box TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The project files are located in the TEST-

CASES/PWE_NEAR_FIELD_FROM_REFL subdirectory in the installation directory.

A rotationally symmetric parabolic reflector is shown below in Figure 6-91. The diameter of the reflector and the focal length are both 2 m and the wavelength is 0.005 m, corresponding to a frequency close to 60 GHz. The feed is a Gaussian beam with an edge taper of -12 dB. The near field is calculated in planar radial cuts (shown in light blue in Figure 6-91) located 15 cm in front of the centre of the reflector corresponding to 2.5 cm in front of the plane of the rim.

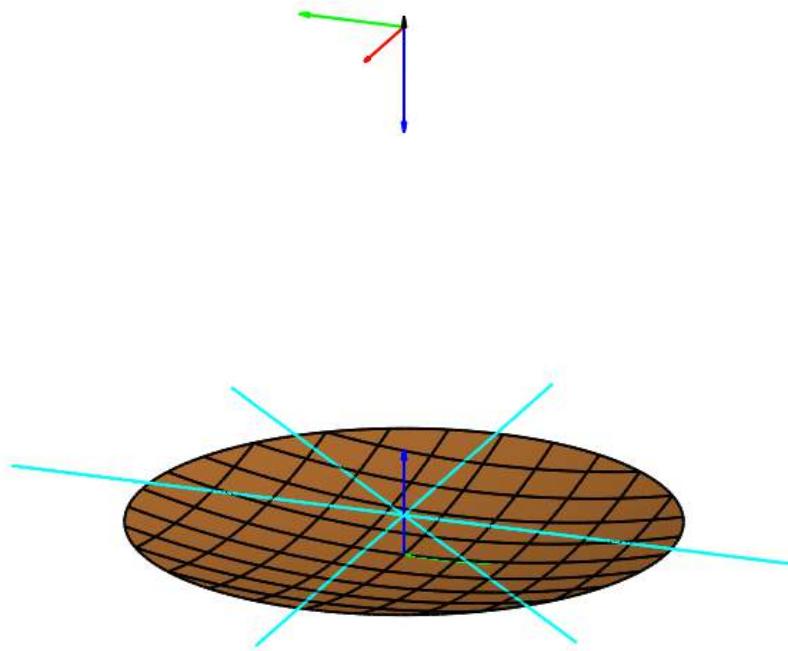


Figure 6-91 Parabolic reflector and near field cuts.

The advantage of using the plane wave expansion in the close near field is that it is much faster than PO. This may be important for e.g. calculating the field incident on struts. When the integration of the PO currents shall converge at field points close to some of the current elements, a dense integration grid is required, and the integration will be time consuming. By means of the plane wave expansion only a sparse integration grid is needed and the integration is much faster. That is because the plane wave expansion is based on the far field within the specified BEAM CONE ANGLE and the integration grid of the PO-currents can be relatively coarse when the field shall be determined in these far field directions.

The results are shown in Figure 6-92. It is seen that the agreement between the two methods is very good inside the aperture, but that the details of the diffraction lobes outside the aperture differ. Further, a small dip at $\theta = 0^\circ$ due to a caustic from the edge diffracted rays is correctly described by PO but missing in the plane wave expansion. The reason for these small inaccuracies

is that they are caused by diffracted rays with an angle from boresight much larger than the BEAM CONE ANGLE, here selected to 5° . The fraction of power inside this cone is 0.9990 compared to the total radiated power of the reflector.

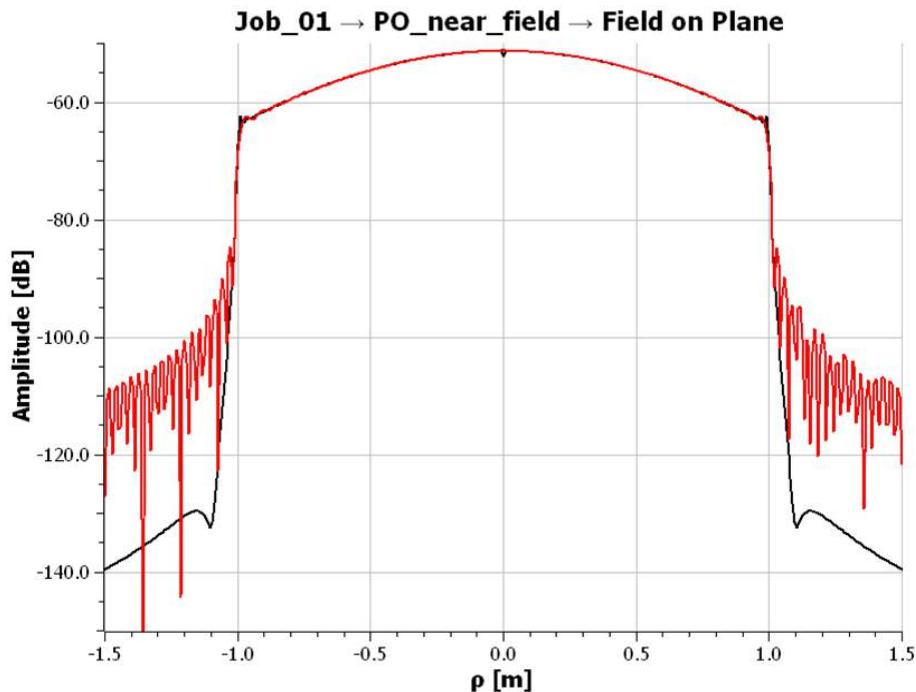


Figure 6-92 Comparison of PO (black) and plane wave expansion (red) in the near field of a parabolic reflector. The pattern cut shown is $\phi = 0^\circ$.

The commands needed to perform these calculation are shown in Figure 6-93. The first two commands perform the standard PO calculation of the near field whereas the last three commands include the plane wave expansion of the field from the reflector.

Command Type	Objects	Arguments
1 Get Currents	Source : Feed_1 Target : PO_Calc_1	auto_convergence_of_po : on convergence_on_scatterer: convergence_on_output_grid : PO_near_field convergence_on_expansion_grid : field_accuracy : -100.0 max_bisections : 5 integration_grid_limit : on obsolete_conv_on_po_grid :
2 Get Field	Source : PO_Calc_1 Target : PO_near_field	auto_convergence_of_po : on convergence_on_scatterer: convergence_on_output_grid :
3 Get Currents	Source : Feed_1 Target : PO_Calc_1	convergence_on_expansion_grid : Pw_exp field_accuracy : -80.0 max_bisections : 5 integration_grid_limit : on obsolete_conv_on_po_grid :
4 Get Plane Wave Expansion	Source : PO_Calc_1 Target : Pw_exp	auto_convergence : on convergence_on_scatterer: convergence_on_near_field_grid : Pw_near_field field_accuracy : -80.0 max_bisections : 5 automatic_convergence_limit : on
5 Get Field	Source : Pw_exp Target : Pw_near_field	

Figure 6-93 Commands for calculating the near field using standard PO (commands 1-2) and including plane wave expansion (commands 3-5).

6.8 Scalable Dual Reflector Using Real Variables

The purpose of this tutorial is to demonstrate the use of REAL VARIABLES. To this end, the dual reflector antenna as shown in Figure 6-94 is considered. The design is carried out such that the antenna is completely scalable with frequency. A GRASP project for this case can be opened from the box TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The project files are located in the TESTCASES/VARIABLES_IN_DUAL_REFL_DESIGN subdirectory in the installation directory.

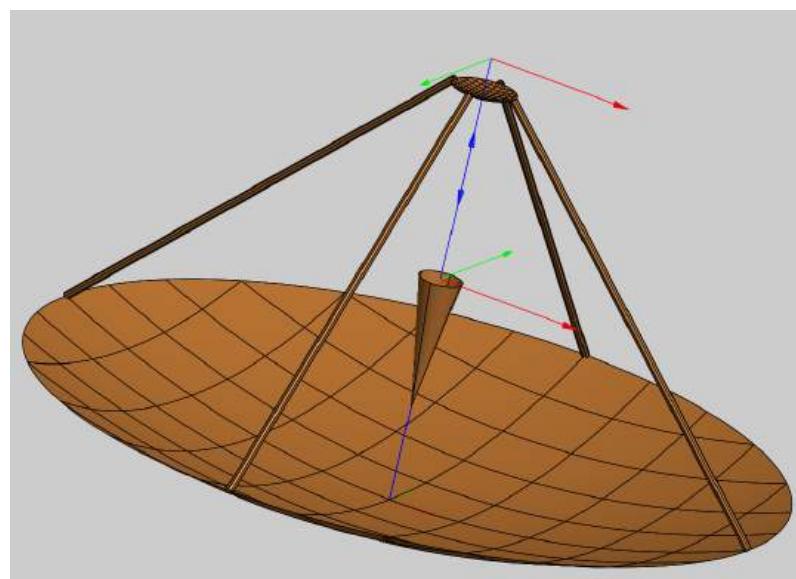


Figure 6-94 Dual reflector antenna with struts.

6.8.1 Creation of Dual Reflector Geometry Using Variables

The first step is to design the dual reflector antenna using the GRASP wizard. The selected parameters in the wizard are shown in Figure 6-95. It is seen that the frequency is selected to 4 GHz, the diameter of the main reflector aperture diameter to 10 m with a f/D of 0.6, yielding a focal length of 6 m. The automatically generated list of objects are displayed in Figure 6-96. For more information on these objects, the reader is referred to Section 6.2 where a dual reflector with blockage is considered. For simplicity, the sub reflector blockage is not treated in this tutorial test case.

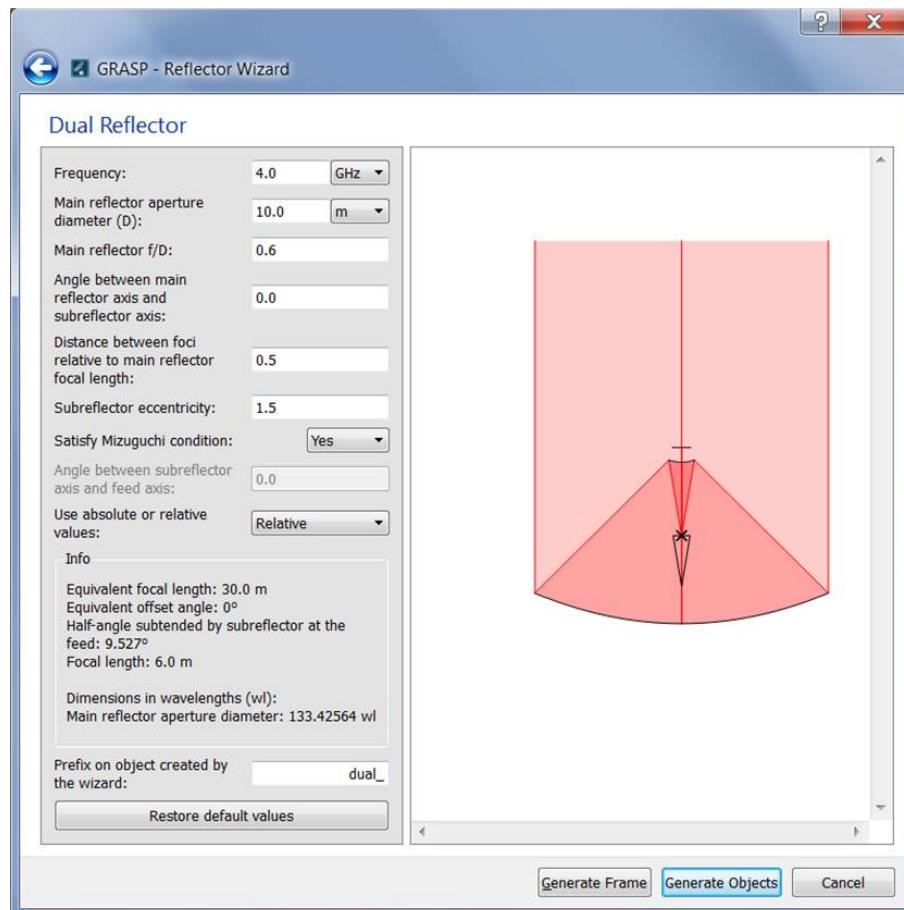


Figure 6-95 Parameters in the wizard and the list of objects.

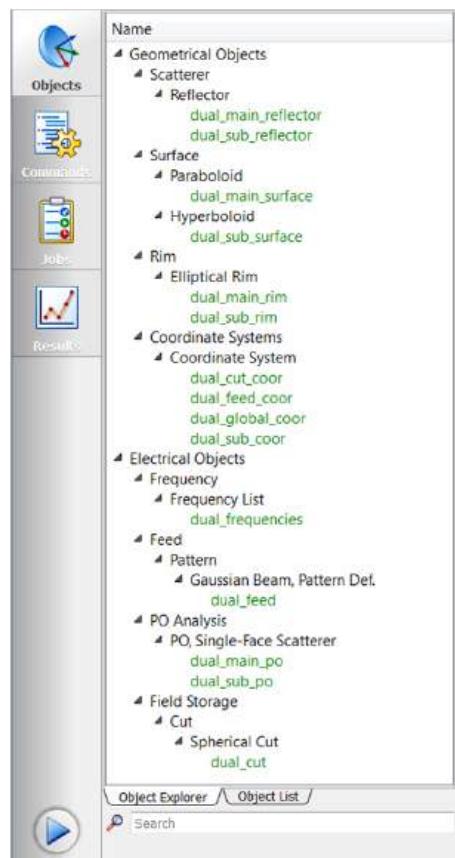


Figure 6-96 Parameters in the wizard and the list of objects.

Prior to creating the objects for generating the circular struts, we define first two real variables. These are created in the menu CREATE, under the VARIABLES class. The first REAL VARIABLE is given the name FREQUENCY and assigned the value 4, see Figure 6-97. This value can for instance be used in the FREQUENCY LIST object to define the frequency, in this case 4 GHz, see Figure 6-98. It is worthwhile to note that variable FREQUENCY should be used assuming the unit 'GHz'. The second REAL VARIABLE is given the name LAMBDA which represents the free-space wavelength. This variable is assigned using the expression '300/FREQUENCY' giving the value 75. Since the wavelength at 4 GHz is 75 mm, LAMBDA should be used with the unit 'mm' in mind.

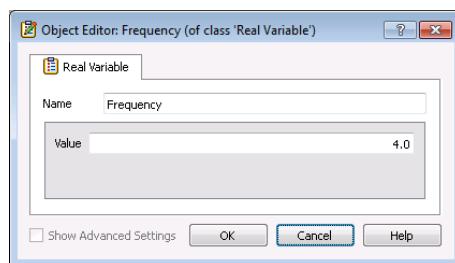


Figure 6-97 Defining the the FREQUENCY variable.

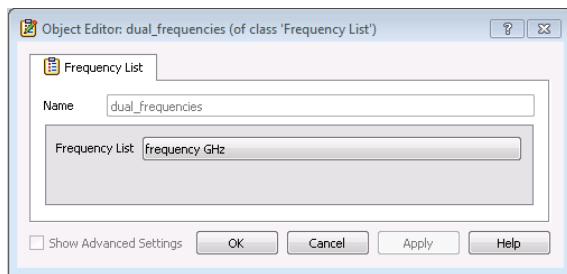


Figure 6-98 The use of the FREQUENCY variable.

Having introduced the first variables, we now proceed to creating the struts. To do so, we introduce a range of new variables:

- MAINR, radius of the main reflector
- MAINZ, z -value of the main reflector rim
- SUBR, radius of the subreflector
- SUBZ: z -value of the subreflector rim
- FOCALL, main reflector focal length
- FOCALD, subreflector foci distance
- VERTEXD, subreflector vertex distance

The values of these variables are found in the objects created by the Wizard. In addition to these, some helping variables, SUBA, SUBB, and SUBC, are defined. The assigned values and expressions are summarised in Figure 6-99².

Variables Editor	
focalb	$\sqrt{\text{subc}^*\text{subc}-\text{suba}^*\text{suba}}$ = 1.118034
focald	3.0
focall	6.0
frequency	4.0
lambda	$300.0/\text{frequency}$ = 75.0
mainr	5.0
mainz	$\text{mainr}^{2.0}/4.0/\text{focall}$ = 1.0416667
suba	$\text{vertexd}/2.0$ = 1.0
subb	$\sqrt{\text{subc}^*\text{subc}-\text{suba}^*\text{suba}}$ = 1.118034
subc	$\text{focald}/2.0$ = 1.5
subr	0.431655
subz	$\text{focal}-\text{subc}+\text{suba}^*\sqrt{(\text{subb}^*\text{subb}+\text{subr}^*\text{subr})/\text{subb}}$ = 5.5719424
vertexd	2.0

Figure 6-99 List of variables.

²The expressions can be found in Chapter 2.2.1 in "K. Pontoppidan, Ed., GRASP, Technical Description, TICRA, 2008."

These variables are used to calculate the ρ - and z -coordinates of the two rim curves, DUAL_MAIN_RIM and DUAL_SUB_RIM, from which the struts are defined. The struts are created through the CREATE menu under the class GEOMETRICAL OBJECTS→SCATTERER→STRUTS→CIRCULAR STRUTS. The RADIUS attribute is given the arbitrary value 50 mm and the values in the END POINTS attribute are shown in Figure 6-100.

	Point1x	Point1y	Point1z	Point2x	Point2y	Point2z
1	subr*lambda/75.0 m	0.0 m	subz*lambda/75.0 m	mainr*lambda/75.0 m	0.0 m	mainz*lambda/75.0 m
2	0.0 m	subr*lambda/75.0 m	subz*lambda/75.0 m	0.0 m	mainr*lambda/75.0 m	mainz*lambda/75.0 m
3	-subr*lambda/75.0 m	0.0 m	subz*lambda/75.0 m	-mainr*lambda/75.0 m	0.0 m	mainz*lambda/75.0 m
4	0.0 m	-subr*lambda/75.0 m	subz*lambda/75.0 m	0.0 m	-mainr*lambda/75.0 m	mainz*lambda/75.0 m

Figure 6-100 Definition of circular struts.

Note that all the values are multiplied by the factor "LAMBDA/75". This factor is actually multiplied in all the objects in the project which contain attributes of length dimension. For instance in the object DUAL_MAIN_SURFACE, the attribute FOCAL LENGTH has been modified from FOCALL to FOCALL*LAMBDA/75. Recall that LAMBDA has the value 75 at 4 GHz, hence LAMBDA/75 equals 1 and does thus not affect the geometry of the dual reflector and the circular struts. However, a change in the FREQUENCY variable should lead to an appropriate scaling of the entire model. In this way, the dual reflector system is completely scalable with frequency just by changing the value of FREQUENCY.

6.8.2 Analysis of the Dual Reflector

After having created the objects, we now turn our attention to analysing the dual reflector. In Figure 6-101, the commands for analysing the dual reflector is listed:

1. the PO currents on the subreflector is calculated using the field from the feed as incident field
2. the PO currents from the subreflector is used to illuminate the main reflector from which the PO currents on the main reflector is determined
3. a plane wave expansion of the radiation of the main reflector due to the subreflector is determined
4. and this plane wave expansion is used to calculate the incident field on the circular struts from which the PO currents on the struts are calculated
5. the field from the main reflector when illuminated by the subreflector is stored in DUAL_CUT

6. the field from the feed, the subreflector, and the circular structs is added in DUAL_CUT

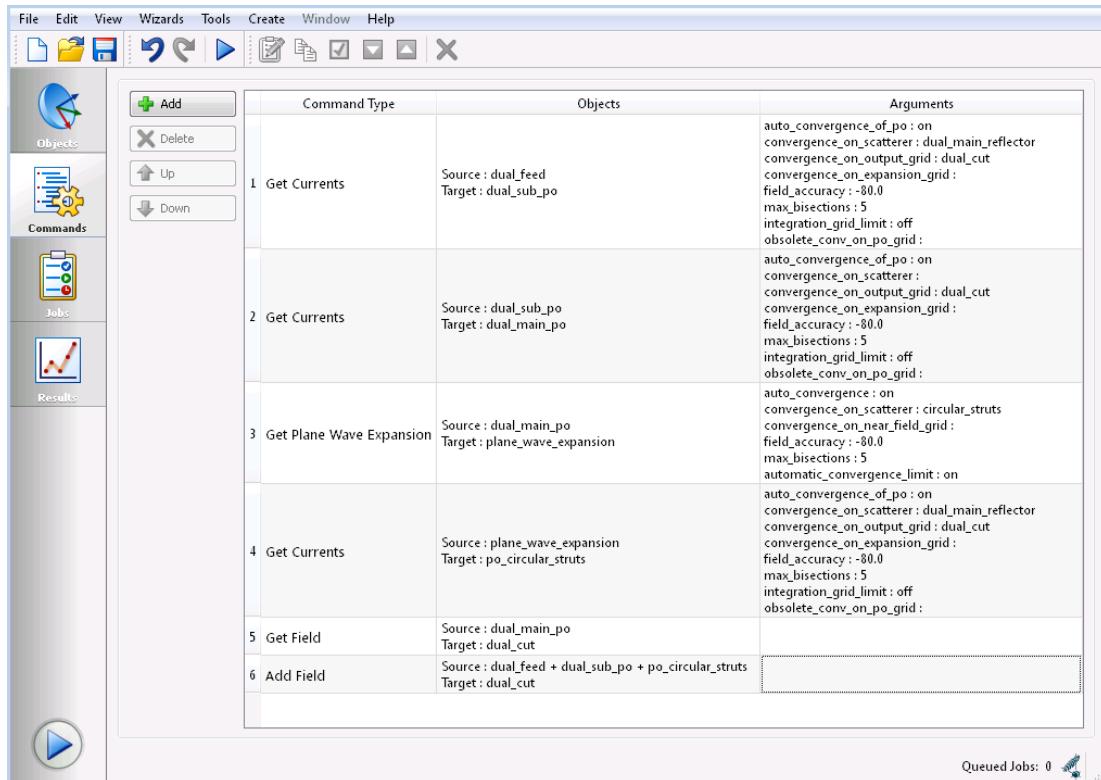


Figure 6-101 List of commands for analysing the dual reflector.

In principle, some of the scattered field from the circular struts will illuminate the main reflector and these contributions should be accounted for. However, for the present case, the effects are negligible and is therefore not included in the analysis.

The plane wave expansion in command 3 is in general not necessary and command 4 can be carried out directly using SOURCE:DUAL_MAIN_PO. However, the use of a plane wave expansion has several advantages when calculating the near-field, e.g. the field incident on struts, why it is included here. For more information on the plane wave expansion, see Section 6.7.

The radiation of the dual reflector at 4 GHz and 8 GHz is shown in Figure 6-102. It is seen that the radiation patterns are identical, verifying that the design has been correctly scaled at 8 GHz.

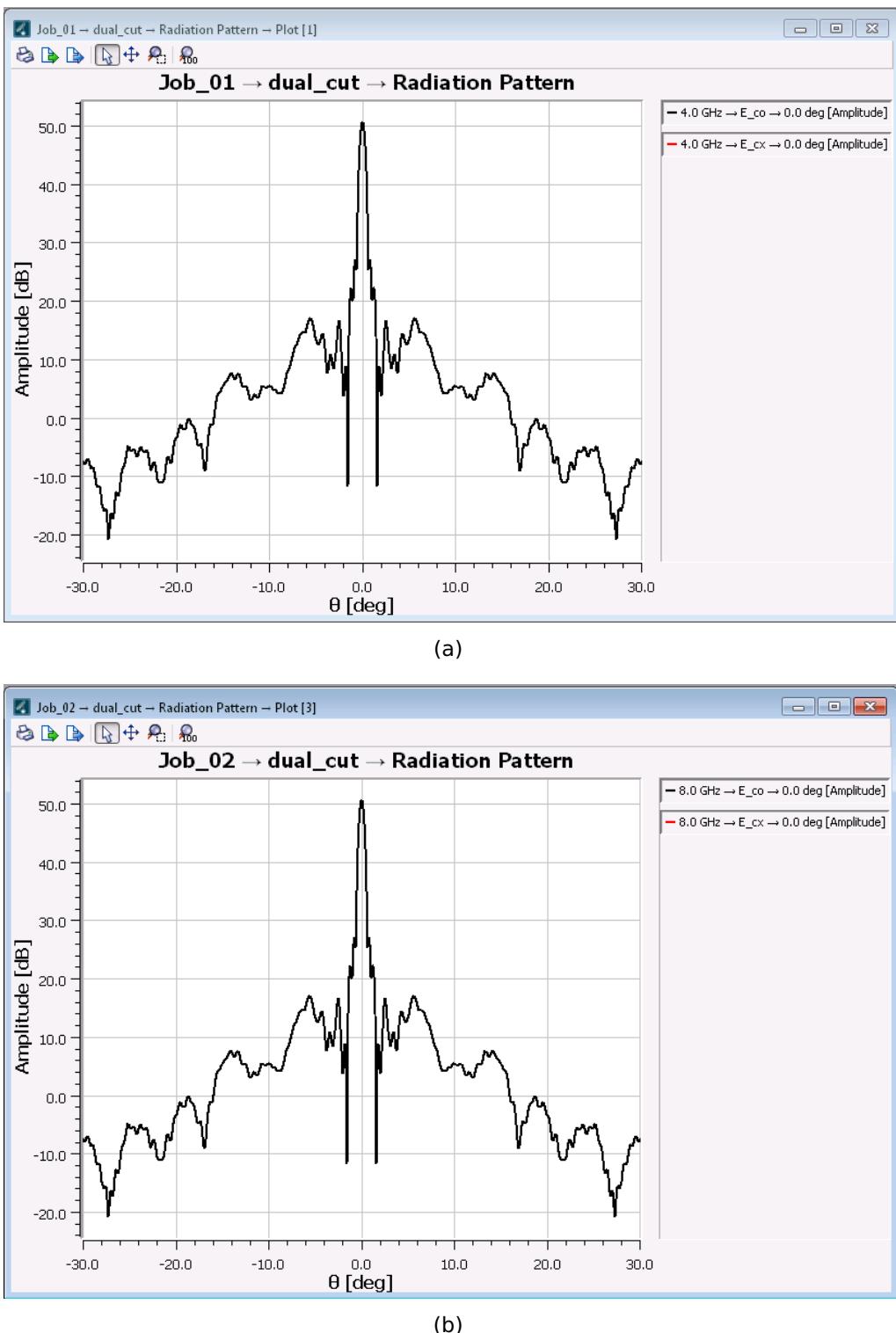


Figure 6-102 Radiation of dual reflector at (a) 4 GHz and (b) 8 GHz.

6.9 Offset Reflector in Radome

The purpose of this tutorial is to demonstrate the analysis of radomes in GRASP. In this example, an offset reflector within an extended hemispherical radome will be analyzed. A GRASP project can be opened from the box TUTORIAL EXAMPLES in the GRASP - NEW PROJECT window. The project files are located in the TESTCASES/OFFSET_REFLECTOR_IN_RADOME subdirectory in the installation directory.

6.9.1 Creation of the Geometry

First, we design an offset single reflector antenna by means of the GRASP wizard. The antenna operates at 12 GHz and has the geometrical characteristics as shown in Figure 6-103.

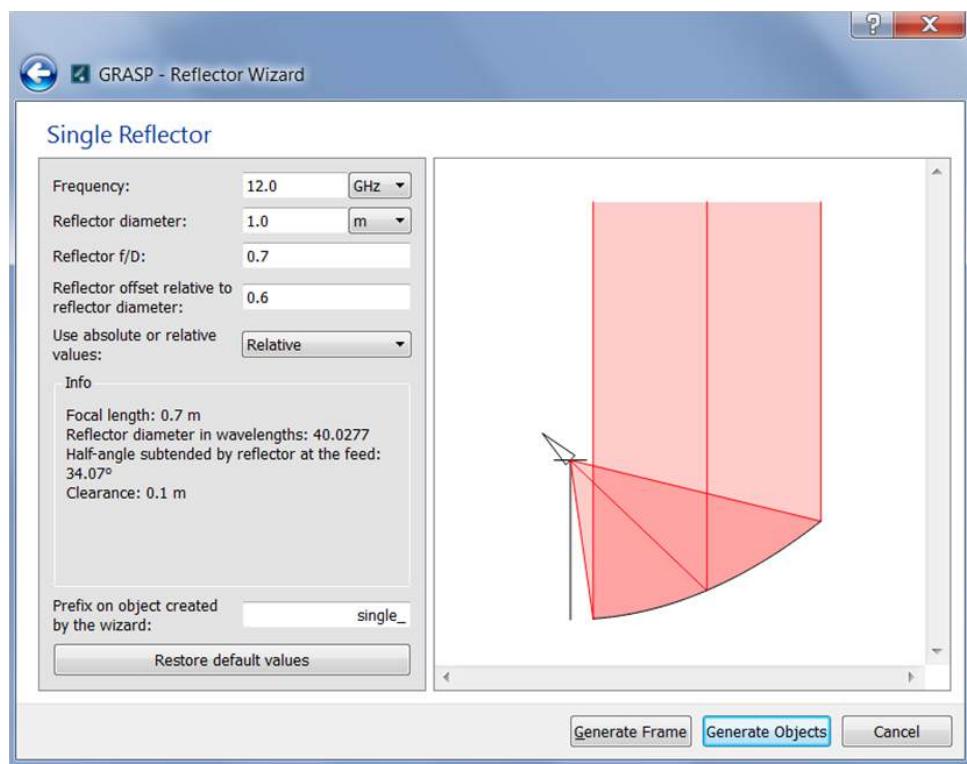


Figure 6-103 Wizard for single offset reflector.

A drawing of the geometry as generated by the wizard is shown in Figure 6-104

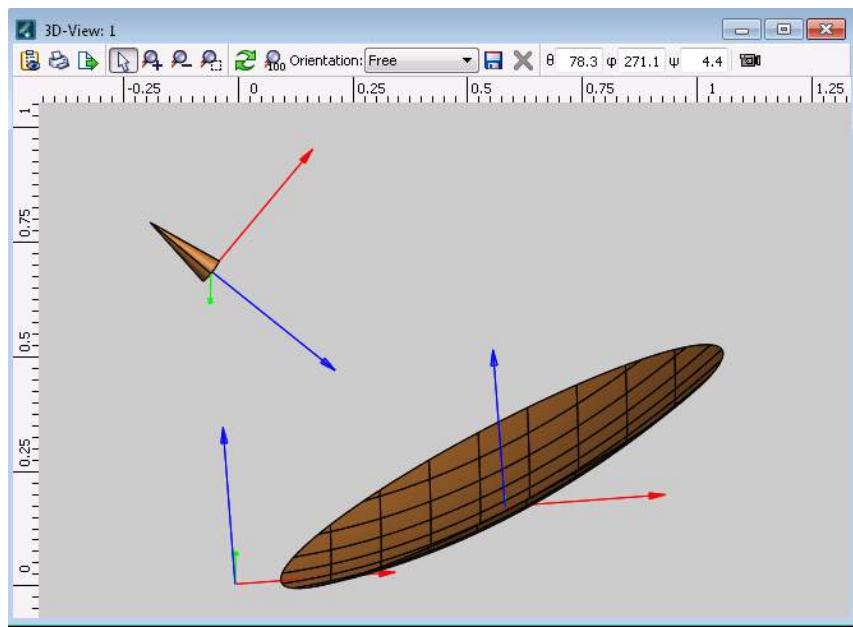


Figure 6-104 Geometry of offset reflector.

Having created the objects associated to the reflector we proceed with the definition of the extended hemispherical radome. In GRASP, there is no dedicated objects for the creation of a radome, thus the radome is built up by using reflectors with special electrical properties. To this end, we need to define several SURFACE and RIM objects.

First, we create an ELLIPTICAL RIM by selecting:
 CREATE→GEOMETRICAL OBJECTS→RIM→ELLIPTICAL RIM
 with the attributes as shown in Figure 6-105.

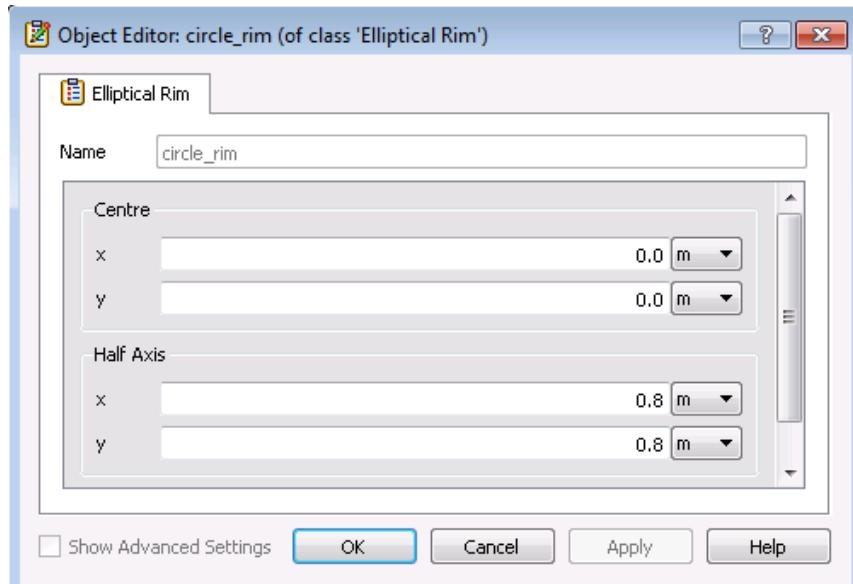


Figure 6-105 Definition of ELLIPTICAL RIM.

Second, a SPHERICAL SURFACE is created under
 CREATE→GEOMETRICAL OBJECTS→OTHER QUADRRICS→SPHERICAL SURFACE

with the attributes shown in Figure 6-106.

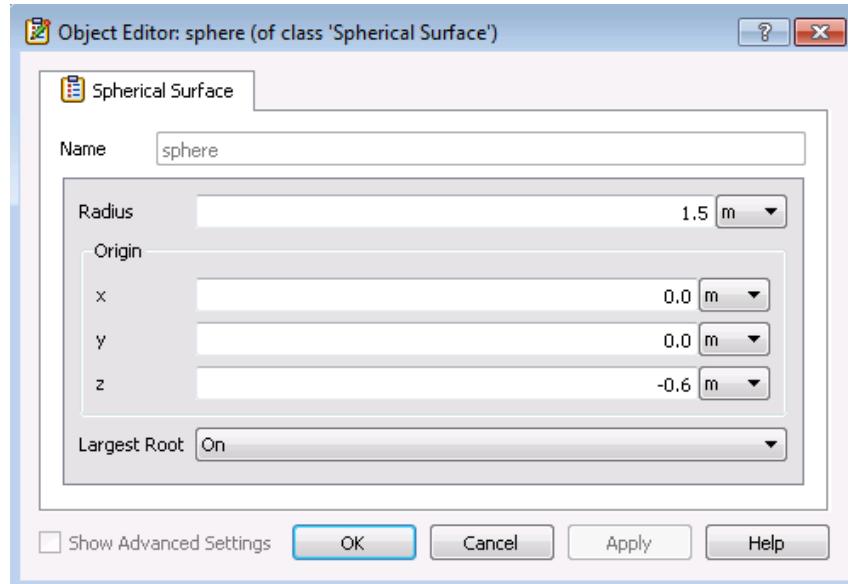


Figure 6-106 Definition of SPHERICAL SURFACE.

Finally, we create a REFLECTOR object with the name SPHERICAL_RADOME where the reference to the COORDINATE SYSTEM, SURFACE, and RIM objects are shown in Figure 6-107.

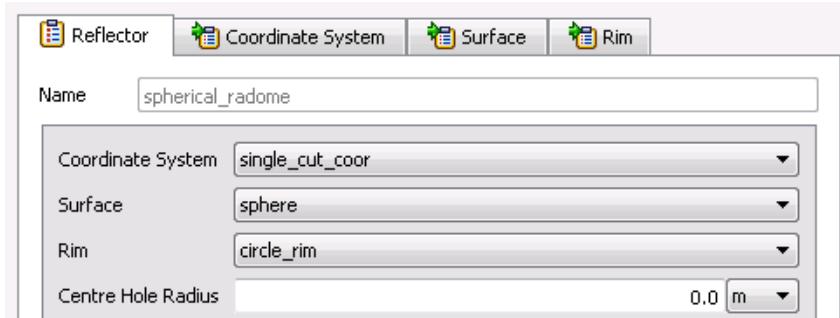


Figure 6-107 Referenced objects for the SPHERICAL_RADOME object.

This then creates the top part of the radome as displayed in Figure 6-108.

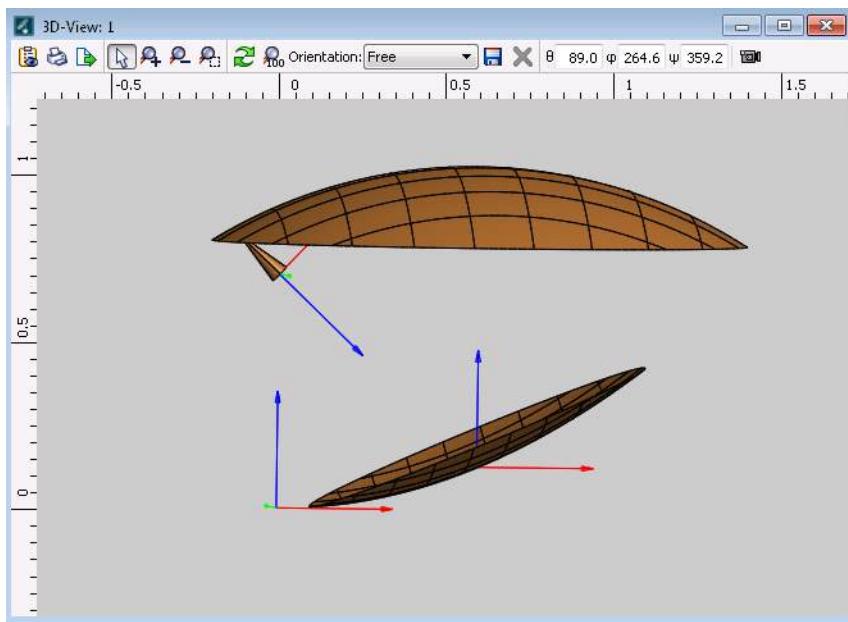


Figure 6-108 Geometry of the offset reflector including the top surface of the spherical radome.

In a similar way, with some simple geometrical considerations, we create a CIRCULAR CYLINDER SURFACE and a RECTANGULAR RIM followed by a number of COORDINATE SYSTEMS and REFLECTORS to create the remaining parts of the radome. The list of GEOMETRICAL OBJECTS is listed in Figure 6-109 and the final geometry created by the combinations of the objects is shown in Figure 6-110. Note that four cylinder surfaces are defined to create the cylinder part of the radome. A single cylinder surface is not possible since the REFLECTOR object requires that the surface is expressed uniquely by $z = f(x, y)$ where $f(x, y)$ is a function describing the surface. Consequently, at least two cylinder surfaces are necessary to create the cylinder, one for each half cylinder. However, this is an inappropriate choice seen from a numerical point of view since the surface has a rapid z -variation at the edges of the half cylinder. Thus, for numerical stability, four surfaces are used.

Name
▪ Geometrical Objects
▪ Scatterer
▪ Reflector
cylinder_radome1
cylinder_radome2
cylinder_radome3
cylinder_radome4
single_reflector
spherical_radome
▪ Surface
▪ Paraboloid
single_surface
▪ Other Quadrics
▪ Spherical Surface
sphere
▪ Circular Cylinder Surface
cylinder
▪ Rim
▪ Elliptical Rim
circle_rim
single_rim
▪ Rectangular Rim
rectangular_rim
▪ Coordinate Systems
▪ Coordinate System
single_cut_coor
single_feed_coor
single_global_coor
▪ Coordinate System, GRASP Angles
cylinder_coor1
cylinder_coor2
cylinder_coor3
cylinder_coor4

Figure 6-109 List of GEOMETRICAL OBJECTS.

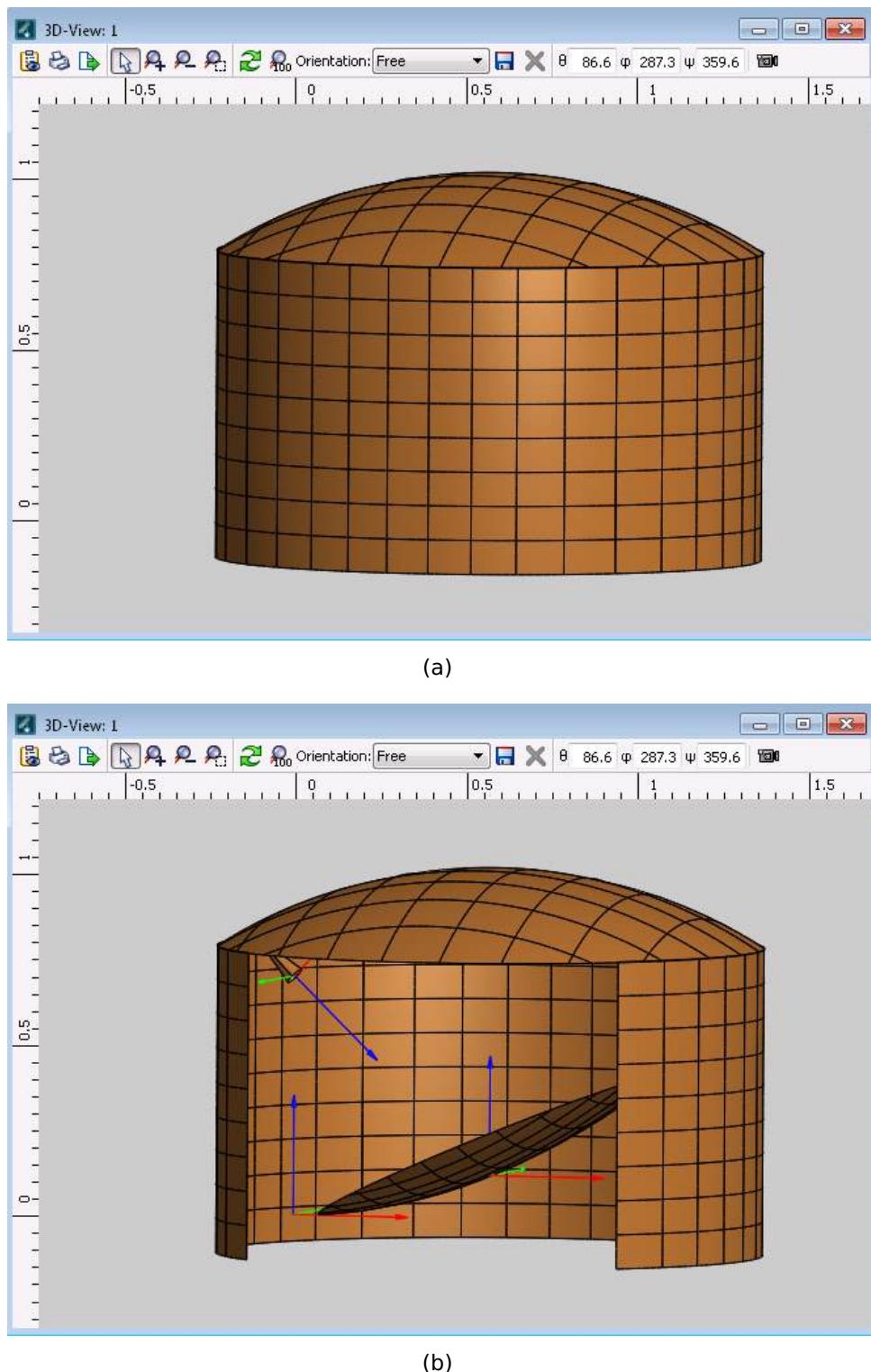


Figure 6-110 Geometry of the extended hemispherical radome. In (a) the full radome is shown, whereas in (b) a part of the radome is hidden.

As the objects are defined now, the radome surface consists of a solid PEC and will fully reflect the radiation from the offset reflector. Therefore, the next step is to define a dielectric surface for the radome such that it appears transparent for the reflector. In many practical applications, a dielectric sandwich structure is preferred, and such a structure will also be used in this example.

It is known, that for an incident plane wave with normal incidence, a dielectric layer will appear transparent if the thickness of the layer is selected as $\lambda/2$ with λ being the wavelength inside the dielectric. For mechanical stability, a sandwich structure is preferred, thus we create two DIELECTRIC LAYER objects under the CREATE→ELECTRICAL OBJECTS→ELECTRICAL PROPERTIES menu. The attributes for the objects are shown in Figure 6-111. We choose a dielectric constant of 4 for both dielectrics, resulting in a wavelength in the dielectric of approximately 1.25 cm at 12 GHz, thus the thicknesses of the dielectric layers become 0.625 cm.

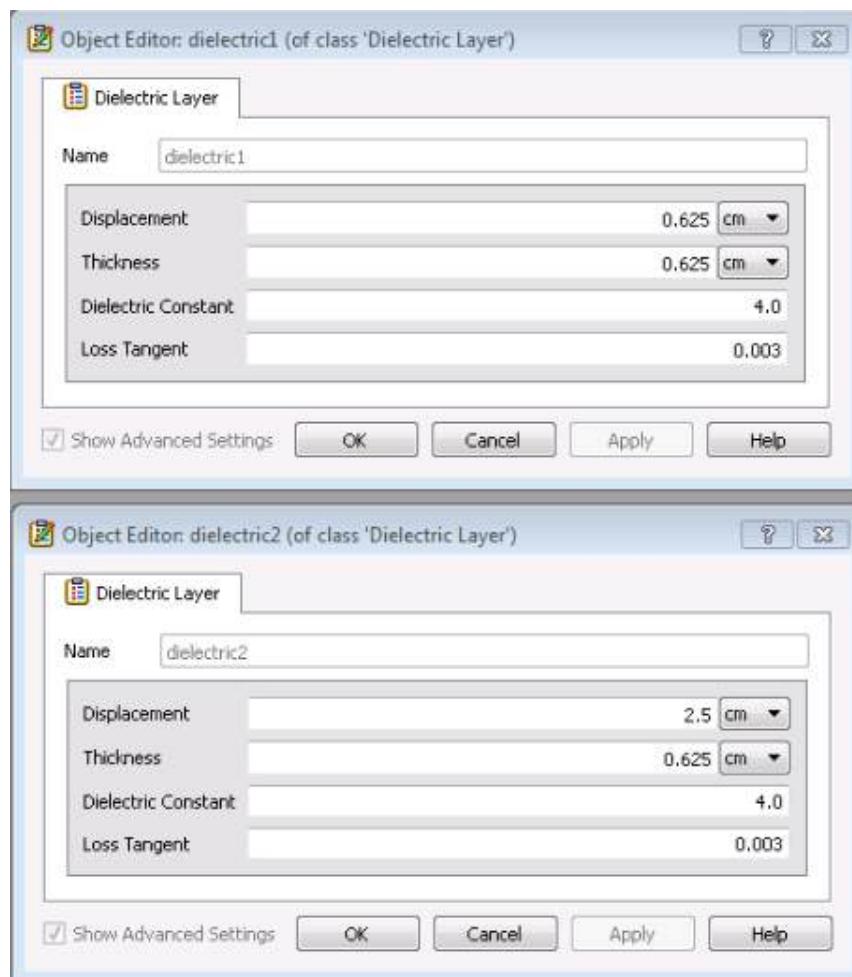


Figure 6-111 OBJECT EDITOR of the DIELECTRIC LAYER objects.

To apply the surface properties in the radome, we open the OBJECT EDITOR for all the REFLECTOR objects related to the radome and add DIELECTRIC1 and DIELECTRIC2 into the attribute ELECTRICAL PROPERTIES, see Figure 6-112.

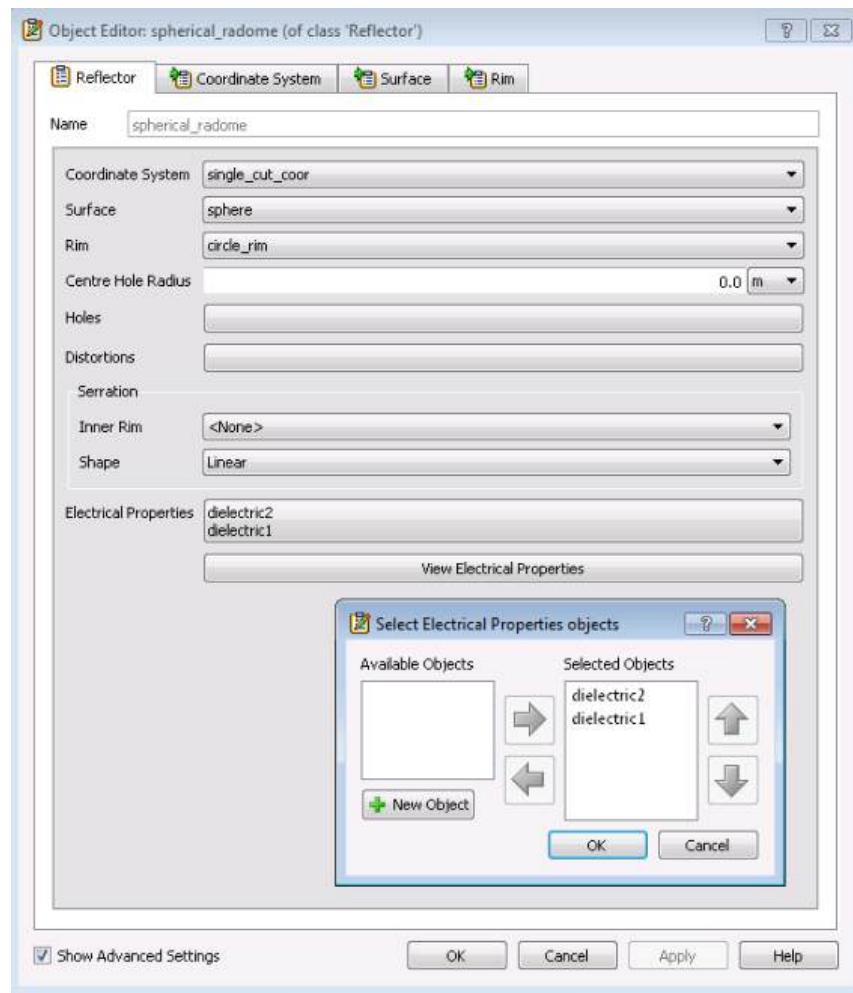


Figure 6-112 Insertion of the dielectric layers in the SPHERICAL_RADOME object.

By clicking on the VIEW ELECTRICAL PROPERTIES button, the sequence of the surface properties is shown, see Figure 6-113. GRASP automatically sorts the sequence of the electrical properties based on the value specified in the DISPLACEMENT attribute.

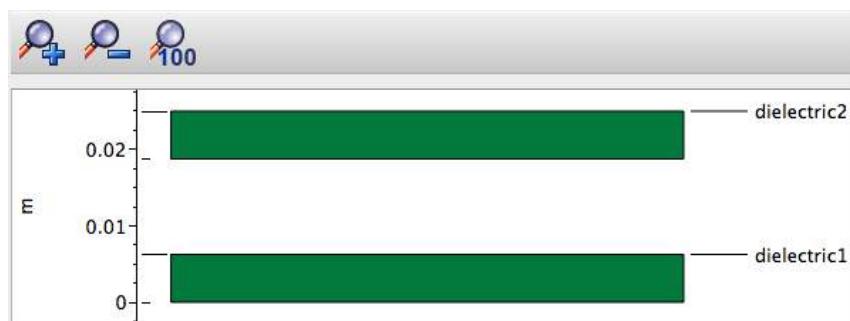


Figure 6-113 VIEW ELECTRICAL PROPERTIES of the radome objects.

The DISPLACEMENT attribute, which is specified in the direction of the normal vector of the scatterer, defines where the reference surface for the dielectric layer is located relative to the surface of the scatterer. A DISPLACEMENT of

0 means that the top surface of the dielectric layer is at the surface of the scatterer. Note that the DISPLACEMENT of DIELECTRIC2 is larger than that of DIELECTRIC1, indicating that DIELECTRIC2 lies above DIELECTRIC1. This also means the the DISPLACEMENT of DIELECTRIC2 essentially is the sum of the thickness of the first layer and the distance between the two layers. This is clearly illustrated in Figure 6-113.

Now, with the addition of the electrical properties, the radome surface consists of two dielectric layers separated with air and should appear transparent for the reflector at the specified frequency.

6.9.2 Analysis of the Offset Reflector and the Radome

After having created the geometrical objects, we now turn our attention to analyzing the reflector system with the radome.

Prior to the calculations, an object of the SCATTERER CLUSTER class is created (CREATE→GEOMETRICAL OBJECTS→SCATTERER). In the SCATTERERS attribute, all the REFLECTOR objects related to the radome should be inserted. In this way, the analysis of the entire radome can be carried out using only one command instead a command for each REFLECTOR object. For the analysis of the entire radome, an electrical object of the PO, MULTI-FACE SCATTERER class is created (CREATE→ELECTRICAL OBJECTS→PO ANALYSIS) with the attributes as shown in Figure 6-114. Note that PTD analysis cannot be applied on scatterers with electrical properties, which is the reason why the METHOD attribute is selected to Po.

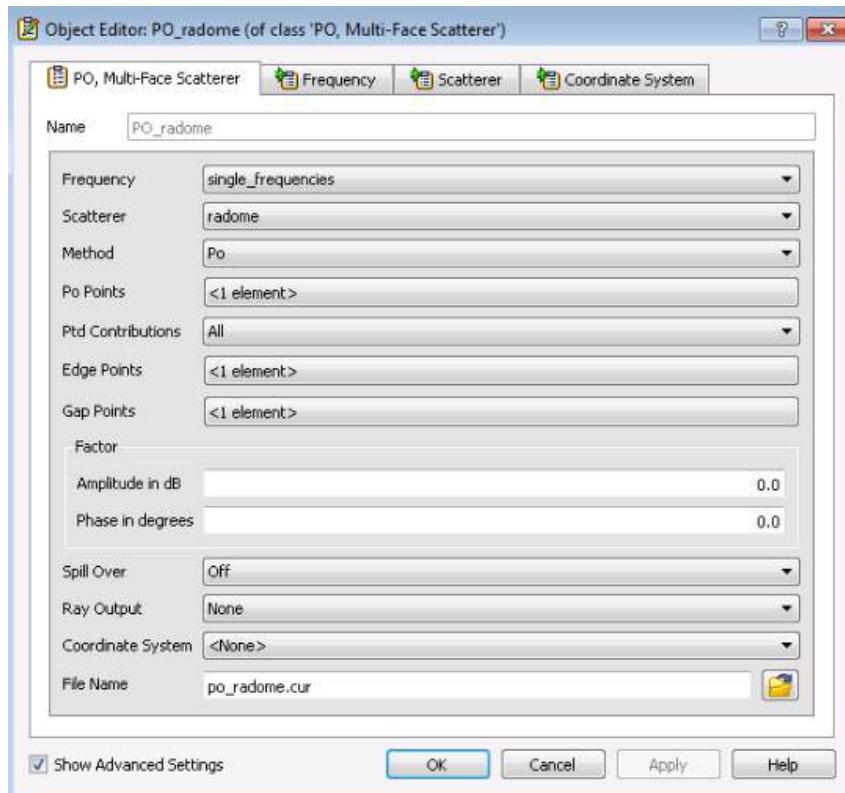


Figure 6-114 OBJECT EDITOR of the PO, MULTI-FACE SCATTERER object.

For storage of the radiation pattern, the GRASP wizard has generated a FIELD STORAGE object of the SPHERICAL CUT class, named SINGLE_CUT. We would like to compare the radiation with and without the presence of the radome, hence we duplicate the object (right-click on SINGLE_CUT and select DUPLICATE SELECTED...) and name the new object SINGLE_CUT_NORADOME. This object will be used to store the radiation of the reflector without the presence of the radome.

The COMMANDS to carry out the analysis and the calculation of the radiation patterns are listed in Figure 6-115:

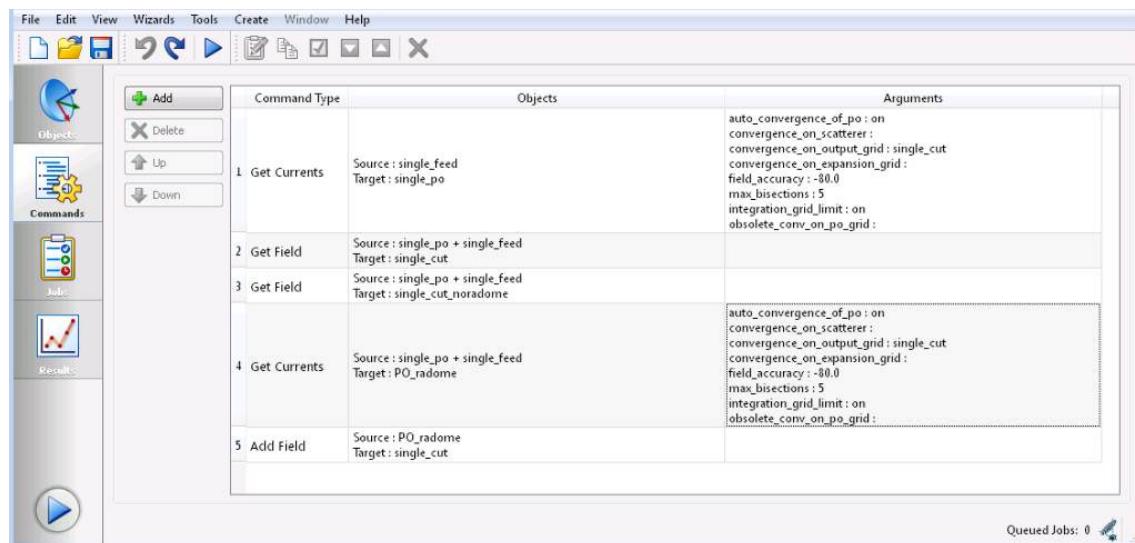


Figure 6-115 COMMAND list for the analysis.

1. The PO currents on the reflector is calculated using the field from the feed as incident field.
2. The radiation pattern of the PO currents and the feed is calculated and stored in the SINGLE_CUT object.
3. The radiation pattern of the PO currents and the feed is calculated and stored in the SINGLE_CUT_NORADOME object.
4. The PO currents and the feed are used as source to calculate the PO currents on the radome.
5. The radiation pattern of the PO currents on the radome is calculated and added to the SINGLE_CUT object.

It is seen in command no. 4 that the PO currents from the reflector are used as a source to calculate another set of currents on a scatterer with electrical properties. When electrical properties are present on a scatterer, the PO currents on the scatterer are determined by means of reflection and transmission coefficients which requires the direction of propagation of the incident field (see GRASP Technical Description for more information). For these cases, the attribute RAY OUTPUT in the PO object, see Figure 6-116, can affect analysis

accuracy. This attribute defines how the field of the currents shall be calculated if the field is used as input to a subsequent GTD or PTD analysis, or in a subsequent PO analysis on a scatterer with electrical properties. When the field from the currents is input to a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified, the direction of propagation of the field is not required and the attribute RAY OUTPUT can be selected to NONE. This is not the case when electrical properties are specified. The analysis accuracy can be enhanced if the RAY OUTPUT attribute is selected to ALL since the direction of propagation is calculated in a different way as compared to if the attribute is NONE. However, the increased accuracy is at the expense of higher computational time, especially for electrically large scatterers. For more information on the RAY OUTPUT attribute, the user is referred to *PO, Single-Face Scatterer* in this Reference Manual.

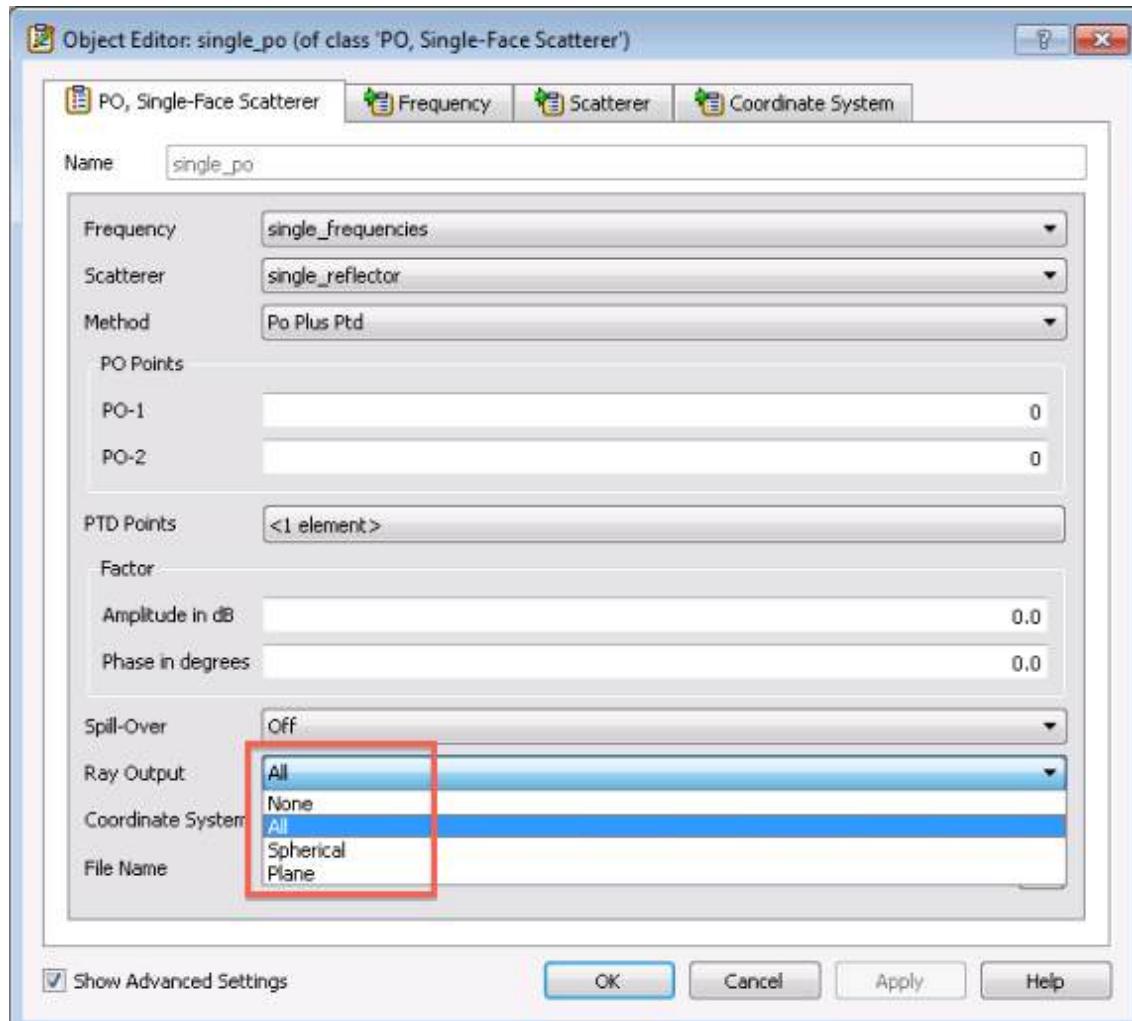


Figure 6-116 OBJECT EDITOR of the PO, SINGLE-FACE SCATTERER object. The red box highlights the selection of the RAY OUTPUT attribute.

In Figure 6-117, the radiation pattern of the reflector with and without the presence of the radome is shown. It is seen that the radiation pattern with radome is slightly distorted as compared to the one without. The peak directivity with and without radome is 40.83 dBi and 40.92 dBi, respectively. The

radome is designed at the specified frequency, which explains why the loss is less than 0.1 dB. At other frequencies, the performance of the radome will decrease and higher losses will be observed. In practice, realistic radomes need to be optimized to operate at multiple frequencies.

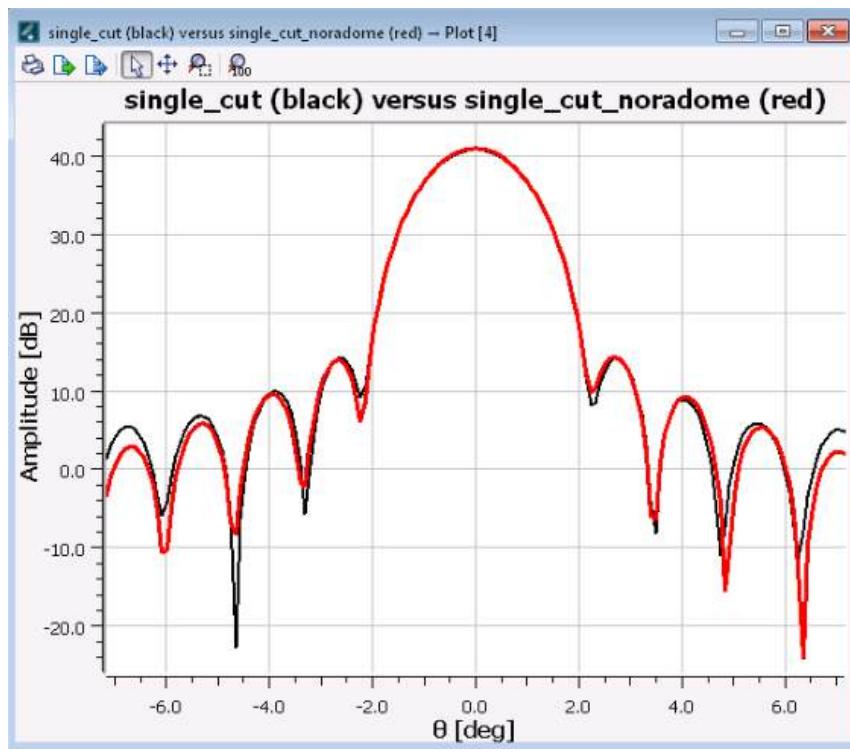
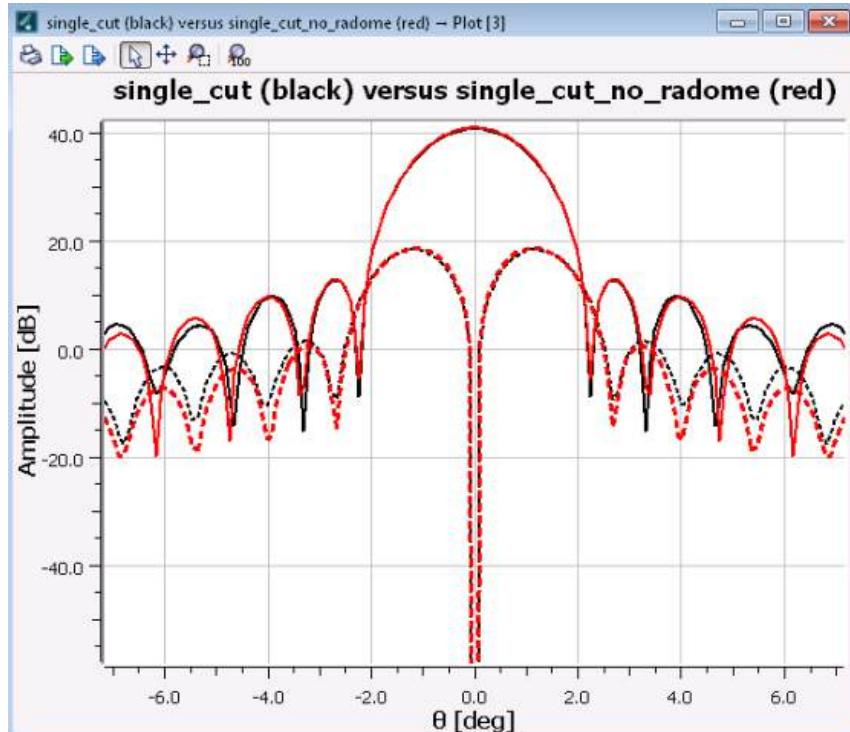
(a) $\phi = 0^\circ$ (b) $\phi = 90^\circ$

Figure 6-117 Radiation of the offset reflector with (black) and without (red) the presence of the radome. Co-polar components are shown with solid lines whereas cross-polar components are drawn with dashed lines.

For GRASP users with the **MoM add-on**, the radome can also be analyzed using the Body of Revolution Method of Moments (*BoR-MoM*) which is more

accurate compared to PO. However, depending on the electrical size and complexity of the radome, the computation time can be significantly higher as compared to PO.

6.10 Dual Reflector with Panels and Subreflector Support Struts

The purpose of this test case is to demonstrate the analysis of a ring-focus dual reflector system where the main reflector consists of a number of panels arranged in a polar grid. In addition the effects of the subreflector support struts will be determined. The project files are located in the TEST-CASES/DUAL_REFL_WITH_PANELS_AND_STRUTS subdirectory in the installation directory.

6.10.1 Antenna geometry

The main parameters selected for the ring-focus antenna are

- type: Gregorian ring-focus
- frequency, 1.5 GHz
- main reflector diameter, 20 m
- subreflector diameter, 2 m
- main reflector focal length, 6 m
- feed opening angle, 25°

These numbers are inserted in the ring-focus wizard and the objects are generated. The resulting antenna system is shown in Figure 6-118. The polarisation for the feed is selected to be right-hand circular (RHC). This means that the final radiated beam, after reflection in the subreflector and the main reflector, will also be RHC.

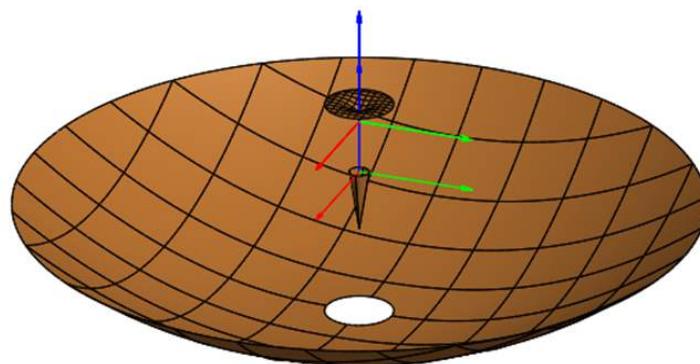


Figure 6-118 The ring-focus antenna system as generated by the wizard.

The wizard generates a hole in the main reflector with the same diameter as that of the subreflector. In the present case we wish to close this hole by a flat plate. This is done by setting the centre hole radius to zero in the `MAIN_REFLECTOR` object and to add an additional point to the `MAIN_SURFACE` table as shown in Figure 6-119.

	<code>rho</code>	<code>z</code>
1	0.0	-6.0
2	1.0	-6.0
3	1.0502793	-5.9998947
4	1.1005587	-5.9995787
5	1.150838	-5.999052
6	1.2011173	-5.9983147
7	1.2513966	-5.9973667

Figure 6-119 Main reflector surface table with additional centre point inserted.

The main reflector defined so far is a single piece 20 m in diameter. For the practical realisation of this large reflector we will define it by a number of smaller panels arranged in a polar grid. We select one circular central panel 2 m in diameter and 4 rings of panels with the same radial dimension and containing 6, 12, 18, and 24 panels from the inner to the outer ring. The gap width is selected to 2 cm in both radial and circumferential direction.

The panels are defined from `CREATE > GEOMETRICAL OBJECTS > SCATTERER > REFLECTOR WITH PANELS > PANELS IN POLAR GRID` in the `PANEL_REFLECTOR` object as shown in Figure 6-120 and the resulting antenna geometry is shown in Figure 6-121. The main reflector is now modeled in two ways, as a single piece in `MAIN_REFLECTOR` and as panels in `PANEL_REFLECTOR`. When looking at the geometry in the view plot it is recommended to hide one of them; otherwise the picture can get blurred.

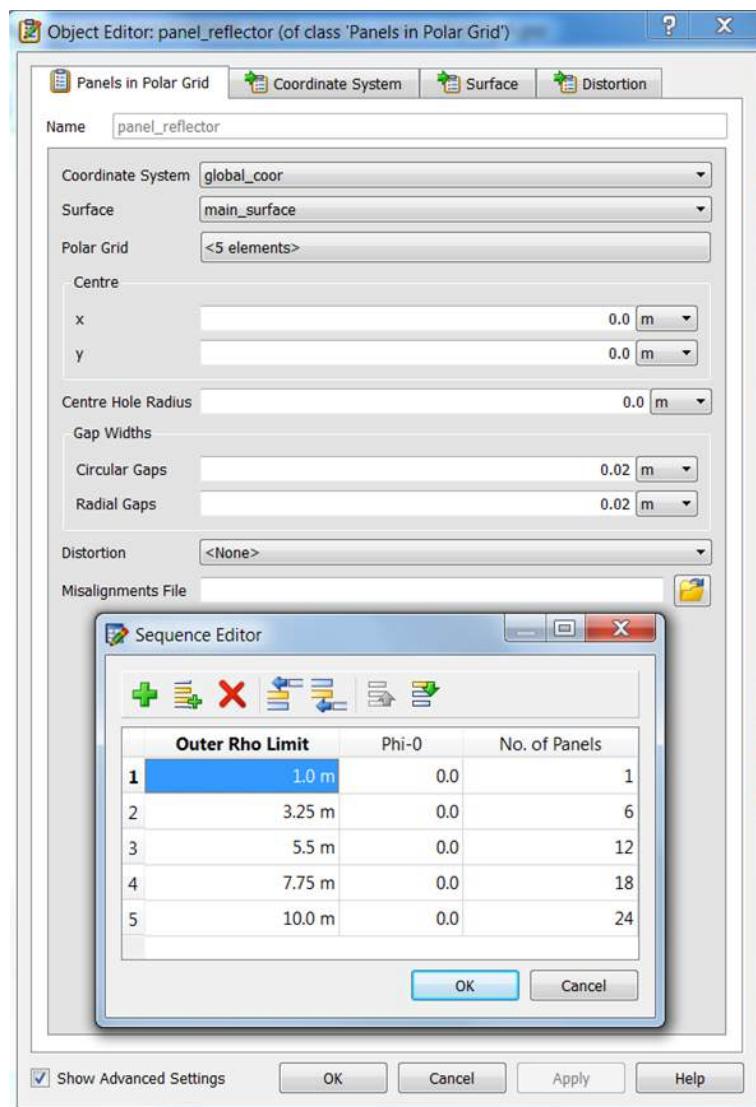


Figure 6-120 Object editor for the definition of panels in a polar grid.

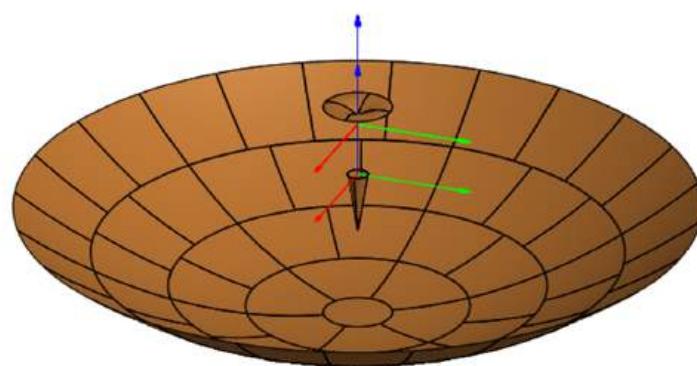


Figure 6-121 Antenna system with main reflector consisting of 61 panels.

The gap width is too small to be seen in Figure 6-121. Figure 6-122 shows an enlarged part of the reflector where the gaps are clearly visible.

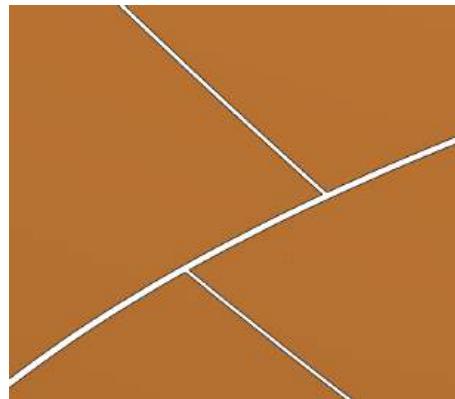


Figure 6-122 Enlarged part of the paneled main reflector.

6.10.2 Calculated radiation patterns with and without panels

The contributors to the radiated field are the feed, the subreflector and the main reflector. The calculations are specified by the commands shown in Figure 6-123. Commands 1, 2 and 3 calculate the PO+PTD currents on the subreflector, the solid main reflector and the paneled main reflector, all with a convergence accuracy of -80 dB. The PO objects for the subreflector and the solid main reflector are generated by the wizard. The PO object, PO_PANELS, for the paneled reflector is created by selecting CREATE> ELECTRICAL OBJECTS> PO ANALYSIS> PO, PANELS IN POLAR GRID. In this object also the effect of the gaps between the panels is included by selecting PO+PTD

Note, that in all the PO objects the SPILL-OVER has been set to ON in order to be able to track the incident power on the different reflectors. In addition, a FILE NAME is defined such that the reflector currents can be used for later calculations. The total fields are determined by commands 4 and 5 for the solid and the paneled main reflector, respectively. The 5 commands in Figure 6-123 are executed as Job_01.

Command Type	Objects	Arguments
1 Get Currents	Source : feed Target : sub_po	<8 arguments>
2 Get Currents	Source : sub_po Target : main_po	<8 arguments>
3 Get Currents	Source : sub_po Target : po_panels	<8 arguments>
4 Get Field	Source : feed + main_po + sub_po Target : total_solid_cuts	
5 Get Field	Source : feed + sub_po + po_panels Target : total_panel_cuts	

Figure 6-123 Commands used to calculate the field from the feed, the subreflector and the main reflector.

The LOG OUTPUT available in the JOBS window shows that the relative power hitting the reflectors is

- Subreflector: 0.9305

- Solid main reflector: 0.9012
- Paneled main reflector: 0.8856

The results are depicted in Figure 6-124. These results are obtained from the RESULTS window by copy and paste the different curves into the same plot.

Right-clicking a curve in the RESULTS window activates a small menu where one can select QUICK VIEW DATA FOR CURVE(S)... which presents a table of the data for the curve. In this way one can find that the peak directivity for the solid and paneled reflector is 48.89 and 48.80 dBi, respectively. This difference is due to the gaps in the paneled reflector such that the power not being reflected passes through the gaps and this is why the total radiation in the direction behind the reflector, from 110° to 180° , is considerably higher for the paneled reflector.

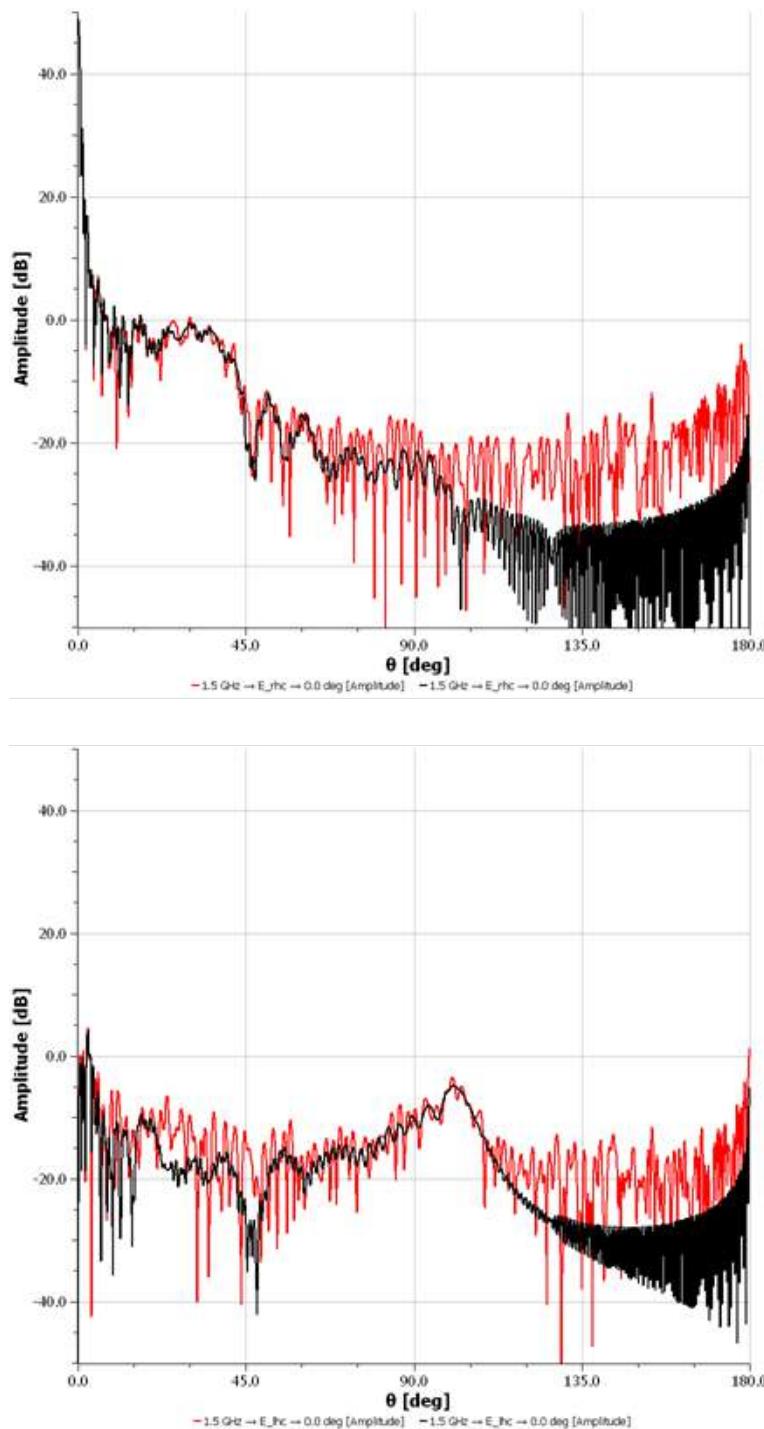


Figure 6-124 Calculated radiation patterns for the solid and the paneled main reflector. Top: RHC components, bottom: LHC components. Black curves: solid reflector, red curves: paneled reflector

6.10.3 Influence of subreflector support struts

The subreflector is supported by four circular struts as shown in Figure 6-125. The ends of the struts are located near the rims of the two reflectors. The struts make an angle of 21° with the plane containing the main reflector rim. The diameter of the struts is selected to 16 cm. The struts are defined in the CIRCULAR_STRUTS_4 object which is created by selecting

CREATE> GEOMETRICAL OBJECTS> SCATTERER> STRUTS> CIRCULAR STRUTS and the coordinates can be seen in Figure 6-126. In order to study the influence of the struts it is easier to start with the effect of only one strut, the CIRCULAR_STRUTS_1 object.

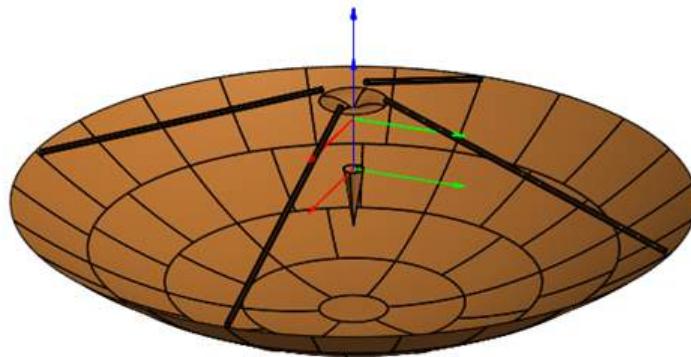


Figure 6-125 Illustration of the four struts carrying the subreflector .

	Point1 x	Point1 y	Point1 z	Point2 x	Point2 y	Point2 z
1	-9.8 m	0.0 m	-2.625 m	-1.0 m	0.0 m	0.753 m
2	9.8 m	0.0 m	-2.625 m	1.0 m	0.0 m	0.753 m
3	0.0 m	9.8 m	-2.625 m	0.0 m	1.0 m	0.753 m
4	0.0 m	-9.8 m	-2.625 m	0.0 m	-1.0 m	0.753 m

Figure 6-126 The coordinates defining the four struts in Figure 6-125.

The primary influence of the strut is to block the field coming from the main reflector and propagating in the antenna boresight direction. The length of the strut in the aperture is 8.8 m and the blocked area is therefore 1.4 m^2 . The total area of the reflector is $100\pi = 314 \text{ m}^2$. We can therefore expect that the strut field in the forward direction will be around $20*\log(1.4/314) = -47 \text{ dB}$ compared to the main beam peak. This indicates that there is no point in calculating the strut currents with a convergence accuracy of -80 dB since this will be much lower than the accuracy of the nominal fields calculated above. This observation is not so important in the present case where the reflectors are not very large in terms of the wavelength. However, if this is the case a reduced accuracy for the strut currents shall be used to save computer time.

The strut currents are calculated in the STRUT_ANALYSIS_1 object which

is created by selecting CREATE > ELECTRICAL OBJECTS > STRUT ANALYSIS > STRUT ANALYSIS, CIRCULAR CROSS SECTION. The induced strut currents could be calculated by a GET CURRENTS command using the paneled reflector currents, PO_PANELS, as the source. However, the struts are located in the very near field from the reflector and in this region a plane wave expansion is much more efficient. We therefore define a plane wave expansion object, PWE, which is created by selecting CREATE> ELECTRICAL OBJECTS> OTHER SOURCES> PLANE WAVE EXPANSION. The plane wave expansion editor is shown in Figure 6-127. The BEAM CONE ANGLE is set to 20° since we have seen from the previous patterns that at this angle the main reflector field has decreased to very low levels. The BEAM POWER is set to ON in order to be able to check that sufficient amount of power is contained in the plane wave expansion. A FILE NAME is defined such that the expansion can be used for later calculations.

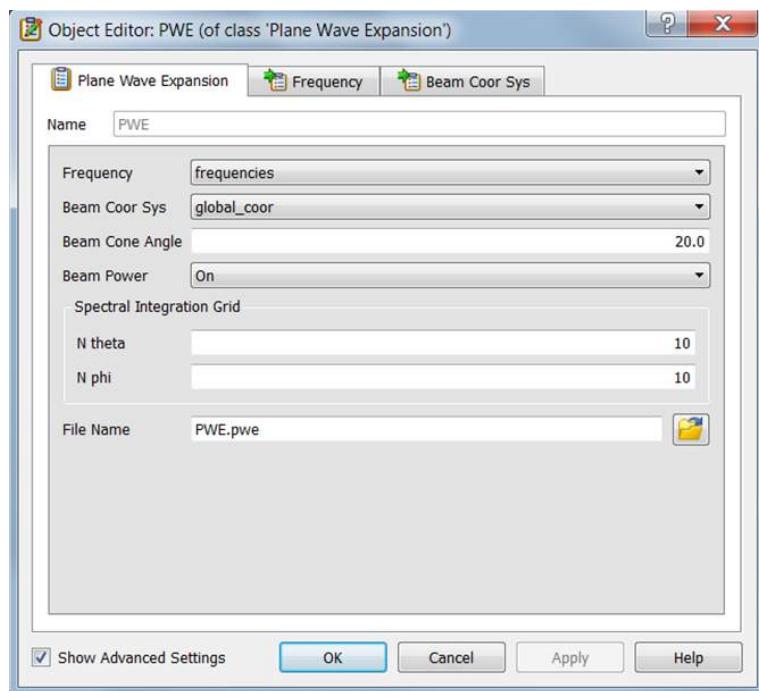


Figure 6-127 The plane wave expansion editor.

The rays from the feed are reflected in the subreflector and in the main reflector and they then propagate as parallel rays along the boresight axis of the antenna. Some of these rays will hit the strut as shown in Figure 6-128 and they will induce currents on the strut which will radiate, not only in the boresight direction but on the complete cone around the strut and with an opening angle of 69° because the struts are tilted 21° with respect to the aperture plane.

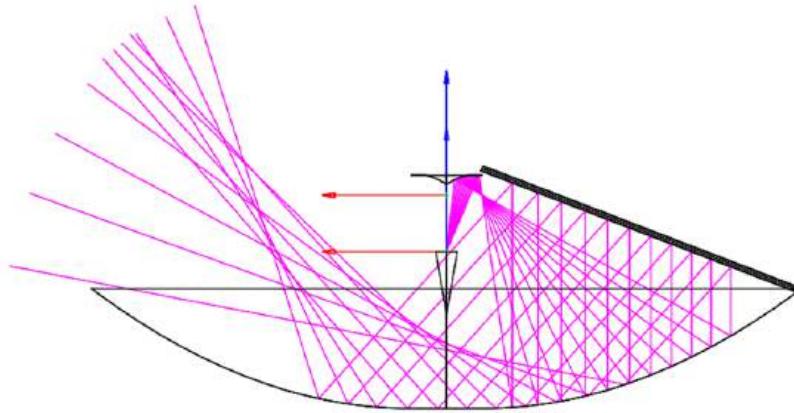


Figure 6-128 Rays propagating through the system in a plane containing a strut.

To calculate the effects of the strut the commands 1-5 are deactivated and the commands 6-11, shown in Figure 6-129, are executed as Job_02. Command 6 generates the plane wave expansion of the paneled main reflector field. In command 3 it was found that the relative power incident on the paneled main reflector is 0.8856. The output from command 6 shows that the power in the plane wave expansion is 0.8829 so the power radiated from the main reflector is sufficiently well contained in the plane wave expansion.

Command Type	Objects	Arguments
1 Get Currents	Source : feed Target : sub_po	<8 arguments>
2 Get Currents	Source : sub_po Target : main_po	<8 arguments>
3 Get Currents	Source : sub_po Target : po_panels	<8 arguments>
4 Get Field	Source : feed + main_po + sub_po Target : total_solid_cuts	
5 Get Field	Source : feed + sub_po + po_panels Target : total_panel_cuts	
6 Get Plane Wave Expansion	Source : po_panels Target : PWE	<6 arguments>
7 Get Currents	Source : PWE Target : strut_analysis_1	<8 arguments>
8 Get Field	Source : strut_analysis_1 Target : strut_cuts	
9 Get Currents	Source : strut_analysis_1 Target : strut_main_po	<8 arguments>
10 Get Field	Source : strut_main_po Target : strut_main_cuts	
11 Get Field	Source : strut_main_po + strut_analysis_1 Target : total_strut_main_cuts	

Figure 6-129 Commands used to calculate the influence from one single strut.

Command 7 determines the currents on the strut from the plane wave expansion and command 8 calculates the field in the plane containing the strut as shown at the top of Figure 6-130. The maximum at 0° is at a level of 2.6 dB or 46.3 below the main beam peak. This is very near the predicted 47 dB which was based on simple geometrical considerations.

It is clearly seen that the strut generates a sharp peak both at 0° and at 138° . The peak at 0° is primarily RHC because it blocks the RHC field from the

reflector whereas the peak at 138° is primarily LHC and it can be considered as a reflection from the strut. It will hit the main reflector where it will be reflected again as also illustrated in Figure 6-128.

Command 9 calculates the currents on the main reflector generated by the strut and command 10 calculates the associated field as shown in the middle of Figure 6-130. It is clearly seen that the lobe at 138° is replicated by the main reflector (but with opposite phase) such that it disappears when the two contributions are added by command 11 and shown at the bottom of Figure 6-130.

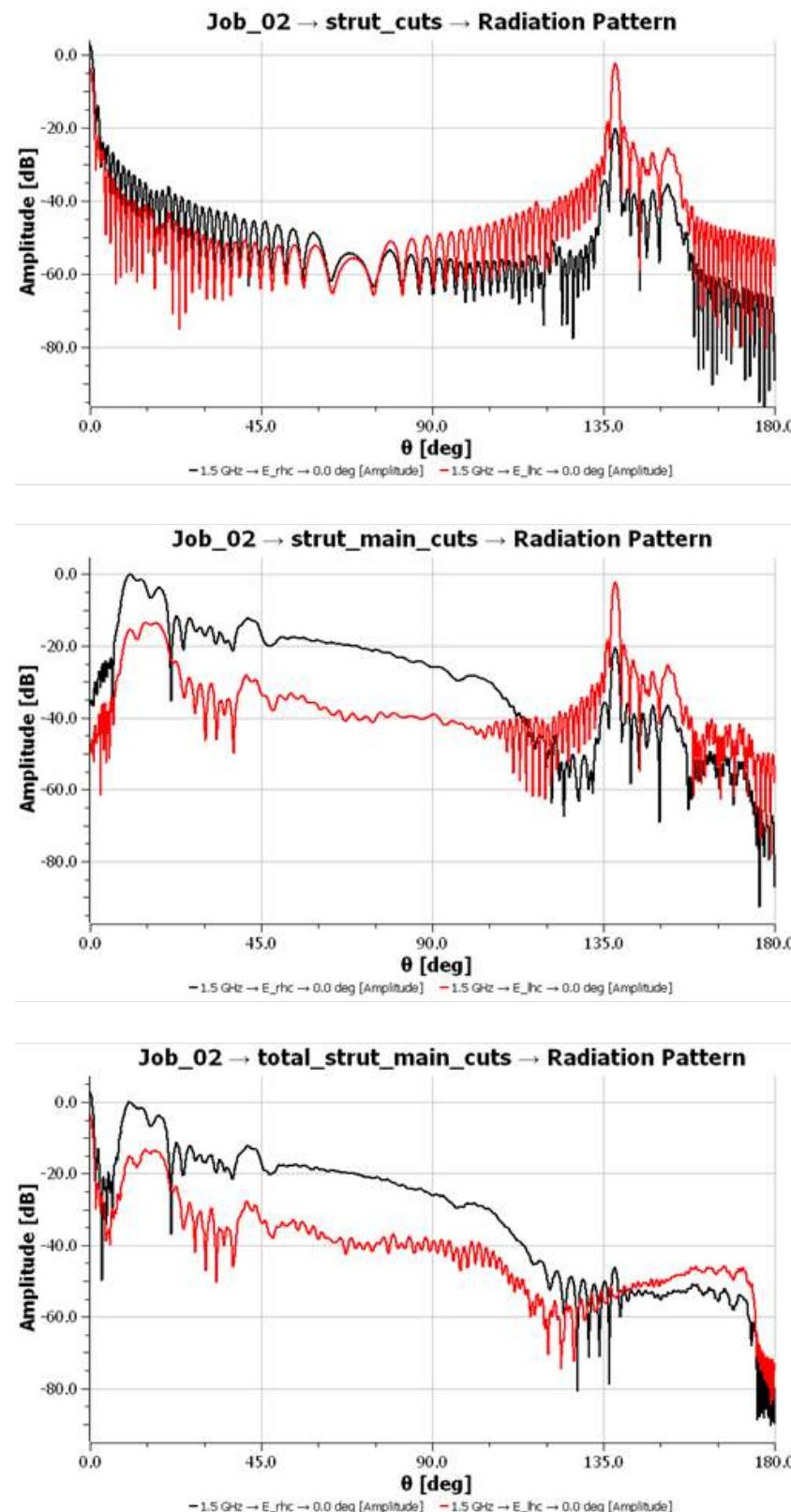


Figure 6-130 Strut effect in the plane containing the strut. Top: strut alone, Middle: main reflector alone, Bottom: strut and main reflector added.

Finally, the total radiation pattern for the antenna including the four struts shall be determined. This is done by running the commands 12-14, shown in Figure 6-131, after deactivating the commands 6-11. Command 12 calculates

the currents on the four struts when illuminated by the previously calculated plane wave expansion and command 13 calculates the currents induced on the main reflector. Command 14 adds the contributions from all the individual sources to obtain the total field shown in Figure 6-132.

Command Type	Objects	Arguments
1 Get Currents	Source : feed Target : sub_po	<8 arguments>
2 Get Currents	Source : sub_po Target : main_po	<8 arguments>
3 Get Currents	Source : sub_po Target : po_panels	<8 arguments>
4 Get Field	Source : feed + main_po + sub_po Target : total_main_cuts	
5 Get Field	Source : feed + sub_po + po_panels Target : total_panel_cuts	
6 Get Plane Wave Expansion	Source : po_panels Target : PWE	<6 arguments>
7 Get Currents	Source : PWE Target : strut_analysis_1	<8 arguments>
8 Get Field	Source : strut_analysis_1 Target : strut_cuts	
9 Get Currents	Source : strut_analysis_1 Target : strut_main_po	<8 arguments>
10 Get Field	Source : strut_main_po Target : strut_main_cuts	
11 Get Field	Source : strut_main_po + strut_analysis_1 Target : total_strut_main_cuts	
12 Get Currents	Source : PWE Target : strut_analysis_4	<8 arguments>
13 Get Currents	Source : strut_analysis_4 Target : strut_main_po	<8 arguments>
14 Get Field	Source : feed + sub_po + po_panels + strut_analysis_4 + strut_main_po Target : total_field_cuts	

Figure 6-131 Commands used to calculate the influence from all four struts.

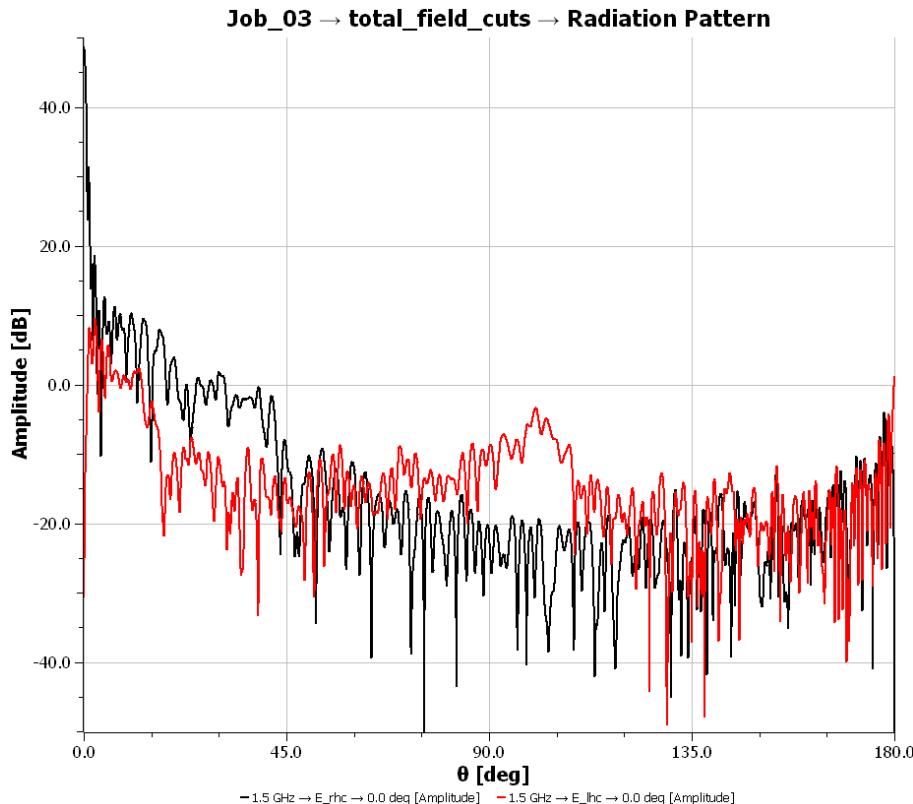


Figure 6-132 Radiation pattern in a plane of a strut when all contributions are included.

One will notice that the main beam peak is now 48.63 dBi compared to the

result without struts, 48.80 dBi, in Figure 6-124. The loss of 0.17 dB is due to the strut blockage. Another effect of the struts is significantly higher side lobes from boresight and up to 20° . The reason to this is that the pattern plane is parallel to the previously mentioned scattering cone from two of the struts.

7. Batch-Mode Operation

The GRASP analysis module can be executed from the command line by following the step-by-step instructions below. Special batch commands types can be found in the *Reference Section* in Chapter 9, under *Command Types*.

1. Prepare a GRASP project. It is recommended to use the GRASP GUI to prepare a valid project containing the correct antenna geometry and commands. Run an analysis once from inside the GUI to generate the required project files, including the `batch.gxp` file to be referenced later.
2. Locate the project files and create a copy of the relevant directory. For instance, if the GRASP project is named "MyProject", the relevant directory is:

`MyProject\working.`

Place the copy of this directory outside the `MyProject` directory.

Note: It is not recommended to perform batch-mode computations in a directory containing a project managed by the GRASP GUI. Doing so may result in a malfunction of the GUI if project files or result files are modified in an unexpected way. Instead, the batch-mode computation should be performed in a dedicated directory.

3. Open a command prompt and navigate to the dedicated batch mode directory created in the previous item.
4. Add the location of the GRASP executable to the PATH by executing the following command:

```
PATH = %PATH%;<installation_directory>\bin
```

where `<installation_directory>` is typically
`C:\Program Files\TICRA\GRASP-10.V` (V being the version number). This step can be avoided if the directory has been added to the system-wide PATH variable.

5. Run the analysis by executing the following command:
`grasp-analysis batch.gxp <name1>.out <name2>.log`
 where the gxp-file is an input file specifying the tor- and tci-files of the project, and the out- and log-files are output files. All files are ASCII-files and the names (`name1`, `name2`) may differ or be identical. As an example:

```
grasp-analysis.exe batch.gxp MyProject.out MyProject.log
```

An index file named `MyProject_<date>_<time>.xml` will be created during the batch run. This file contains information on the results of the batch run.

It is now possible in the Results Window of the GUI (cf. Section 4.4) to specify

`FILE > IMPORT > INDEX FILE`

and browse to the directory of the batch run after which the file `output_files.xml` can be imported. Thereby an entry with the name `IMPORTED_MYPROJECT_1` is created in the RESULTS EXPLORER and the files generated in the batch run may be explored in the same way as files generated from a job submitted from the GUI.

By default, the index file will be created during the batch run. The generation of the index file can be deactivated by adding `-no-index-file` or in short `-nif` at the end of the command:

```
grasp-analysis batch.gxp <name1>.out <name2>.log -nif
```

6. Subsequent batch runs can be performed by repeating step 5 above.

The contents of the tor-file can be modified using a standard text editor or by creating automated scripts, e.g., using Matlab. The tor-file contains an arbitrary number of objects - each object is of a certain class and contains a number of parameters. A detailed description of the classes and the associated parameters are included in the *Classes* chapter of the Reference Section.

The tci-file contains a number of commands executed sequentially by GRASP. A detailed description of the available commands is included in the *Command Types* chapter of the Reference Section. Of these commands the following two are useful for reading and writing tor-files:

1. `files read all <tor file>`. This command reads all objects of the specified tor-file. Additional `files read all` commands can be added by advanced users if it is desirable to split the project in multiple tor-files.
2. `files write all <tor file>`. This command writes all objects to the specified tor-file.

8. GRASP 10 for GRASP9 Users

This section is intended for those who are familiar with GRASP9 and wish to know the main differences between GRASP9 and GRASP 10. The general changes will be described in the following. Special attention will be given on how input and output files are stored.

Antenna design tasks often require many calculations with different settings of the parameters, to achieve the desired antenna performance. In GRASP9 this means that, in order not to overwrite previous results, it is necessary to either define a new file name or save the project in a new directory. This process is quite cumbersome and it is easy to make errors.

GRASP 10 makes use of the same objects and commands of GRASP9. However, its design flow is much more fluent. This means that when saving the project for the first time the user is asked to specify a project directory. The GRASP 10 project file will be stored here and at the same time a subdirectory named “Working” will be generated. This subdirectory will contain the .tor and .tci files as well as possible input files, such as tabulated surfaces and feeds. The structure is illustrated in Figure 8-1.

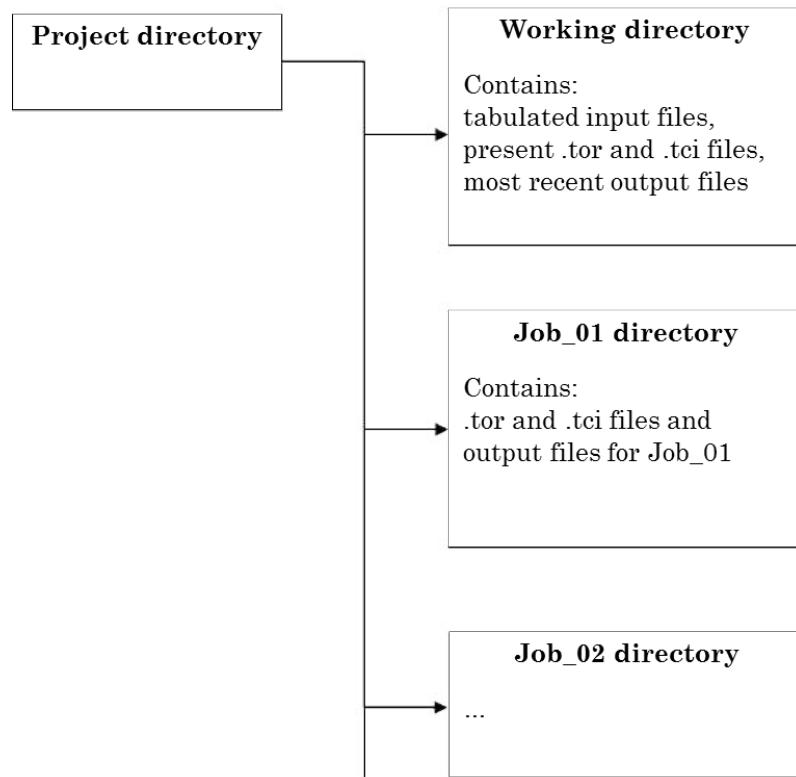


Figure 8-1 Directory structure of GRASP 10.

Before the first execution of the commands, the user is prompted to give a

job name, for example Job_01. All calculations are performed using the data in the Working directory and the output results are also stored here. When the job is terminated the output results are copied to the directory Job_01 together with the .tor and .tci that generated these results. The user may now wish to change some data and run again. These new results will then be stored in Job_02 together the modified .tor and .tci files. The Working directory always contains the most recent data.

The advantage of this new structure is that one can easily check the results obtained at any time during the design process. Since the results are stored automatically in different directories it is no longer necessary to change the file names and, in fact, it is not necessary to define output file names at all, since this is done automatically by GRASP 10. Another very important advantage is that it is possible to revert to a previous design because the .tor and .tci files are kept for each individual run.

GRASP 10 is everywhere equipped with tool tips that pop up when hovering the mouse over an object, text field or menu button. Moreover, by right-clicking on the objects, jobs, drawing canvas and plotted results a number of possibilities are given to the user. For example, right-clicking on the drawing canvas gives immediate access to a number of facilities for manipulating the view.

The wizards for single and dual reflector antenna design work in the same way as in GRASP9, but the input parameters are given in a more convenient form.

GRASP 10 now also contains a wizard for the very popular ring focus systems. Two different types are available, a Cassegrain design and a Gregorian design.

The wizards now include a sketch of the feed element with approximately correct size. This is very useful in order to evaluate the influence of possible blockage from the feed.

The GRASP 10 graphical user interface is divided in four main windows:

1. Objects
2. Commands
3. Jobs
4. Results

Each window is activated by pressing the corresponding tab on the left side of the main window. These four windows will be described in the next sections.

8.1 The Objects Window

The Objects window is shown in Figure 8-2. It is divided in two parts. The left side shows a navigator, denoted "Object Explorer", and the right side presents automatically a 3D-view of the antenna structure. An object can be edited directly by double-clicking on the object either in the "Object Explorer" or in

the 3D-view. When an object is marked in the "Object Explorer" it changes colour in the 3D-view.

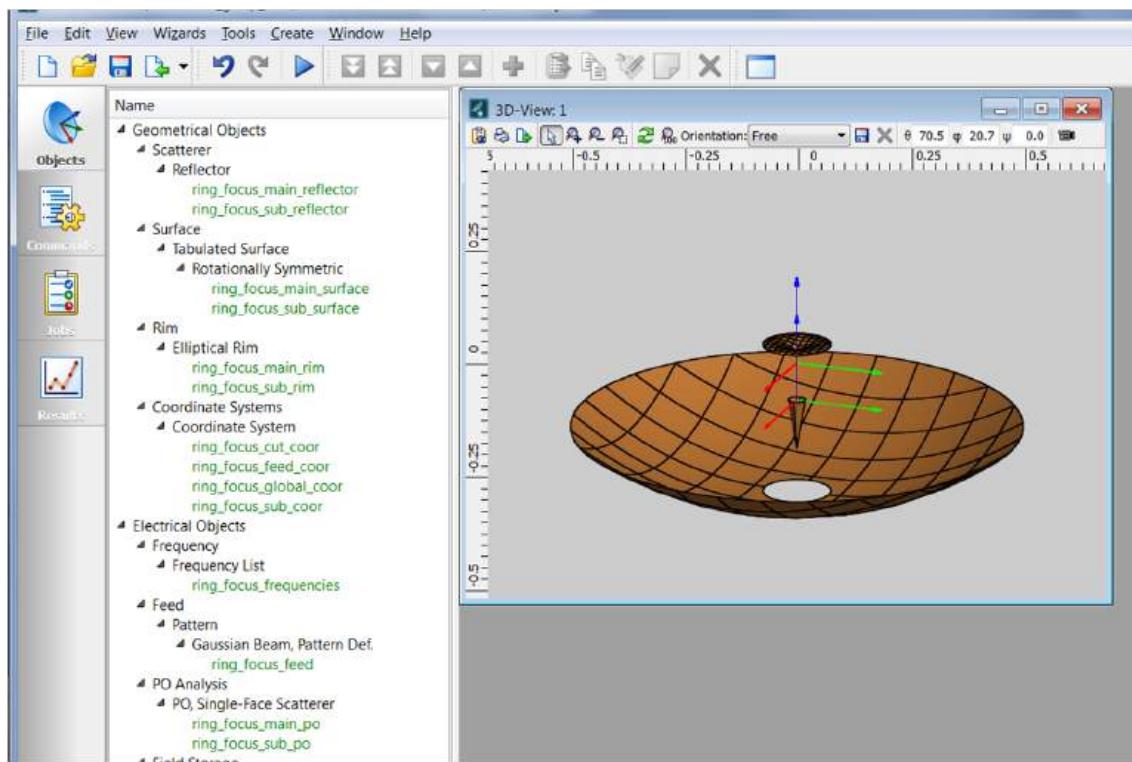


Figure 8-2 GRASP 10, Objects window.

Many of the object editor menus now have a check box, "Show Advanced Settings". If it is checked, all attributes are shown. If it is not checked, only the most important attributes are shown in the editor.

Right-clicking an object in the "Object Explorer" makes it possible to generate a copy of the object with another name.

A particularly nice feature in GRASP 10 is the possibility to rename an existing object. This was a very tedious task in GRASP9 because it had to be done manually in all objects and commands that were referring to this object. In GRASP 10 the name is checked for validity and automatically changed everywhere.

The GRASP9 plot objects are no longer necessary. Ray plots, which were generated by specific ray plot objects in GRASP9, are now generated in the "3D-view settings" menu, under "Visualization Objects". It is possible to have several 3D-views with different settings.

8.2 The Commands Window

All commands are specified in the Commands window. The command types are identical to those of GRASP9, the only difference being that it is now possible to have more than one source in several commands. One can for example generate the total field from the feed, the subreflector and the main reflector in one "Get Field" command. This would require one "Get Field" and two "Add Field" commands in GRASP9.

8.3 The Jobs Window

The Jobs window is shown in Figure 8-3. It is divided in two parts. The left side lists all the jobs while the right side shows the log file for each job.

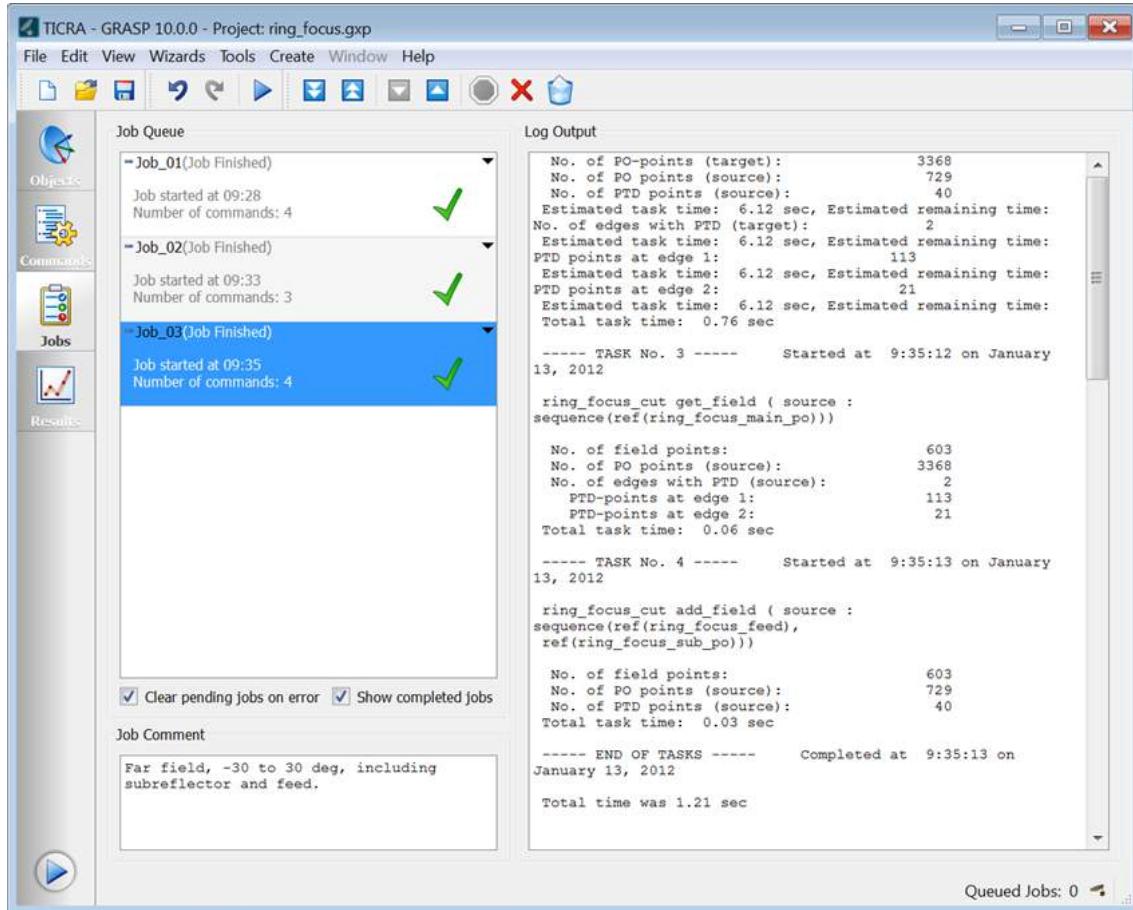


Figure 8-3 GRASP 10, Jobs window.

A new job is initiated by clicking the blue “play” button (this corresponds to the “submit” button in GRASP9). An updated job-number is automatically proposed and a small text field is presented where the user can write comments for this particular job. This job comment is written at the bottom of the job list to the left of the window. It is recommended to make use of this comment field because it helps identifying the job in the “Results” tab to be presented next.

8.4 The Results Window

The Results window is used to visualize the results of the computations as shown in Figure 8-4. The left side of the window is the Results Explorer showing a list of all the jobs while the right side is used to plot the results. The bottom of the Results Explorer repeats the job comments mentioned above when a particular job is selected.

By right-clicking on a job name, it is possible to:

1. "Revert to", which will load the objects used in the selected job and use

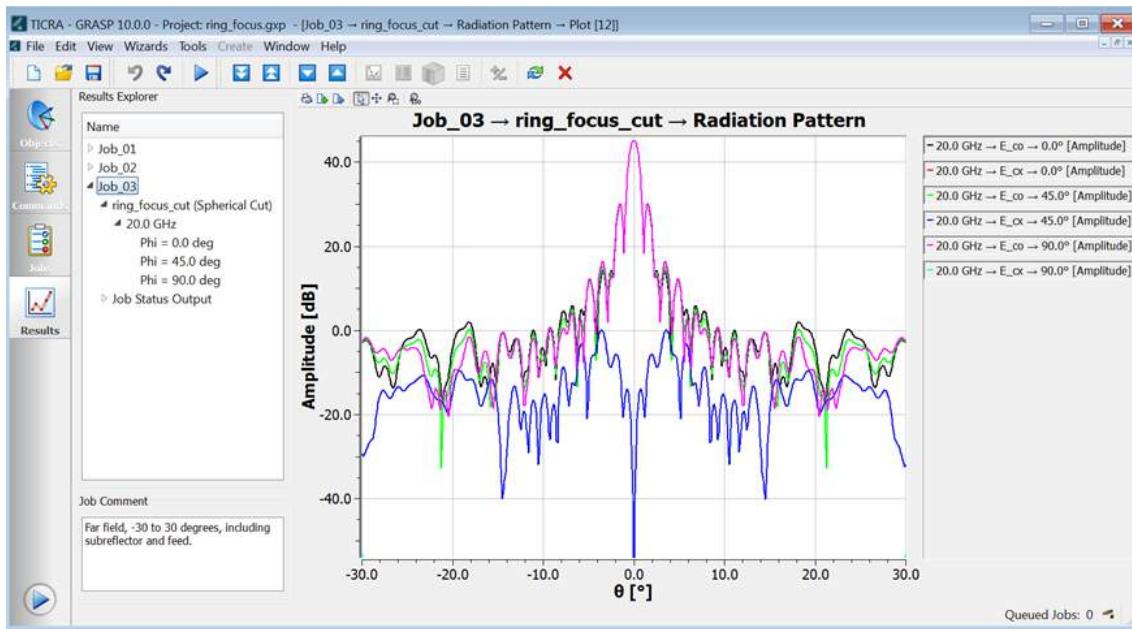


Figure 8-4 GRASP 10, Results window.

that as the active antenna geometry.

2. "Delete job", which will delete the calculated results from the disk and remove the selected entry from the Results Explorer.

The Results window replaces some of the features from the "Postprocessor" which was delivered together with GRASP9 and previous versions. Since many advanced features of the "Postprocessor", such as for example contour plots, are not yet available in GRASP 10, the "Postprocessor" is still included in the GRASP 10 package.

The different parts of the plot may be edited and manipulated by right-clicking at a curve, at the canvas, at the axes, at the text along the axes or at the legend area in the right side. Attention is also given to the green icon "+/-" on the top bar, which allows one to subtract and add curves from and to each other.

9. Reference Section

The reference section contains descriptions of the GRASP classes and command types, as well as the applicable units and file formats. A *Guide to the Reference Section*, containing descriptions of the used nomenclature as well as hints for navigation in this section, is found on the following pages. After the guide follows an *Alphabetical List of Classes and Command Types* from which descriptions of classes and command types may be easily reached.

Links

[Alphabetical List of Classes and Command Types](#)

[Classes](#)

[Command Types](#)

[Applicable Units](#)

[File Extensions](#)

[File Formats](#)

[Appendices](#)

[Contents](#)

9.1 Guide to the Reference Section

This section gives first a short introduction to the concept of objects and commands, which form the input to GRASP. Next, the structure of the documentation in the Reference Section will be described.

The normal mode for GRASP applies a graphical user interface which in the following will be referred to as the GUI.

9.1.1 Classes, objects and commands

The input to GRASP is specified in the form of objects and commands. An input object to GRASP is a set of data which describes e.g. a physical object such as a reflector or a feed, or a non-physical object such as a coordinate system. In order to distinguish between the objects these must each be given a unique name. The object names are chosen by the user; blanks and special characters - apart from underscore (_) - are not allowed.

Each object belongs to a certain class that determines which types of data the object contains. If, for instance, a dual reflector system is to be analyzed, the user can specify two objects of the class *Elliptical Rim* for defining the rims of the two reflectors and may choose the names Main_Rim and Sub_Rim for the two objects.

Objects belonging to the same class contain the same types of data: the attributes of the object. The values of the attributes are usually different. In the example above the two *Elliptical Rim* objects have the same attributes describing the centre and the half axes of the elliptical rim, but the values of the attributes are different.

The input objects to be build by the user belongs to one of the classes:

- Geometrical Objects
which describe geometrical entities (reflectors, plates, etc. but also coordinate systems and definitions of movements), and
- Electrical Objects
which describe electrical entities (frequencies/wavelengths, feeds but also cuts and grids specifying the points at which the field shall be determined).

Objects of a third class are generated automatically from the GUI. The descriptions of these are included in the Reference Manual for completeness:

- Plot Objects
which hold information on how the objects shall be depicted in the 3D-View of the GUI (the Graphical User Interface)

The analyses to be performed shall be specified by a set of commands. As for the objects, the commands may be of different types which may handle different operations. Typically, a command takes input from one object and return results in another such as the command *Get Field* which has a *Source* object as input and returns the field values in a *Field Storage* object.

Commands are executed in the order as they are specified and do therefore not need a name. Thus several commands of the same type may exist, e.g. [Get Field](#), without conflict.

In the following reference section, the objects are described class by class in a tree structure corresponding to the CREATE menu of the GUI under the tab OBJECTS (the tabs are placed to the left in the GRASP main window). Correspondingly, the commands are described type by type as found under the tab COMMANDS.

Throughout the reference sections, all *names* (*menu names*, *class names*, and *command names*) are hyperlinks and can be used to navigate through the document. In general, hyperlinks are written in red. The navigation is performed by pointing with the mouse on the name and then click the left mouse button. It is possible to navigate back again by the key-board shortcut Alt + left-arrow.

Description of all classes and command types may be reached from the [Alphabetical List of Classes and Command Types](#).

An introduction to how objects and commands are set up is given in Chapter 3, Getting Started. Further, more detailed examples on the use of GRASP are given in the tutorial examples of Chapter 6.

Analyses by GRASP may also be run as background jobs (batch mode). This is described in Chapter 7.

9.1.2 Description of Classes

The classes in GRASP appear in general under two different names. The first name is the *display name* which is displayed in the GUI. The other name is the *associated name* of the class (or the *class name*) and is used in the .tor file (where the object data are stored, see Chapter 5). In this manual, each heading shows the *display name* followed by the *class name* in parenthesis, e.g.:

COORDINATE SYSTEM (coor_sys)

The first section, **Purpose**, of each class explains the functionality of the class.

In the next section, **Links**, the path to the class in the GUI menu CREATE is given. For example, in the description of the Triangular Plate the menu path is given as

Classes → *Geometrical Objects* → *Scatterer* → *Plate* → *Triangular Plate*

which means that the class *Triangular Plate* can be found in the menu CREATE (under the OBJECTS) tab choosing GEOMETRICAL OBJECTS. Here it is found in the submenu SCATTERER and in this menu it is found as a PLATE. In the manual it is possible to navigate back through the menu tree by clicking the respective menus and thus see the description for example of other *Plates* or the full menu of *Geometrical Objects*.

The following section is **Syntax**. Here the attributes of the class are listed and the syntax given in which an object of this class is written in the .tor file.

In the section **Attributes**, the different attributes of the classes are described in details. The first line contains in general four terms, namely

- the display name of the attribute,
- the class name in italics and in parentheses,
- the type of the specification in brackets, e.g. [integer], and
- the default value (if any) in bold.

Examples are:

- Aperture Radius (*aperture_radius*) [real number with unit of length].

(for which a default value does not exist) and

- Coordinate System (*coor_sys*) [name of another object], default: **blank**.

The following lines give then a description of the attribute.

Some classes contain **Commands**. In that section, the relevant commands for activating the calculations are listed. See the description of command types below.

Last, but not least, some **Remarks**, directions and useful hints on the application of most of the classes are given.

9.1.3 Description of Command Types

The description of the command types is structured the same way as the description of the classes. Thus, the sections **Purpose**, **Links**, **Attributes** and **Remarks** in the description of the command types are the same as those in the description of the classes while the **Syntax** used in the .tci-file is slightly different because the commands operate on a **Target**.

9.2 Alphabetical List of Classes and Command Types

Classes and command types are listed according to their display names as they appear in the GUI. The associated name of the class or command type is specified in parenthesis following the display name. The descriptions of the classes and command types are reached by clicking the names written in red.

[Link to initial letter](#)

**A B C D E F G H I
J K L M N O P Q R
S T U V W X Y Z**

A

[Top of list](#)

<i>Activate Movement</i> (activate_movement)	1009
<i>Add Field</i> (add_field)	983
<i>Add OpenGL Plot (Obsolete)</i> (add ogl_plot)	1016
<i>Add Pattern</i> (add_pattern)	987
<i>Add XYZ Plot (Obsolete)</i> (add_xyz_plot)	1013
<i>Aperture in Screen</i> (aperture_in_screen)	283
<i>Aperture in Screen Plot</i> (aperture_in_screen_plot)	905
<i>Array</i>	667
<i>Array Element Data</i> (F5)	1084
<i>Array Element Excitation Coefficients</i> (F5.3)	1089

B

[Top of list](#)

<i>babinet_po</i>	see <i>PO, Aperture in Screen</i>
<i>Beam Grid Directions</i> (F2.4)	1068
<i>blockage</i>	see <i>Blockage Check</i>
<i>Blockage Check</i> (blockage_check)	413
<i>Body of Revolution</i>	244
<i>Body of Revolution Plot</i> (bor_plot)	909
<i>BoR Mesh</i> (bor_mesh)	248
<i>BoR-MoM</i> (bor_mom)	598
<i>Box</i> (box)	225
<i>Box Plot</i> (box_plot)	892
<i>Browse Ghosts</i> (browse ghosts)	1030
<i>Browse Incomplete</i> (browse incomplete)	1031
<i>Browse Objects</i> (browse objects)	1029

C[Top of list](#)

<i>Cardioid Pattern</i> (cardioid_pattern)	458
<i>cardioid_feed</i>	see <i>Cardioid Pattern</i>
<i>Change Coordinate System Base</i> (coor_sys_change_base)	1004
<i>Check Blockage</i> (check_blockage)	1010
<i>Circular Cone</i> (circular_cone)	314
<i>Circular Cylinder Surface</i> (circular_cylinder_surface)	316
<i>Circular Plate</i> (circular_plate)	216
<i>Circular Stiffeners</i> (circular_stiffeners)	264
<i>Circular Struts</i> (circular_struts)	219
<i>Circular Struts Plot</i> (circular_struts_plot)	889
<i>Circular Symmetric Reflector</i>	256
<i>circular_cylinder</i>	see <i>Circular Cylinder Surface</i>
<i>concentric_support_rings</i>	see <i>Concentric Support Rings</i>
<i>Compute Chain</i> (compute_chain)	1018
<i>Concentric Support Rings</i> (concentric_support_rings)	278
<i>Conic Mirror Surface</i> (conical_surface)	304
<i>Conical Horn</i> (conical_horn)	463
<i>conical_feed</i>	see <i>Conical Horn</i>
<i>coor_plot</i>	see <i>Coordinate System Plot</i>
<i>coor_sys_change_base</i>	see <i>Change Coordinate System Base</i>
<i>coor_sys_file</i>	see <i>Tabulated Coordinate System</i>
<i>Coordinate System</i> (coor_sys)	393
<i>Coordinate System Plot</i> (coor_sys_plot)	867
<i>Coordinate System, Euler Angles</i> (coor_sys_euler_angles)	396
<i>Coordinate System, GRASP Angles</i> (coor_sys_grasp_angles)	399
<i>Coordinate Systems</i>	392
<i>coordinate_system</i>	see <i>Coordinate System</i>
<i>Copy</i> (copy)	1026
<i>Coupling (Obsolete)</i> (coupling)	823
<i>Coupling Data</i> (F9)	1100
<i>Coupling Data in Cuts</i> (F9.1)	1101
<i>Coupling Data in Grids</i> (F9.2)	1102
<i>Coupling System</i> (coupling_system)	820
<i>Create Object</i> (create)	1023
<i>Currents Colour Plot</i> (currents_colour_plot)	933
<i>Currents Colour Plot File</i> (F3.2)	1073
<i>Currents Plot</i> (currents_plot)	919
<i>Currents Stored on File</i> (F3)	1071
<i>Curved Wire</i> (curved_wire)	241
<i>Cut</i>	696
<i>Cylindrical Cut</i> (cylindrical_cut)	712
<i>Cylindrical Grid</i> (cylindrical_grid)	750
<i>cylindrical_field_grid</i>	see <i>Cylindrical Grid</i>

D[Top of list](#)

<i>d3_plot</i>	see <i>Plot Settings (Obsolete)</i>
<i>Definition of Coordinate Systems</i> (F8)	1098
<i>Definition of Tabulated Coordinate System</i> (F8.1)	1099
<i>Delete Object</i> (delete)	1024
<i>Derived Electrical Data</i>	803
<i>Derived Geometry Data</i>	403
<i>DGR Intercostals</i>	263
<i>DGR Intercostals Plot</i> (dgr_intercostals_plot)	901
<i>Dielectric Layer</i> (dielectric_layer)	787

E[Top of list](#)

<i>el_prop_output</i>	see <i>Electrical Properties Data Output</i>
<i>Electrical Objects</i>	429
<i>Electrical Properties</i>	764
<i>Electrical Properties Data Output</i> (el_prop_data_output)	818
<i>Electrical Properties for Scatterers</i> (F6)	1090
<i>Ellipsoid</i> (ellipsoid)	300
<i>elliptic_feed</i>	see <i>Elliptical Pattern</i>
<i>Elliptical Pattern</i> (elliptical_pattern)	448
<i>Elliptical Rim</i> (elliptical_rim)	366
<i>Execute External Command</i> (execute_external_command)	1011
<i>Export to IGES File</i> (export_iges)	1000
<i>Export to STEP File</i> (export_step)	998
<i>External Command</i> (external_command)	837

F[Top of list](#)

<i>Feed</i>	436
<i>Feed Data in Regular Grid</i> (F5.2)	1087
<i>Feed Plot</i> (feed_plot)	912
<i>feed_cone_plot</i>	see <i>Plot Feed as Conical Horn</i>
<i>Field Data</i> (F2)	1055
<i>Field Data in Cuts</i> (F2.1)	1056
<i>Field Data in Rectangular Grid</i> (F2.2)	1060
<i>Field Data in Tabulated Directions</i> (F2.3)	1067
<i>Field Directions</i> (F2.5)	1069
<i>Field Storage</i>	695
<i>Files Read All</i> (files read all)	1027
<i>Files Write All</i> (files write all)	1028
<i>Finite Conductivity</i> (finite_conductivity)	797
<i>Frame</i> (frame)	842
<i>Frequency</i>	430
<i>Frequency List</i> (frequency)	431
<i>Frequency Range</i> (frequency_range)	432

Fundamental Mode Circular Waveguide (fundamental_mode_circular_waveguide) 469
fundamental_mode_waveguide see *Fundamental Mode Circular Waveguide*

G[Top of list](#)

<i>Gauss-Laguerre Beam Coefficients</i> (F13)	1116
<i>Gaussian Beam Tube</i> (gaussian_beam_tube)	963
<i>Gaussian Beam, Near-Field Def.</i> (gaussian_beam)	442
<i>Gaussian Beam, Pattern Def.</i> (gaussian_beam_pattern)	439
gaussian_beam_near_field_definition	see <i>Gaussian Beam, Near-Field Def.</i>
gaussian_feed	see <i>Gaussian Beam, Pattern Def.</i>
<i>General Array</i> (general_array)	668
<i>General Geometries</i> (F10)	1104
<i>Geometrical Objects</i>	177
<i>Get All OpenGL Plot (Obsolete)</i> (get_all_ogl_plot)	1017
<i>Get All XYZ Plot (Obsolete)</i> (get_all_xyz_plot)	1014
<i>Get Coordinate System</i> (get_coor_sys)	1003
<i>Get Coupling</i> (get_coupling)	997
<i>Get Currents</i> (get_currents)	972
<i>Get Field</i> (get_field)	982
<i>Get OpenGL Plot (Obsolete)</i> (get_ogl_plot)	1015
<i>Get Pattern</i> (get_pattern)	985
<i>Get Plane Wave Expansion</i> (get_plane_wave_expansion)	994
<i>Get Reflector Data</i> (get_reflector_data)	1002
<i>Get SWE</i> (get_swe)	993
<i>Get XYZ Plot (Obsolete)</i> (get_xyz_plot)	1012
<i>Grid</i>	725
grid_array	see <i>Planar Grid Array</i>
grid_array_file	see <i>Tabulated Planar Grid Array</i>
<i>GTD Analysis</i>	609
<i>GTD Circular Struts</i> (gtd_circular_struts)	637
<i>GTD Plate</i> (gtd_plate)	634
<i>GTD Plot</i> (gtd_plot)	924
<i>GTD Reflector</i> (gtd_reflector)	630
<i>GTD Scatterer</i>	629

H[Top of list](#)

<i>Half Wave Dipole</i> (half_wave_dipole)	534
<i>Helix</i> (helix)	530
<i>Helix or Dipole</i>	529
helix_feed	see <i>Helix</i>
<i>Hertzian Dipole</i> (hertzian_dipole)	536
<i>Hexagonal Horn</i> (hexagonal_horn)	489
<i>Hexagonal Wave Guide Modal Aperture Field</i> (F4.4)	1083
hexagonal_feed	see <i>Hexagonal Horn</i>
<i>Horn</i>	462
<i>Hybrid Mode Conical Horn</i> (hybrid_mode_conical_horn)	478

hybrid_mode_feed	<i>see Hybrid Mode Conical Horn</i>
<i>Hyperboloid</i> (hyperboloid)	298

L[Top of list](#)

<i>Ideal Grid</i> (ideal_grid)	765
<i>Imperfection Modelling</i>	347
<i>Individually Defined Gaps</i> (F1.3)	1046
<i>Individually Defined Panels</i> (panels_individually_defined)	199
irreg_array	<i>see General Array</i>
irreg_array_file	<i>see Tabulated General Array</i>
irreg_field_points	<i>see Tabulated uv-Points</i>
<i>Irregular xy-Grid, Pseudo Splines</i> (irregular_xy_grid_pseudo_splines)	343
<i>Irregular xy-Grid, Triangulation</i> (irregular_xy_grid_triangulation)	337
irregular_surface	<i>see Irregular xy-Grid, Pseudo Splines</i>
<i>Irregularly Spaced Feed Data</i> (F5.1)	1085

L[Top of list](#)

<i>Lens Plot</i> (lens_plot)	907
<i>Load</i> (load)	230
<i>Load Plot</i> (load_plot)	894

M[Top of list](#)

<i>Martin Puplett Interferometer</i> (martin_puplett_interferometer)	293
mesh	<i>see Wire Mesh</i>
<i>Meshed Geometries</i> (F10.1)	1105
<i>Microstrip Feed</i> (microstrip_feed)	520
<i>MoM</i> (mom)	587
<i>MoM Analysis</i>	586
<i>MoM Plot</i> (mom_plot)	939
<i>MoM Source Plot</i> (mom_source_plot)	943
mom_mesh_plot	<i>see MoM Plot</i>
<i>Movement Definition</i> (movement_definition)	416
<i>Multi GTD</i> (multi_gtd)	622
<i>Multi-GTD</i>	621
<i>Multi-Reflector GTD</i> (multi_reflector_gtd)	614
multi_face_po	<i>see PO, Multi-Face Scatterer</i>
multi_refl_gtd	<i>see Multi-Reflector GTD</i>

O[Top of list](#)

<i>Other Quadrics</i>	311
<i>Other Sources</i>	655

<i>Output Points Plot</i> (<code>output_points_plot</code>)	945
<code>output_plot</code>	see <i>Output Points Plot</i>

P

[Top of list](#)

<i>Panel Misalignments</i> (<code>F1.4</code>)	1048
<i>Panels in Polar Grid</i> (<code>panels_in_polar_grid</code>)	186
<i>Paraboloid</i> (<code>paraboloid</code>)	297
<i>Parallelogram</i> (<code>parallelogram</code>)	212
<code>parallelogram_plot</code>	see <i>Plate Plot</i>
<i>Pattern</i>	438
<i>Perfect Absorption</i> (<code>perfect_absorption</code>)	793
<i>Perfect Conductivity</i> (<code>perfect_conductivity</code>)	791
<i>Piecewise Linear Body of Revolution</i> (<code>piecewise_linear_bor</code>)	245
<i>Piecewise Straight Wire</i> (<code>piecewise_straight_wire</code>)	238
<i>Planar Cut</i> (<code>planar_cut</code>)	705
<i>Planar Feed</i>	519
<i>Planar Grid</i> (<code>planar_grid</code>)	744
<i>Planar Grid Array</i> (<code>planar_grid_array</code>)	677
<i>Plane</i> (<code>plane</code>)	302
<i>Plane Wave</i> (<code>plane_wave</code>)	656
<i>Plane Wave Expansion</i> (<code>plane_wave_expansion</code>)	658
<i>Plane Wave Plot</i> (<code>plane_wave_plot</code>)	916
<code>plane_cut</code>	see <i>Planar Cut</i>
<code>plane_field_grid</code>	see <i>Planar Grid</i>
<code>plane_gore_surface</code>	see <i>Pyramidal Surface</i>
<i>Plate</i>	203
<i>Plate Plot</i> (<code>plate_plot</code>)	886
<code>plate_gtd</code>	see <i>GTD Plate</i>
<i>Plot Feed as Conical Horn</i> (<code>plot_feed_as_conical_horn</code>)	930
<i>Plot Objects</i>	863
<i>Plot Settings (Obsolete)</i> (<code>plot_settings</code>)	864
<i>PO Analysis</i>	539
<i>PO Convergence (Obsolete)</i> (<code>po_convergence</code>)	580
<i>PO, Aperture in Screen</i> (<code>po_aperture_in_screen</code>)	569
<i>PO, Lens</i> (<code>po_lens</code>)	575
<i>PO, Multi-Face Scatterer</i> (<code>po_multi_face_scatterer</code>)	548
<i>PO, Panels in Polar Grid</i> (<code>po_panels_in_polar_grid</code>)	560
<i>PO, Polygonal Struts (Obsolete)</i> (<code>po_polygonal_struts</code>)	648
<i>PO, Single-Face Scatterer</i> (<code>po_single_face_scatterer</code>)	540
<code>po_circular_struts</code>	see <i>Strut Analysis, Circular Cross Section</i>
<i>Polygonal Stiffeners</i> (<code>polygonal_stiffeners</code>)	269
<i>Polygonal Struts</i> (<code>polygonal_struts</code>)	222
<i>Polygonal Struts Plot</i> (<code>polygonal_struts_plot</code>)	890
<code>polygonal_struts_po</code>	see <i>PO, Polygonal Struts (Obsolete)</i>
<i>Polynomial (Obsolete)</i> (<code>polynomial</code>)	321
<i>Potter Horn</i> (<code>potter_horn</code>)	474
<i>Power Splitting</i> (<code>power_splitting</code>)	794
<i>Pyramidal Surface</i> (<code>pyramidal_surface</code>)	361

Q[Top of list](#)

<i>Quast frame</i>	841
<i>Quilting Surface</i> (<i>quilting_surface</i>)	350
<i>Quit</i> (<i>quit</i>)	1032

R[Top of list](#)

<i>Random Surface</i> (<i>random_surface</i>)	348
<i>Ray Plot</i>	947
<i>Ray Traces</i> (F11)	1113
ray_coor_plot	see <i>Rays from Coordinate Systems</i>
ray_from_elements_plot	see <i>Rays from Array Elements</i>
ray_plane_plot	see <i>Rays from Plane Waves</i>
ray_plot	see <i>Rays from Point Sources</i>
<i>Rays from Array Elements</i> (<i>rays_from_array_elements</i>)	956
<i>Rays from Coordinate Systems</i> (<i>rays_from_coordinate_systems</i>)	959
<i>Rays from Plane Waves</i> (<i>rays_from_plane_waves</i>)	952
<i>Rays from Point Sources</i> (<i>rays_from_point_sources</i>)	948
<i>Real Variable</i> (<i>real_variable</i>)	833
rectangle	see <i>Rectangular Plate</i>
<i>Rectangular Horn</i> (<i>rectangular_horn</i>)	484
<i>Rectangular Plate</i> (<i>rectangular_plate</i>)	208
<i>Rectangular Rim</i> (<i>rectangular_rim</i>)	368
rectangular_feed	see <i>Rectangular Horn</i>
refl_plot	see <i>Reflector Plot</i>
<i>Reflector</i> (<i>reflector</i>)	180
<i>Reflector Data</i> (F1)	1041
<i>Reflector Data Output</i> (<i>reflector_data_output</i>)	404
<i>Reflector Plot</i> (<i>reflector_plot</i>)	871
<i>Reflector Rim Data</i> (F1.1)	1042
<i>Reflector with Panels</i>	185
<i>Reflector with Panels Plot</i> (<i>reflector_with_panels_plot</i>)	878
reflector_amp	see <i>Reflector Data Output</i>
reflector_gtd	see <i>GTD Reflector</i>
<i>Regular xy-Grid</i> (<i>regular_xy_grid</i>)	325
regular_grid	see <i>Regular xy-Grid</i>
<i>Rename</i> (<i>rename</i>)	1025
<i>Replace Pattern</i> (<i>replace_pattern</i>)	991
rib_surface	see <i>Unfurlable Surface, Rib Attached</i>
<i>Rim</i>	365
<i>Rim Data Output</i> (<i>rim_data_output</i>)	411
rim_output	see <i>Rim Data Output</i>
<i>Rooftop Mirror</i> (<i>rooftop_mirror</i>)	285
rotational	see <i>Rotationally Symmetric</i>
<i>Rotationally Symmetric</i> (<i>rotationally_symmetric</i>)	333
<i>Rotationally Symmetric Surface</i> (F1.2)	1044

S[Top of list](#)

<i>Scatterer</i>	178
<i>Scatterer Cluster</i> (<i>scatterer_cluster</i>)	228
<i>Scatterer Plot</i>	870
scissor_surface	<i>see Unfurlable Surface, Button Attached</i>
<i>Second Order Polynomial</i> (<i>polynomial_2nd_order</i>)	318
<i>Set Attribute</i> (<i>set</i>)	1021
<i>Simple Lens</i> (<i>simple_lens</i>)	288
<i>Simple Tapered Pattern (m=1)</i> (<i>simple_tapered_pattern</i>)	454
<i>Single-Reflector GTD</i> (<i>single_reflector_gtd</i>)	610
<i>single_refl_gtd</i>	<i>see Single-Reflector GTD</i>
<i>Source</i>	435
<i>Source Plot</i>	911
sphere	<i>see Spherical Surface</i>
<i>Spherical Cut</i> (<i>spherical_cut</i>)	697
<i>Spherical Grid</i> (<i>spherical_grid</i>)	726
<i>Spherical Surface</i> (<i>spherical_surface</i>)	312
<i>Spherical Wave ab-Coefficients</i> (F4.2)	1077
<i>Spherical Wave Expansion (SWE)</i> (<i>swe</i>)	804
<i>Spherical Wave Q-Coefficients</i> (F4.3)	1079
<i>spherical_feed</i>	<i>see Tabulated SWE Coefficients</i>
<i>spherical_field_grid</i>	<i>see Spherical Grid</i>
<i>Spiral Feed</i> (<i>spiral_feed</i>)	525
<i>Spline Reflector</i> (<i>spline_circ_sym_reflector</i>)	257
<i>Spline Surface</i> (<i>spline_surface</i>)	327
<i>Standard Currents File</i> (F3.1)	1072
<i>standard_feed</i>	<i>see Simple Tapered Pattern (m=1)</i>
<i>standard_po</i>	<i>see PO, Single-Face Scatterer</i>
<i>standard_po_plot</i>	<i>see Currents Plot</i>
<i>stiffeners_plot</i>	<i>see DGR Intercostals Plot</i>
<i>Strip Grid</i> (<i>strip_grid</i>)	768
<i>Strip Grid in Dielectric Layer</i> (<i>strip_grid_dielectric_layer</i>)	772
<i>Strut Analysis</i>	639
<i>Strut Analysis, Arbitrary Cross Section</i> (<i>strut_analysis_arbitrary_cross</i>)	640
<i>Strut Analysis, Circular Cross Section</i> (<i>strut_analysis_circ_cross</i>)	644
<i>Struts</i>	218
<i>Subtract Field</i> (<i>subtract_field</i>)	984
<i>Subtract Pattern</i> (<i>subtract_pattern</i>)	989
<i>Superelliptical Rim</i> (<i>superelliptical_rim</i>)	390
<i>Support Ring</i> (<i>support_ring</i>)	274
<i>Surface</i>	296
<i>Surface Cut</i> (<i>surface_cut</i>)	718
<i>Surface Data as Zernike Modes</i> (F1.7)	1053
<i>Surface Data in Irregular Points</i> (F1.6)	1052
<i>Surface Data in Rectangular Grid</i> (F1.5)	1051
<i>Surface Data Output</i> (<i>surface_data_output</i>)	406
<i>Surface Defined by Cubic Splines</i> (F1.8)	1054

<i>Surface Grid</i> (surface_grid)	756
surface_field_cut	see <i>Surface Cut</i>
surface_field_grid	see <i>Surface Grid</i>
surface_output	see <i>Surface Data Output</i>
<i>S, Y, and Z-Parameters</i> (F12)	1114

T[Top of list](#)

<i>Tabulated Coordinate System</i> (tabulated_coor_sys)	402
<i>Tabulated Electrical Properties</i> (tabulated_el_prop)	799
<i>Tabulated Electrical Properties for Scatterers</i> (F6.1)	1091
<i>Tabulated Feed</i>	495
<i>Tabulated Feed Data</i> (F4)	1074
<i>Tabulated General Array</i> (tabulated_general_array)	673
<i>Tabulated Mesh</i> (tabulated_mesh)	232
<i>Tabulated Mesh Plot</i> (tabulated_mesh_plot)	896
<i>Tabulated Pattern</i> (tabulated_pattern)	496
<i>Tabulated Pattern Data</i> (F4.1)	1075
<i>Tabulated Planar Grid Array</i> (tabulated_planar_grid_array)	683
<i>Tabulated Planar Source</i> (tabulated_planar_source)	688
<i>Tabulated Planar Source Plot</i> (tabulated_planar_source_plot)	927
<i>Tabulated Reflector</i> (tabulated_circ_sym_reflector)	260
<i>Tabulated Rim</i>	372
<i>Tabulated Rim (Obsolete)</i> (tabulated_rim)	388
<i>Tabulated Rim, rho-phi-Input</i> (tabulated_rim_rho_phi)	381
<i>Tabulated Rim, xy-Input</i> (tabulated_rim_xy)	373
<i>Tabulated Surface</i>	324
<i>Tabulated SWE Coefficients</i> (tabulated_swe_coefficients)	512
<i>Tabulated uv-Points</i> (tabulated_uv_points)	762
tabulated_feed	see <i>Tabulated Pattern</i>
tabulated_mom_mesh	see <i>Tabulated Mesh</i>
tabulated_mom_mesh_plot	see <i>Tabulated Mesh Plot</i>
tabulated_refl_trans	see <i>Tabulated Electrical Properties</i>
tabulated_rim_rho_phi_input	see <i>Tabulated Rim, rho-phi-Input</i>
<i>Three-Dimensional Plotter Data</i> (F7)	1096
<i>Three-Dimensional Plotter Data as XYZ-Values</i> (F7.1)	1097
triangle	see <i>Triangular Plate</i>
triangle_plot	see <i>Triangular Plate Plot</i>
<i>Triangular Plate</i> (triangular_plate)	204
<i>Triangular Plate Plot</i> (triangular_plate_plot)	882
<i>Triangular Rim</i> (triangular_rim)	370
triangulated_surface	see <i>Irregular xy-Grid, Triangulation</i>

U[Top of list](#)

<i>Unfurlable Surface</i>	355
<i>Unfurlable Surface, Button Attached</i> (unfurlable_surface_button_attached)	359
<i>Unfurlable Surface, Rib Attached</i> (unfurlable_surface_rib_attached)	356

V[Top of list](#)

<i>Variables and Ext. Commands</i>	832
<i>Voltage Generator</i> (voltage_generator)	604

W[Top of list](#)

<i>Wavelength List</i> (wavelength)	433
<i>Wavelength Range</i> (wavelength_range)	434
<i>Wire Grid</i> (wire_grid)	777
<i>Wire Mesh</i> (wire_mesh)	781
<i>Wires</i>	237
<i>Wires Plot</i> (wires_plot)	899
<i>Write MoM Mesh</i> (write_mom_mesh)	1007

X[Top of list](#)

<i>xy_tabulated_rim</i>	see <i>Tabulated Rim, xy-Input</i>
-------------------------------	------------------------------------

Z[Top of list](#)

<i>Zernike Surface</i> (zernike_surface)	329
--	-----

Links[Classes](#)[Command Types](#)[Applicable Units](#)[File Extensions](#)[File Formats](#)[Appendices](#)[Reference Section](#)[Contents](#)

9.3 Classes

The GRASP classes are divided into the following groups:

- *Geometrical Objects*
- *Electrical Objects*
- *Variables and Ext. Commands*
- *Quast frame*
- *Plot Objects*

Links

Alphabetical List of Classes and Command Types

Command Types

Applicable Units

File Extensions

File Formats

Appendices

Reference Section

Contents

GEOMETRICAL OBJECTS

Purpose

Geometry-related classes are collected in this menu. The following types are available:

Scattering structures, including reflectors, are specified in the menu

Scatterer.

The geometry of a reflector is defined by a surface limited by a rim, as specified in

Surface and

Rim, respectively.

The geometry may be described in individual user-defined

Coordinate Systems.

Data for a given reflector may be tabulated in

Derived Geometry Data.

For special purposes it is possible to move geometrical objects by means of

Movement Definition.

Links

Classes→*Geometrical Objects*

SCATTERER

Purpose

This class describes the geometries of the electromagnetic scatterers being present in the configuration under consideration. The following types of scatterers are available:

General curved surfaces:

Reflector and

Reflector with Panels.

Simple plane structures:

Plate.

Scattering struts, which support e.g. a feed or a subreflector:

Struts.

Regular boxes:

Box.

Load (absorbing wall) which terminates beams and rays:

Load.

Several scatterers may be combined to a single scatterer in

Scatterer Cluster.

MoM add-on: The following classes are parts of the add-on package MoM for specifying scatterers for Method of Moment calculations.

Scatterers in general are specified in

Tabulated Mesh.

and wires may be specified as

Wires.

For rotationally symmetric scatterers the rotational symmetry can be exploited in the MoM formulation such that it is possible to reduce the computation time and to handle larger scatterers than in the usual 3D-MoM. These scatterers are specified in

Body of Revolution.

Important scatterers for antenna designs are the intercostals (distance pieces) between the two reflectors in a dual gridded reflector system. These may be specified in

DGR Intercostals.

QUAST add-on: In the add-on package QUAST, the following scatterers are available:

An optical waveguide stop:

Aperture in Screen.

A rooftop mirror:

Rooftop Mirror.

The classical dielectric lens denoted:

Simple Lens.

Interferometers:

Martin Puplett Interferometer.

Command Types

The geometry of the following types of *Scatterers* may be exported to a CAD file: *Reflector*, *Plate*, *Struts*, and *Scatterer Cluster*.

The geometry may be exported in the STEP format by the command

Export to STEP File,

and in the IGES format by the command

Export to IGES File.

Links

Classes→*Geometrical Objects*→*Scatterer*

REFLECTOR (reflector)

Purpose

The class **Reflector** specifies a **Scatterer** which is a part of a general curved surface. The scattering by a **Reflector** may in general be determined by **PO Analysis**, **MoM**, or **GTD Analysis** (the exceptions are described in the remarks).

A rotational symmetric reflector for MoM calculations may also be specified from **Circular Symmetric Reflector**. The rotational symmetry is then exploited such that it is possible to handle larger reflectors than in the standard 3D-MoM.

Links

[Classes](#)→[Geometrical Objects](#)→[Scatterer](#)→[Reflector](#)

[Remarks](#)

Syntax

```
<object name> reflector
(
    coor_sys           : ref(<n>),
    surface            : ref(<n>),
    rim                : ref(<n>),
    centre_hole_radius : <rl>,
    holes              : sequence(ref(<n>), ...),
    distortions        : sequence(ref(<n>), ...),
    serration          : struct(inner_rim:ref(<n>), shape:<si>),
    el_prop            : sequence(ref(<n>), ...)
)
where
<n> = name of an object
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a **Coordinate Systems** object in which the reflector is specified. The default is the global coordinate system.

Surface (*surface*) [name of an object].

Reference to a **Surface** object that defines the shape of the reflector surface.

Rim (*rim*) [name of an object].

Reference to a **Rim** object that defines the edge of the reflector rim.

Centre Hole Radius (*centre_hole_radius*) [real number with unit of length], default: **0**.

The reflector may have a central hole which is a circle of radius Centre Hole Radius when projected on the *xy*-plane of the reflector coordinate system. Use the attribute *Holes* to define a reflector hole with more general shape and position.

Holes (*holes*) [sequence of names of other objects], default: **blank**.

Sequence of one or more references to *Rim* objects, each specifying a hole in the reflector surface. The shape of the hole, when projected on the *xy*-plane of the reflector coordinate system, is given by the *Rim* object. A central hole may be specified (by setting Centre Hole Radius to a positive value) along with the holes. The rim of a hole is not allowed to intersect a rim of another hole, the outer rim nor a possible central hole.

Distortions (*distortions*) [sequence of names of other objects], default: **blank**.

Sequence of one or more references to other *Surface* objects, which define surfaces to be added to the above defined reflector surface. This may be used to simulate surface distortions.

Serration (*serration*) [struct].

Definition of serrations on the edge of the reflector, as e.g. encountered in a compact antenna test range. Serrations can only be analysed by PO, and the attribute has no effect in GO or GTD analyses.

Inner Rim (*inner_rim*) [name of an object], default: **blank**.

Reference to a *Rim* object, which defines the base of the serrations. The centre of the inner rim must coincide with the centre of the reflector rim (defined by the attribute *rim* above), which in turn defines the tips of the serrations.

Shape (*shape*) [item from a list of character strings], default: **linear**.

Defines the shape of the serrations. The PO currents are given a weight, which varies from one at the base to zero at the tip of the serrations. The weight is a measure of the relative amount of solid reflector area in the serrated zone. The possible specifications are linear

When "linear" is chosen, the weight varies linearly from base to tip.

cosine

When "cosine" is chosen, the weight follows a co-sine-squared function.

Electrical Properties (*el_prop*) [sequence of names of other objects], default: **blank**.

Sequence of one or more references to objects of the class *Electrical Properties*. Each object defines electrical reflection and transmission properties of a layer of surface material. When no references are

given the surface will be assumed perfectly conducting. When at least one reference is given then a conducting layer is not included unless it is specified as one of the layers. When several references are given, the sequence of the *Electrical Properties* are sorted based on the *Displacement* attribute of each *Electrical Properties* object.

Command Types

Parameters for a *Reflector*, such as rim data, surface data or data describing its electrical properties, may be written to a file by the command

Get Reflector Data.

The commands available for exporting the geometry of a *Reflector* to a CAD file are described in *Scatterer*.

Remarks

This section contains discussions on the following topics:

- Analysis Methods
- Materials (*Electrical Properties*)
- Reflector Holes

Analysis Method

The scattering by a *Reflector* may be determined by

PO	Using the <i>PO, Single-Face Scatterer</i> or <i>PO, Multi-Face Scatterer</i> classes,
MoM	Using the <i>MoM</i> class,
GTD	Using the a <i>GTD Analysis</i> ,

with the following exceptions which cannot be handled:

PO	<i>Reflector</i> s with serrations (PO allowed, PTD not allowed), <i>Reflector</i> s with materials (PO allowed, PTD not allowed)
MoM	<i>Reflector</i> s with serrations, <i>Reflector</i> s with materials.

Materials (*Electrical Properties*)

A reflector has a front side and a backside. The front side has a surface normal with a positive component along the z -axis of the reflector coordinate system, while the backside has the opposite surface normal. Usually it is not important to know which is the backside and which is the front side, since this is automatically taken into account in the analysis. If, however, the reflector surface is composed of a number of layers, each with its own surface material, it is important to distinguish between the front side and backside, because the position of the layers is specified relative to the front side.

Reflector Holes

In GRASP, there are two ways of specifying a hole in the reflector:

- A simple definition, obtained if the attribute Centre Hole Radius is set to a positive value. This results in a so-called 'central hole' which in the projected *xy*-plane is a circle around the centre of the Reflector Rim (c.f. the individual *Rim* classes for the definition of the 'centre'). The radius of circle is the length set in the attribute Centre Hole Radius.
- A more general definition of a hole is obtained using the attribute Holes. The attribute allows a number of references to *Rim* objects. Each referenced object defines the rim of a hole in the projected *xy*-plane. This results in a number of holes, which may have any centre and any of the shapes available through the *Rim* classes. The rim of the hole is not allowed to intersect the outer rim, the rim of a central hole, or the rim of another hole.

When holes are defined in the reflector surface, they are handled as follows. PO currents only exist over the solid part of the reflector; if PTD currents are included they exist along the outer rim of the reflector as well as along the rims of the holes. For PTD (and GTD) calculations the edge numbering has importance and follows the numbering of the edges of the outer reflector rim, as described in the individual *Rim* classes. If a reflector has a central hole, the edge number of the hole is one higher than the largest edge number of the outer rim. The rim numbers of possible holes, specified by the Holes attribute, are assigned subsequent numbers according to the order the rims are listed in the sequence in the Holes attribute. For example, if a reflector has an *Elliptical Rim* with a central hole as well as a rectangular and a triangular hole, Figure 1, defined by

```
Holes: sequence(ref(rectangular_hole), ref(triangular_hole)),
```

then the outer rim has number 1, the central hole has rim number 2, the four edges of the rectangular hole have numbers 3, 4, 5, and 6, respectively, and the three edges of the triangular hole have numbers 7, 8, and 9, respectively. The rim numbers are not shown in the figure as the numbering depends on the way in which the rims of the holes are defined, see the actual *Rim* for the numbering system.

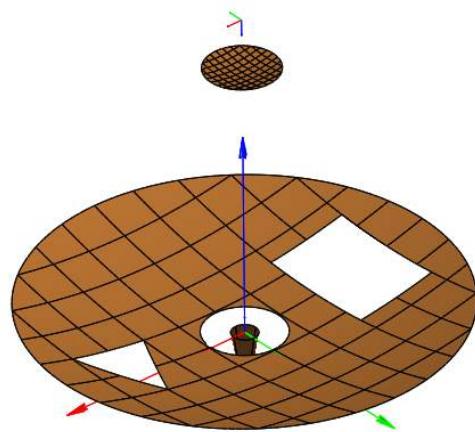


Figure 1

An example on a reflector with holes: a central hole defined by its Centre Hole Radius and a rectangular and a triangular hole, defined by the attribute Holes.

REFLECTOR WITH PANELS

Purpose

The class *Reflector with Panels* is a menu containing the reflectors constructed by surface panels. The panels may constitute a regular polar grid:

Panels in Polar Grid,

or the panels may be positioned individually:

Individually Defined Panels.

Links

[Classes](#)→[Geometrical Objects](#)→[Scatterer](#)→[Reflector with Panels](#)

PANELS IN POLAR GRID (panels_in_polar_grid)

Purpose

The class *Panels in Polar Grid* specifies a *Scatterer* (usually a reflector antenna), which is given by a number of panels arranged in a polar grid. An interpanel gap width may be specified. All panels belonging to the same ring of the polar grid have identical size and shape when projected onto the *xy*-plane of the coordinate system in which the *Panels in Polar Grid* reflector is defined.

In the initial configuration, all panels are part of the same general curved surface. However, it is possible to specify corrections to the position of each individual panel.

The scattering from *Panels in Polar Grid* may be determined by *PO, Panels in Polar Grid* and *PO, Multi-Face Scatterer*.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Reflector with Panels*→*Panels in Polar Grid*

Remarks

Syntax

```
<object name> panels_in_polar_grid
(
    coor_sys           : ref(<n>),
    surface            : ref(<n>),
    polar_grid         : sequence(
        struct(
            outer_rho_limit:<rl>,
            phi_0:<r>,
            n_panels:<i>),
        ...),
    centre             : struct(x:<rl>, y:<rl>),
    centre_hole_radius : <rl>,
    gap_widths         : struct(circular_gaps:<rl>,
                                radial_gaps:<rl>),
    distortion          : ref(<n>),
    misalignments_file : <f>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<f> = file name
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the reflector and its panels are specified (the reflector coordinate system). The default is the global coordinate system.

Surface (*surface*) [name of an object].

Reference to a *Surface* object that defines the shape of the *Panels in Polar Grid* reflector surface. Deviations from this surface may be specified in the attributes Distortion and Misalignments File below.

Polar Grid (*polar_grid*) [sequence of structs].

Defines the polar grid of panels. The grid is specified as a sequence of structs, where each struct defines a ring of panels. The rings must be given in the correct order starting with the innermost ring. All panels belonging to the same ring of the polar grid have *xy*-projections of identical size and shape.

Outer Rho Limit (*outer_rho_limit*) [real number with unit of length].

The ρ -coordinate of the outer limit of the current panel ring. The value includes half the circular gap width, except for the outermost ring (see the remarks below).

Phi-0 (*phi_0*) [real number], default: **0**.

The ϕ_0 -coordinate of the first panel in the present ring, positive from *x* towards *y* (see the remarks below).

No. of Panels (*n_panels*) [integer], default: **0**.

The number of panels in the present ring.

Centre (*centre*) [struct].

Defines the centre of the polar grid in the reflector coordinate system.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of the centre.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of the centre.

Centre Hole Radius (*centre_hole_radius*) [real number with unit of length], default: **0**.

The reflector may have a central hole which is a circle of radius *centre_hole_radius* when projected to the *xy*-plane of the coordinate system. The *centre_hole_radius* must be smaller than the outer radius of the panels in the innermost ring. The hole has the same centre as the polar grid.

Gap Widths (*gap_widths*) [struct].

Specification of the widths of the inter-panel gaps. All circular gaps are specified as having the same gap width. Similarly, all radial gaps are specified as having the same gap width. The specified gap width is measured tangentially to the surface.

The gap width used may be slightly modified as described in the remarks below.

Circular Gaps (*circular_gaps*) [real number with unit of length], default: **0**.

Specified width of all circular gaps.

Radial Gaps (*radial_gaps*) [real number with unit of length], default: **0**.

Specified width of all radial gaps.

Distortion (*distortion*) [name of an object], default: **blank**.

Reference to another *Surface* object, which defines a surface to be added to the above defined reflector surface. This may be used to simulate surface distortions. If left blank, there will be no distortions present.

Misalignments File (*misalignments_file*) [file name].

Name of file defining misalignments of some or all of the panels (See the remarks below). The structure of the file is described in *Panel Misalignments*.

Remarks

This section contains discussions on the following topics:

- Projection
- Polar Grid
- Numbering of panels, faces, edges and gaps
- Offset reflector
- Gap widths
- Panel misalignments

The polar-grid panels are analysed by PO and PTD as described in the class *PO, Panels in Polar Grid*.

Hints for the 3D-View Settings for *Panels in Polar Grid* can be found in the Remarks section in the description of class *Reflector with Panels Plot*

The projection

The *xy*-projection of a *Panels in Polar Grid* reflector is illustrated in Figure 1. This reflector consists of 37 panels arranged in 4 rings. There are 16 edges, 3 circular gaps and 36 radial gaps. These quantities are explained in the following.

The panels shown in Figure 1 are defined by the attributes listed in Figure 2

Polar grid

The *Panels in Polar Grid* object is defined in the reflector coordinate system $x_r y_r z_r$. A local *xy*-coordinate system is defined in the $x_r y_r$ -plane with axes parallel to the x_r - and y_r -axis. The origin of this local system is at $(x_r, y_r) =$

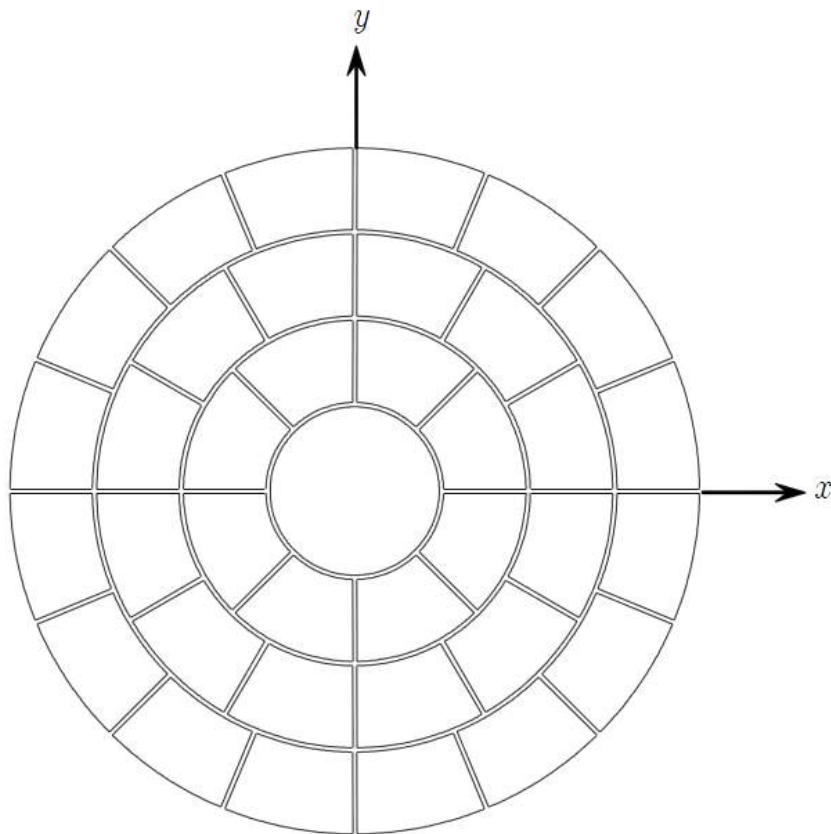


Figure 1 Projection of a polar grid

```
polar_grid: sequence(
    struct(outer_rho_limit:1 m, phi_0:0, n_panels:1),
    struct(outer_rho_limit:2 m, phi_0:0, n_panels:8),
    struct(outer_rho_limit:3 m, phi_0:30, n_panels:12),
    struct(outer_rho_limit:4.98 m, phi_0:90, n_panels:16),
),
gap_widths: struct(circular_gaps:4 cm, radial_gaps:4 cm),
```

Figure 2 The attribute settings defining the reflector shown in Figure 1.

(x_c, y_c) where x_c and y_c are the values of the attributes x and y , respectively, of attribute *centre*, cf. Figure 3.

In this local system a usual circular (ρ, ϕ) -coordinate system is then defined.

Each ring of the polar grid is specified as a struct in the attribute *polar_grid*. In the struct the outer radius of the ring of panels (*outer_rho_limit*), the position of the first radial gap (*phi_0*) and the number of panels (*n_panels*) shall be defined.

The *outer_rho_limit* is the radius along the centre line of the outer circular gap. In the outermost ring, the *outer_rho_limit* equals the radius of the reflector. The *phi_0*-value is the value of ϕ at the centre of one of the radial gaps in the ring. If there is only one panel in a ring, the *phi_0*-value is not

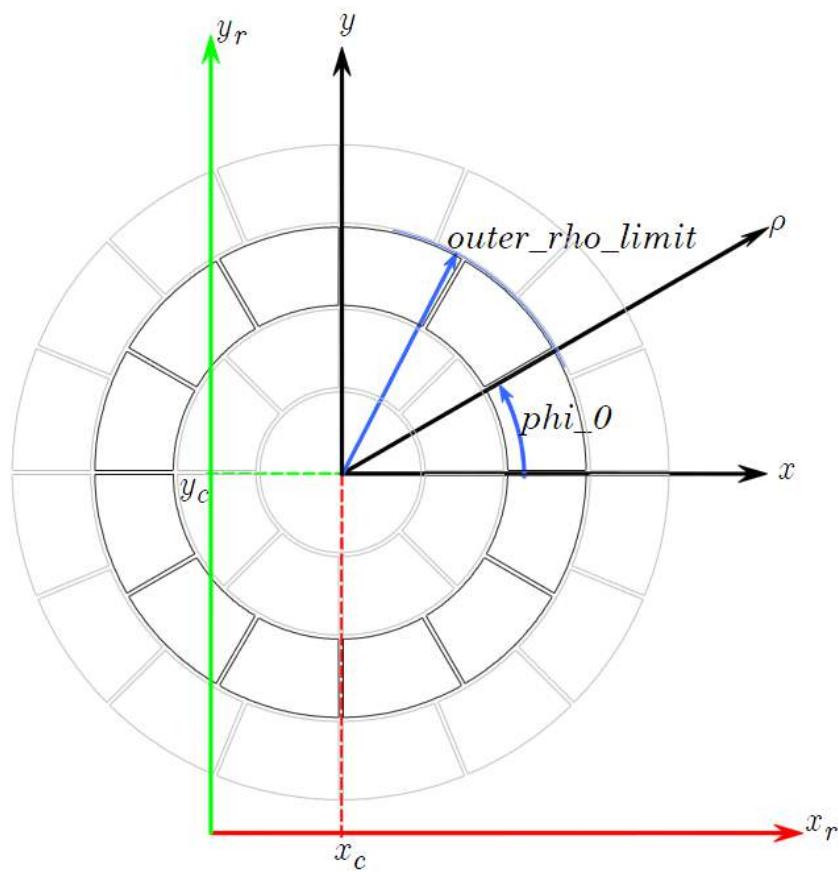


Figure 3

Parameters for definition of one ring of panels. This ring is the third ring of the previous example.

used. The definition of the polar grid is illustrated in Figure 3.

Numbering of panels, faces, edges and gaps

The panels are numbered in two different ways: In the specification of panel misalignments, the panels are numbered by two indices. In the specification of PO calculations, the panels are referred to as faces and numbered by a single index.

In the specification of panel misalignments, the panels are numbered by two indices: the ring number and the panel number starting with 1 in each ring such that panel 1 is the first panel (in direction of increasing ϕ) after the gap defined by the angle phi_0 of attribute *polar_grid*. The resulting panel numbering of the example of Figure 1 and Figure 2 is shown in Figure 4.

In PO calculation, the panels are referred to as faces and numbered by a single index. The faces of the reflector are numbered ring-by-ring starting with the innermost ring. In each ring the faces are numbered anti-clockwise with the first face belonging to panel 1 in the present ring. The edges of the reflector are the outer rims of the panels in the outermost ring. The edges are numbered anti-clockwise with the first edge belonging to the first panel in the outermost ring. The face- and edge-numbering (italic) are shown in Figure 5.

The gaps are numbered sequentially, starting with all circular gaps and then all radial gaps. With N rings of panels there are $N - 1$ circular gaps and the

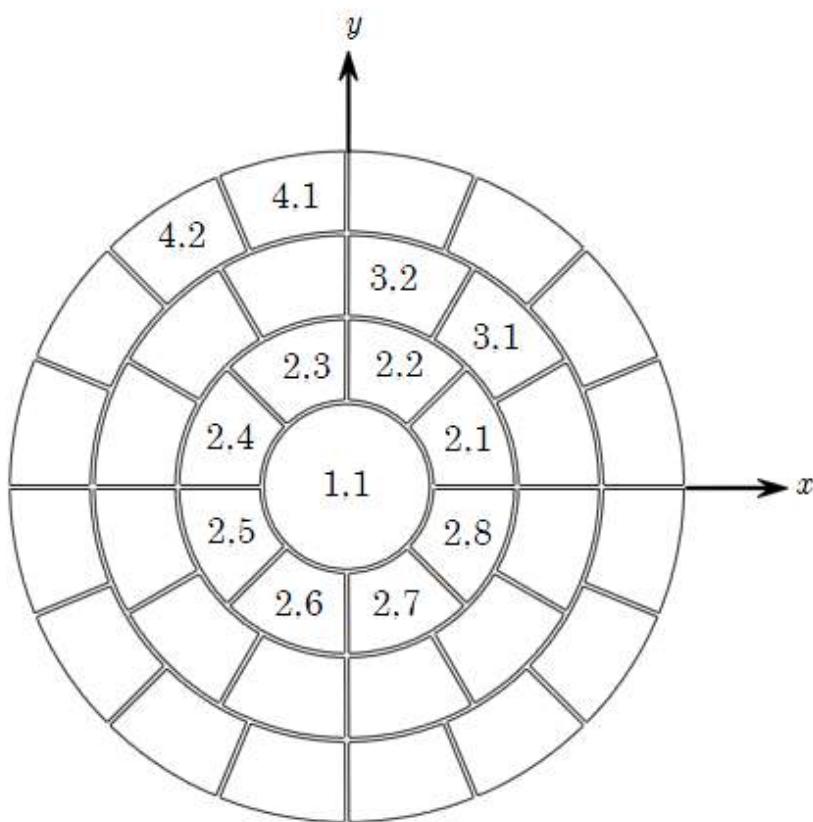


Figure 4 Numbering of panels used in the specification of panel misalignments. Note that for the third ring phi_0 is defined to $\text{phi_0} = 30^\circ$ and for the fourth ring $\text{phi_0} = 90^\circ$, cf. the attribute specifications in Figure 2.

circular gaps are numbered from the centre and outwards. The radial gaps are next numbered ring-by-ring starting from the innermost ring. In each ring the radial gap are numbered anti-clockwise with the first gap being at the angle phi_0 . The gap numbering is illustrated in Figure 6.

Offset reflector

The *Panels in Polar Grid* reflector may be an offset antenna. This is determined by the settings of the attributes *centre*, *surface* and *distortion* in the same way as for a simple Reflector not made by panels.

The *Panels in Polar Grid* reflector always has a circular rim. Further, in the *xy*-plane the polar grid and the rim centre are concentric. Hence, the *Panels in Polar Grid* class cannot be used to model a reflector such as the "Robert C. Byrd Green Bank Telescope" where the panels are arranged in a polar grid, but where the rim and the polar grid are not concentric (cf. the menu-item 'surface' at <http://www.gb.nrao.edu/gallery/gbt/index.html>).

Gap widths

A single gap width can be specified for all circular gaps. Similarly, a single gap width can be specified for all radial gaps. Both gap widths are given tangentially to the reflector surface.

Internally in the program the gap widths need to be specified in the *xy*-plane of the reflector coordinate system and the gap width can not vary along

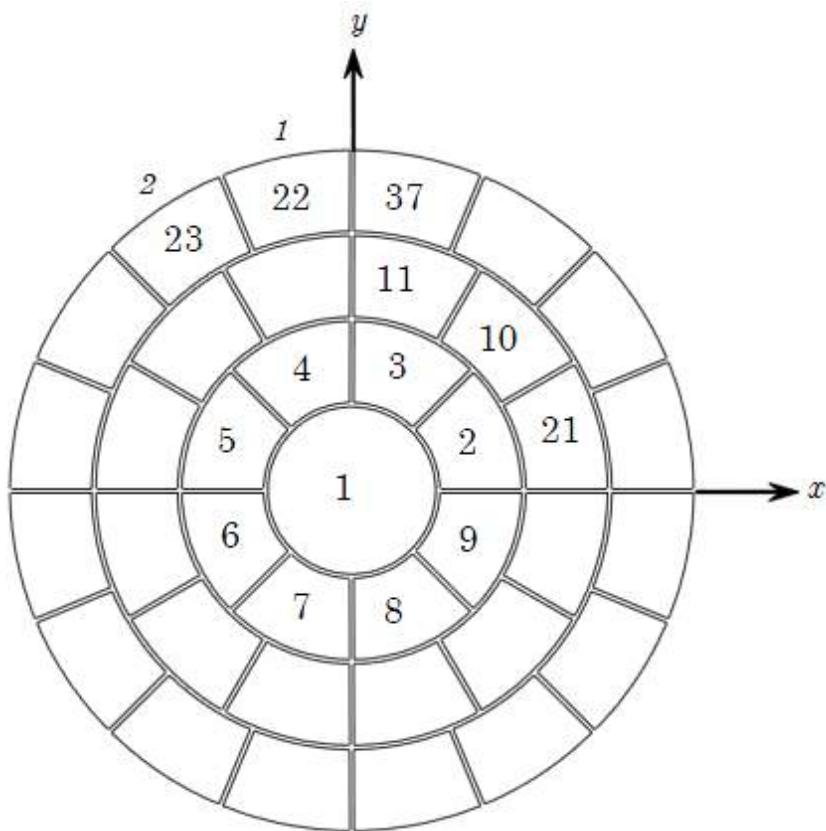


Figure 5 Numbering of faces and edges (*italic*) used in PO calculations.

a gap. However, a gap that has a constant width measured tangentially to the reflector surface does not necessarily project into a gap that has a constant width in the xy -plane, and vice versa. Therefore, a number of test points along each gap are projected onto the xy -plane and the average value of the projected gap width is calculated. This value is then used in the computations. As a consequence the tangential gap widths may vary slightly over the surface.

This is illustrated by the following two examples. In the first example a circular symmetric reflector is considered and the application of test points along the gap is not obvious. In the second example, however, an offset geometry is applied and the request for test points becomes clear.

Then, consider first the circular symmetric, parabolic *Panels in Polar Grid* reflector in Figure 7. The reflector has three rings with one panel in each ring. Hence, the centre panel is a disk and the two other panels are proper rings. We thus have two circular gaps and no radial gaps. The outer radii of the three rings are 4 m, 8 m and 12 m, respectively.

The two circular gaps are specified to a gap width of 0.5 m, denoted w in Figure 8 which shows the reflector from the side (with the z -values scaled by a factor 5 for a better illustration).

The data applied for the polar grid and the gap widths are written in the GRASP standard output file. The data are divided into four sections, cf. Figure 9:

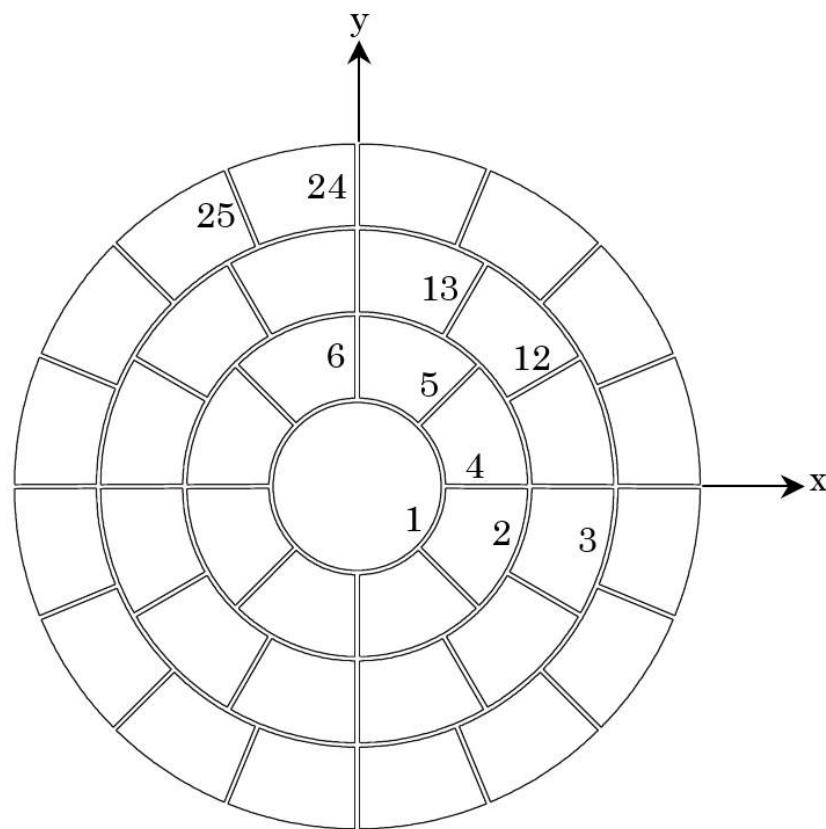


Figure 6 Numbering of gaps used in PO calculations.

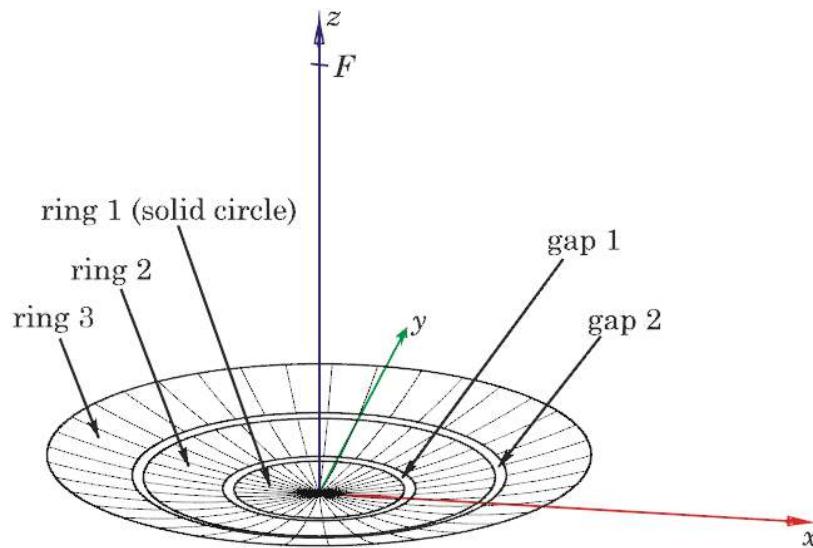


Figure 7 Front-fed reflector of three (circular symmetric) panels. Focal point at F . This and the following figures are based on drawings by [Reflector with Panels Plot](#) with 36 radial lines and 0 azimuthal lines in each ring panel.

- the first section tells that we have defined three rings forming two circular gaps,
- the next section (denoted Ring definition) gives the Polar Grid Data

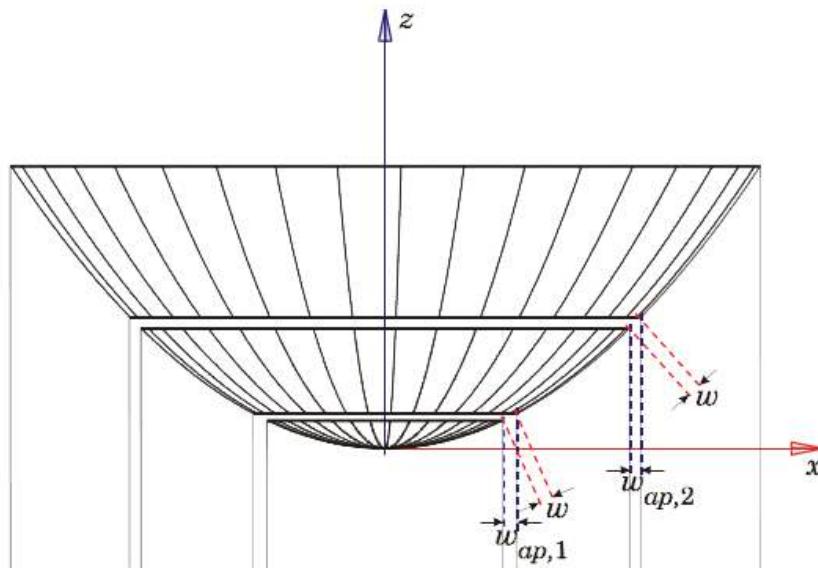


Figure 8 Side view of the front-fed reflector with the z -values scaled by a factor 5. The specified gap width for the circular gaps, w , is common for all circular gaps. When projected in the aperture plane, the widths vary from gap to gap, here $w_{ap,1}$ and $w_{ap,2}$.

defining the rings,

- the third section defines the circular gaps, and
- the fourth section defines the radial gaps – there are no radial gaps for this case.

The third section with the circular gaps data contains five columns. In the first column, the radius of the centre line of the circular gaps are listed, and in the second column (highlighted in red in Figure 9) the specified width of the gaps on the reflector are given, 0.5 m (w in Figure 8).

When the gaps are projected on the aperture plane, the xy -plane, the resulting widths ($w_{ap,1}$ and $w_{ap,2}$ in Figure 8) depend on the inclination of the reflector surface with the aperture plane. These widths are given in the third column (highlighted in blue in Figure 9).

In the present example with rotational symmetric gaps, the gaps in the aperture plane are realised by the original defined gaps of width 0.5 m upon the reflector. This is stated in the fourth and fifth columns (highlighted in green in Figure 9). The situation becomes more complicated when the reflector is not rotational symmetric. We therefore next consider the example of an offset reflector as depicted in Figure 10. The reflector has definitions identical to those of the front-fed reflector in Figure 7, except for the attribute *centre* which equals $(x, y) = (12m, 0m)$ for the offset configuration.

A side view of the offset reflector is shown in Figure 11 again (with the z -values scaled by a factor 5 for a better illustration).

The part of the standard output file which shows the data for the rings and the gaps is shown in Figure 12.

The panels_in_polar_grid consists of 3 panels in 3 rings with a total of 2 circular gaps and 0 radial gaps.

Ring definition:

Polar Grid Data					Panel Limits	
Ring no.	No. of panels	phi0 [deg]	rho-min [m]	rho-max [m]	rho-min [m]	rho-max [m]
1	1	0.000000E+00	0.000000E+00	0.400000E+01	0.000000E+00	0.375124E+01

Figure 9

The section of the standard output file describing the ring and the gap data for the front-fed reflector of Figure 7.

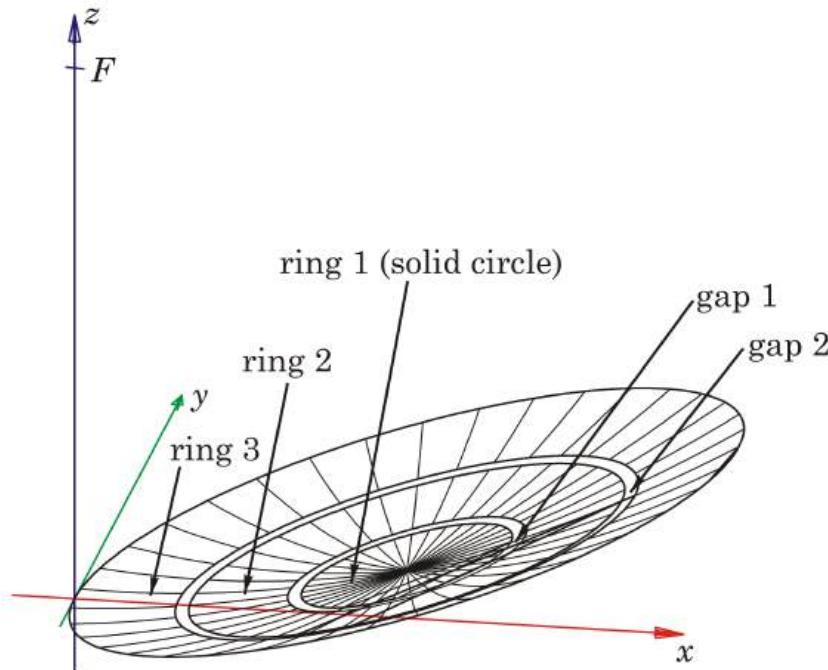


Figure 10

Offset-fed reflector of three (circular) panels

It is here the third section “Circular gaps data” which is interesting as the circular gaps are not rotational symmetric:

When the gaps are projected on the aperture plane, the xy -plane, the resulting gap widths ($w_{ap,2}$ for gap 2, cf. Figure 11) depend on the inclination of the reflector surface with the aperture plane. The width of each projected gap is therefore determined at 16 regularly distributed test points and an average value is applied. The average value is printed in the column “used width in aperture” in the third column highlighted in blue in Figure 12. For gap 1 the average value equals 0.4879 m (to four decimals).

The average width of the projected gaps are for each gap projected back (along the z -axis) upon the reflector surface at the 16 test points. The minimum and maximum values of the gap widths along the reflector surface are then found (printed in the fourth and fifth column “Realized width on reflector”, highlighted in green in Figure 12). It is seen that the hereby realized width of gap 1 varies from 0.4879 m to 0.5255 m. For gap 2 the values are 0.4824 m and 0.5393 m. This is between -4% and +8% from the specified value of 0.5 m.

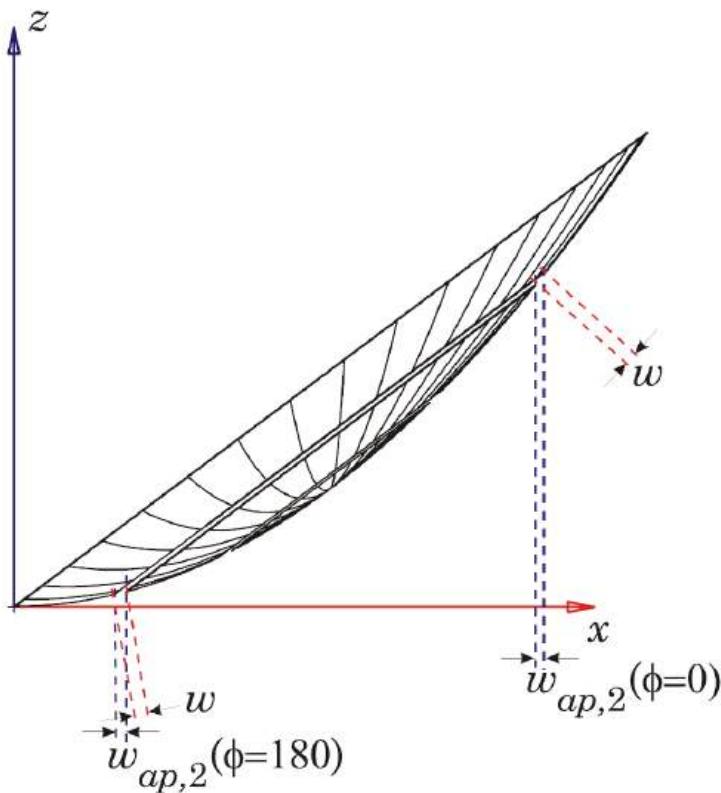


Figure 11

Side view of the offset reflector with the z -values scaled by a factor 5. The specified gap width for the circular gaps, w , is common for all circular gaps. When projected in the aperture plane, the width of the gaps vary with ϕ , the angular position along the circular gap. This is illustrated by the width of gap 2, $w_{ap,2}(\phi)$, which is smaller for $\phi = 0^\circ$ than for $\phi = 180^\circ$.

The panels_in_polar_grid consists of 3 panels in 3 rings with a total of 2 circular gaps and 0 radial gaps.

Ring definition:

	Polar Grid Data	Panel Limits				
Ring no.	No. of panels	phi0 [deg]	rho-min [m]	rho-max [m]	rho-min [m]	rho-max [m]
1	1	0.000000E+00	0.000000E+00	0.400000E+01	0.000000E+00	0.375606E+01

Figure 12

The section of the standard output file describing the ring and the gap data for the offset-fed reflector of Figure 10.

It is the gaps in the aperture plane with the “used width in aperture” which define the size of the modelled panels. The panels extend from their inner gap centre line to their outer gap centre line less the half widths of the respective gaps.

As an example, consider the panel in the innermost ring (ring 1). The panel is the central disk with no inner rim but the radius of its outer rim, rho-max, is found as the specified *outer_rho_limit* of this first ring, 4.0 m (cf. Figure 12, section “Ring definition”, header “Polar Grid Data”, column “rho-max”), minus the half of the projected width of the corresponding gap (“used width

in aperture"), i.e. for the panel we get

$$\rho_{\text{max}} = \frac{4.0 \text{ m} - 0.4879 \text{ m}}{2} = 3.7561 \text{ m}. \quad (1)$$

The found value is listed as rho-max under the header "Panel Limits" of the section of the output denoted "Ring definition".

The same procedure is applied for the radial gaps, and the procedure is applied independent of a possible reflector symmetry, as for Figure 9.

Panel misalignments

The panels of the reflector are considered to follow the ideal surface (given by the attributes *surface* and *distortion*) as long as misalignments are not specified. For practical applications misalignments may occur and these are specified panel by panel.

In the *misalignments_file*, the misalignments of some or all of the panels may be specified. For each panel in the file, three or more control points are introduced, cf. Figure 13. For each of these control points the user specifies a misalignment vector defined as the vector from the position of the control point on the ideally positioned panel (shown in red) to the actual position of the control point on the misaligned panel (shown in black).

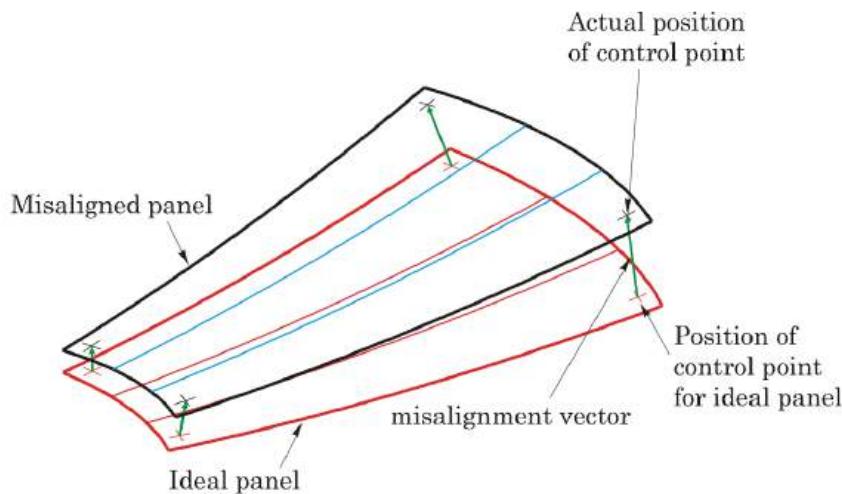


Figure 13

Illustration of the misalignment modelling for a case where the misaligned control points can be reached by translation and rotations of the ideal panel. In red is shown a panel modelled as part of the ideal surface. Near each of the four corners a control point is indicated by a cross. From each of these crosses a misalignment vector points at an actual position of the control point (black crosses). The panel with its control points can be brought undistorted to that position (shown in black).

Each panel is assumed to be stiff and is displaced (translated and rotated) to fit the specified misalignment.

If it is not possible to fit the stiff panel to the misaligned control points, then the panel is displaced such that the distances from the specified, misaligned control points to the control points of the modelled panel are minimised (least square sum of the distances), see Figure 14.

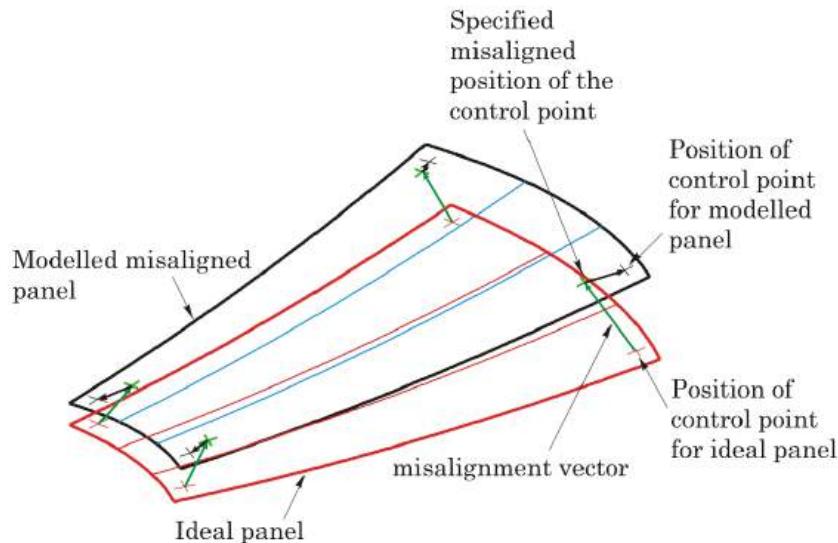


Figure 14

Illustration of the misalignment modelling for a case where the misaligned control points can not be reached by translation and rotations of the ideal panel. In red is again shown the panel modelled as part of the ideal surface. Near each of the four corners a control point is indicated by a cross. From each of these crosses the displacement vector points on the specified misaligned position of the control point (green). The panel is modelled at the position (shown in black) where the distances from the actual control points to the modelled controlled points (which follow the panel that remains undistorted) are minimised.

In the *misalignments_file* the positions of the control points for the ideally positioned panel are specified by their *x*- and *y*-coordinates. The displacement vectors are given by their *x*-, *y*- and *z*-components.

INDIVIDUALLY DEFINED PANELS (panels_individualy_defined)

Purpose

The class *Individually Defined Panels* specifies a *Scatterer*, which is formed by a number of panels defined individually. The panels may be arranged arbitrarily.

For each panel a hinge may be specified. Each hinge rotates the panel around a specified axis.

The scattering from *Individually Defined Panels* may be determined by *PO*, *Multi-Face Scatterer*.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Reflector with Panels*→*Individually Defined Panels*

Remarks

Syntax

```
<object name> panels_individualy_defined
(
    coor_sys : ref(<n>),
    panels   : sequence(
        struct(
            surface:ref(<n>),
            rim:ref(<n>),
            distortion:ref(<n>),
            hinge_coor_sys:ref(<n>),
            hinge_rotation:<r>),
        ...))
)
where
<n> = name of an object
<r> = real number
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the *Individually Defined Panels* reflector is specified. The default is the global coordinate system.

Panels (*panels*) [sequence of structs].

Definition of the geometry of the panels. The geometry is specified as a sequence of structs, where each struct defines one panel.

Surface (*surface*) [name of an object].

Reference to a *Surface* object that defines the shape of the panel surface. The surface is defined in the coordinate system specified by *coor_sys*.

Rim (*rim*) [name of an object].

Reference to a *Rim* object that defines the panel rim. The rim is defined in the coordinate system specified by *coor_sys*.

Distortion (*distortion*) [name of an object], default: **blank**.

Reference to a *Surface* object, which define surfaces to be added to the above defined panel surface.

Hinge Coordinate System (*hinge_coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object defining the position and orientation of a possible hinge. The current panel rotates around the *z-axis* of the *hinge_coor_sys* (See the remarks below).

Hinge Rotation (*hinge_rotation*) [real number], default: **0**.

The rotation of the panel around the *z-axis* of the *hinge_coor_sys* (See the remarks below).

Remarks

The panels are analysed by PO and PTD as described in the class *PO, Multi-Face Scatterer*. Gaps between the panels appear as the space between the defined *rims* of the panels but it is emphasised that a gap analysis with interaction between the two rims of the gap is not included in this class.

Hints for the 3D-View Settings for *Individually Defined Panels* can be found in the Remarks section in the description of class *Reflector with Panels Plot*

Numbering of panels, faces and edges.

The panels are numbered sequentially according to the appearance in the attribute *panels*. In the specification of the PO calculations, the panels are referred to as faces. The face number equals the panel number.

The edges are numbered sequentially panel-by-panel starting with the edge(s) of panel 1. The edge numbering for the individual panels follows the numbering defined for the *Rim* of the panel. If for example, panel 1 has an *Elliptical Rim* and panel 2 has a *Rectangular Rim*, then there are a total of five edges. Edge 1 is the *Elliptical Rim* and the edges 2 through 5 are the four edges of the *Rectangular Rim* in the order specified in the *Rectangular Rim* class.

Hinges

Hinges may be defined to model non-perfect alignment or incorrect deployment of panels connected by hinges. The hinge is specified to GRASP by a coordinate system, *hinge_coor_sys*, and a rotation angle, *hinge_rotation*.

It is not required to specify a hinge coordinate system. In that case a hinge is not defined and the attribute *hinge_rotation* has no effect.

The same *Coordinate Systems* object must not be referred to from more than one panel and it is strongly recommended that coordinate systems referred to as a *hinge_coor_sys* are not applied for any other purposes.

The *z-axis* of the *hinge_coor_sys* defines the hinge rotation axis. Otherwise the definition of the coordinate system may be chosen arbitrarily.

When the rotation angle equals 0, the panels are perfectly aligned or deployed in the sense that the panel definition (given by the specification in *surface* and *rim*) is left unchanged. When the rotation angle differs from 0, the current panel is rotated this angle around the *z*-axis of the hinge coordinate system. The rotation is positive according to the right-hand rule, i.e. when the *x*-axis rotates towards the *y*-axis.

The positions and orientations of the *hinge_coor_sys* are not affected by the value of the *hinge_rotation*. It is only the panels that are rotated.

The coordinate system of a hinge may be specified relative to the coordinate system of another hinge. Consider the following example:

Let panel 7 be mounted with a hinge on the edge of panel 2. This is done by specifying *hinge_coor_sys* to CS_Hinge7, a coordinate system which is defined with its *z*-axis along the appropriate edge of panel 2, cf. Figure 1. In the same way panel 2 is hinged on the edge of panel 1 by specifying *hinge_coor_sys* to CS_Hinge2 for panel 2. In Figure 1, the *hinge_rotation* is specified to zero for all panels. Only panels 1, 2 and 7 are drawn.

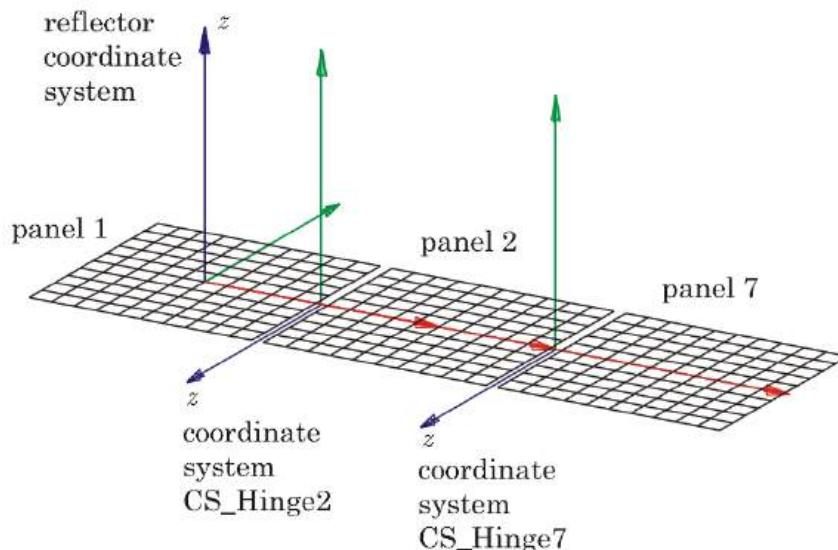


Figure 1 Panel 7 is hinged on the right edge of panel 2 which is hinged on the right edge of panel 1; no panel rotations.

As panel 7 is hinged upon panel 2 it follows the movements of panel 2 and CS_Hinge7 shall be specified relative to CS_Hinge2. A *hinge_rotation* of panel 2 will then affect both panel 2 and panel 7, see Figure 2 in which the *hinge_rotation* is specified to 10° for panel 2 as well as for panel 7. The hinge of panel 7 (the axis around which panel 7 rotates) is mounted on panel 2 and therefore follows the movements of panel 2, but the coordinate system of panel 7, CS_Hinge7, is not affected by the *hinge_rotation*.

If, alternatively, CS_Hinge7 is specified in the reflector coordinate system then a *hinge_rotation* of panel 2 will affect only this panel, not panel 7. This is illustrated in Figure 3.

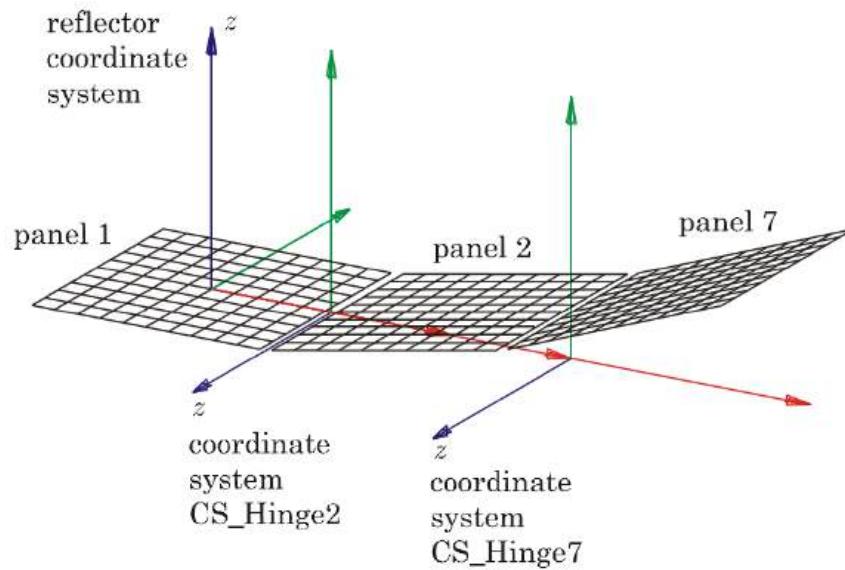


Figure 2

As Figure 1 but now panel 2 is rotated 10° relative to panel 1 and panel 7 is rotated 10° relative to panel 2. Note that the coordinate systems are not affected by the *hinge_rotation*.

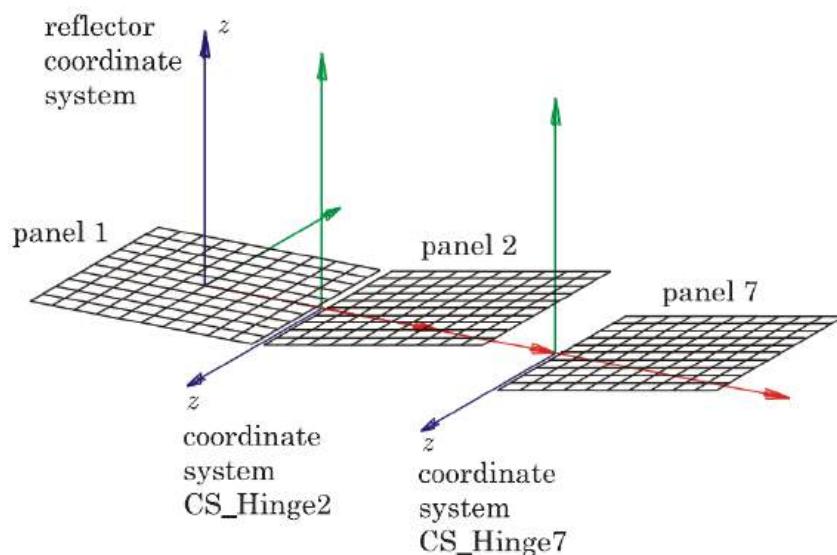


Figure 3

As Figure 2 with both panel 2 rotated 10° relative to panel 1 and panel 7 rotated 10°, also relative to panel 1 because the coordinate system CS_Hinge7 here is defined relative to the reflector coordinate system.

PLATE

Purpose

In the menu *Plate* it is possible to describe the plane structures:

Triangular Plate

Rectangular Plate

Parallelogram and

Circular Plate.

When PO/PTD is applied to a *Plate*, the PTD contribution is based on second-order equivalent edge currents that take into account the finiteness of the incremental strip where the PTD fringe current is integrated. See the Technical Description of GRASP (PTD) for details. As a consequence, a *Plate* is more accurately analysed by PO/PTD than the corresponding plane *Reflector*. The difference is most distinct for an electrically small *Plate*.

Command Types

The commands available for exporting the geometry of a *Plate* to a CAD format are described in *Scatterer*.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Plate*

TRIANGULAR PLATE (triangular_plate)

Purpose

The class *Triangular Plate* is used to define a *Scatterer*, which is a plane with a triangular rim.

The scattering from a *Triangular Plate* may be determined by *PO Analysis* (with *PO*, *Single-Face Scatterer*), *MoM* and *GTD Analysis* (except for plates with materials, see the remarks for details).

Links

Classes→*Geometrical Objects*→*Scatterer*→*Plate*→*Triangular Plate*

Remarks

Syntax

```
<object name> triangular_plate
(
    coor_sys           : ref(<n>),
    corner_1          : struct(x:<rl>, y:<rl>, z:<rl>),
    corner_2          : struct(x:<rl>, y:<rl>, z:<rl>),
    corner_3          : struct(x:<rl>, y:<rl>, z:<rl>),
    el_prop           : sequence(ref(<n>), ...)
```

)

where

<n> = name of an object

<rl> = real number with unit of length

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the triangular plate is specified.

Corner 1 (*corner_1*) [struct].

Defines the coordinates of the first corner of the triangular plate. The corner is also origin in the local coordinate system.

x (x) [real number with unit of length].

x-coordinate of the first corner.

y (y) [real number with unit of length].

y-coordinate of the first corner.

z (z) [real number with unit of length].

z-coordinate of the first corner.

Corner 2 (*corner_2*) [struct].

Defines the coordinates of the second corner of the triangular plate.

x (x) [real number with unit of length].

x-coordinate of the second corner.

y (y) [real number with unit of length].

 y-coordinate of the second corner.

z (z) [real number with unit of length].

 z-coordinate of the second corner.

Corner 3 (corner_3) [struct].

 Defines the coordinates of the third corner of the triangular plate.

 x (x) [real number with unit of length].

 x-coordinate of the third corner.

 y (y) [real number with unit of length].

 y-coordinate of the third corner.

 z (z) [real number with unit of length].

 z-coordinate of the third corner.

Electrical Properties (el_prop) [sequence of names of other objects], default:
blank.

Sequence of one or more references to objects of the class *Electrical Properties*. Each object defines electrical reflection and transmission properties of a layer of surface material.

When no references are given the plate will be assumed perfectly conducting. When at least one reference is given then a conducting layer is not included unless it is specified as one of the layers. When several references are given, the sequence of the *Electrical Properties* are sorted based on the Displacement attribute of each *Electrical Properties* object.

Command Types

The commands available for exporting the geometry of a *Triangular Plate* to a CAD file are described in *Scatterer*.

Remarks

Analysis methods

The scattering from a *Triangular Plate* may be determined by *PO Analysis* (with *PO*, *Single-Face Scatterer*), *MoM* and *GTD Analysis*.

A *Triangular Plate* with materials can, however, only be analysed with *PO*, *Single-Face Scatterer* (without PTD) and *GTD Analysis*.

Geometry

The direction of the normal vector to the triangular plate is defined by the right-hand rule applied to the corner points in the order Corner 1, Corner 2, and Corner 3.

When the *Triangular Plate* is part of a larger structure it is suggested to specify the corners so the normals of all elements of the structure point out of the structure such that the overall direction of the normal does not change from element to element.

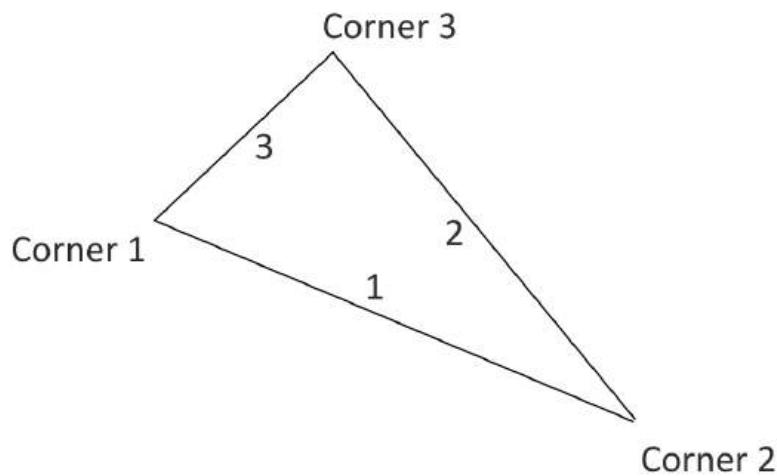


Figure 1

A triangular plate seen from the side of the positive normal.

The edges of the triangular plate are numbered as shown in Figure 1. This numbering is important when PTD is employed in the field analysis, since it is possible to specify different sampling densities along the individual edges (see the class *PO, Single-Face Scatterer*).

For the purpose to calculate a field across the surface of the triangular plate, a local xy -coordinate system is defined in the plane of the triangular plate. The origin of this local coordinate system is located at Corner 1 and the x -axis is oriented from Corner 1 to Corner 2. The y -axis is perpendicular to the x -axis and oriented such that Corner 3 has a positive y -coordinate. The z -axis is then parallel to the normal of the triangle, see Figure 2.

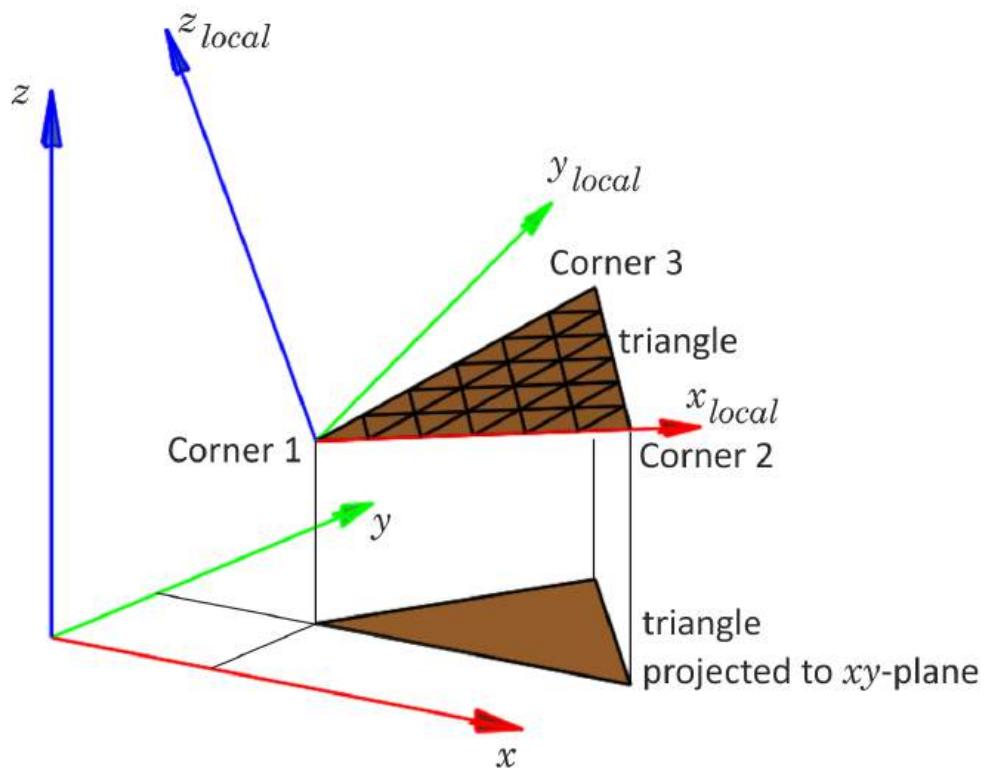


Figure 2

The triangular plate is defined in the (x, y, z) -coordinates according to the attribute Coordinate System. The projection of the triangular plate in the xy -plane is also shown. The number of lines on the triangular plate and parallel to the edges of the triangle are all set to 5 in the 3D-View Settings window. For illustration purposes, a local coordinate system, $(x_{local}, y_{local}, z_{local})$, is shown with the triangular plate in the (x_{local}, y_{local}) -plane. The normal of the plate is co-parallel to the z_{local} -axis.

RECTANGULAR PLATE (rectangular_plate)

Purpose

The class *Rectangular Plate* is used to define a *Scatterer*, which is a plane with a rectangular rim.

The scattering from a *Rectangular Plate* may be determined by *PO Analysis* (with *PO*, *Single-Face Scatterer*), *MoM* and *GTD Analysis* (except for plates with materials, see the remarks for details).

Links

Classes→*Geometrical Objects*→*Scatterer*→*Plate*→*Rectangular Plate*

Remarks

Syntax

```
<object name> rectangular_plate
(
    coor_sys           : ref(<n>),
    corner_1          : struct(x:<rl>, y:<rl>, z:<rl>),
    corner_2          : struct(x:<rl>, y:<rl>, z:<rl>),
    opp_point         : struct(x:<rl>, y:<rl>, z:<rl>),
    el_prop           : sequence(ref(<n>), ...)
)
where
<n> = name of an object
<rl> = real number with unit of length
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the rectangular plate is specified.

Corner 1 (*corner_1*) [struct].

Defines the coordinates of one corner of the rectangular plate. The corner is also origin in a local coordinate system.

x (x) [real number with unit of length].

x-coordinate of Corner 1.

y (y) [real number with unit of length].

y-coordinate of Corner 1.

z (z) [real number with unit of length].

z-coordinate of Corner 1.

Corner 2 (*corner_2*) [struct].

Coordinates of a corner of the rectangular plate, adjacent to Corner 1. The direction from Corner 1 to Corner 2 is also defining the x -axis of the local coordinate system.

x (x) [real number with unit of length].

x -coordinate of Corner 2.

y (y) [real number with unit of length].

y -coordinate of Corner 2.

z (z) [real number with unit of length].

z -coordinate of Corner 2.

Opp Point (*opp_point*) [struct].

Coordinates of any point on the edge opposite to that defined by Corner 1 and Corner 2

x (x) [real number with unit of length].

x -coordinate of Opp Point.

y (y) [real number with unit of length].

y -coordinate of Opp Point.

z (z) [real number with unit of length].

z -coordinate of Opp Point.

Electrical Properties (*el_prop*) [sequence of names of other objects], default: **blank**.

Sequence of one or more references to objects of the class *Electrical Properties*. Each object defines electrical reflection and transmission properties of a layer of surface material.

When no references are given the plate will be assumed perfectly conducting. When at least one reference is given then a conducting layer is not included unless it is specified as one of the layers. When several references are given, the sequence of the *Electrical Properties* are sorted based on the Displacement attribute of each *Electrical Properties* object.

Command Types

The commands available for exporting the geometry of a *Rectangular Plate* to a CAD file are described in *Scatterer*.

Remarks

Analysis methods

The scattering from a *Rectangular Plate* may be determined by *PO Analysis* (with *PO*, *Single-Face Scatterer*), *MoM* and *GTD Analysis*.

A *Rectangular Plate* with materials can, however, only be analysed with *PO*, *Single-Face Scatterer* (without PTD) and *GTD Analysis*.

Geometry

The direction of the normal vector to the rectangular plate is defined by the right-hand rule applied to the points in the order Corner 1, Corner 2, and Opp Point.

When the *Rectangular Plate* is part of a larger structure it is suggested to specify the points so the normals of all elements of the structure point out of the structure such that the overall direction of the normal does not change from element to element.

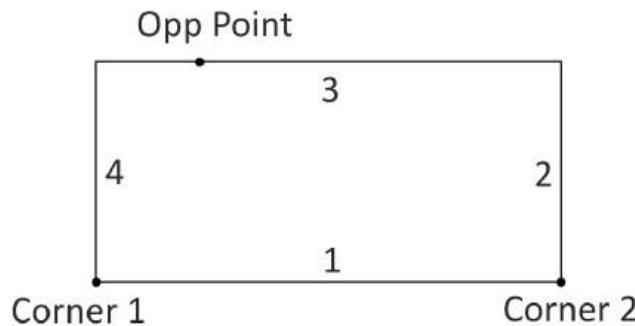


Figure 1

A rectangular plate seen from the side of the positive normal.

The edges of the rectangular plate are numbered as shown in Figure 1. This numbering is important when PTD is employed in the field analysis, since it is possible to specify different sampling densities along the individual edges (see the class *PO*, *Single-Face Scatterer*).

For the purpose to calculate a field across the surface of the rectangular plate, a local xy -coordinate system is defined in the plane of the rectangular plate. The origin of this local coordinate system is located at Corner 1 and the x -axis is oriented from Corner 1 to Corner 2. The y -axis is perpendicular to the x -axis and oriented such that Opp Point has a positive y -coordinate. The z -axis is then parallel to the normal of the rectangular plate.

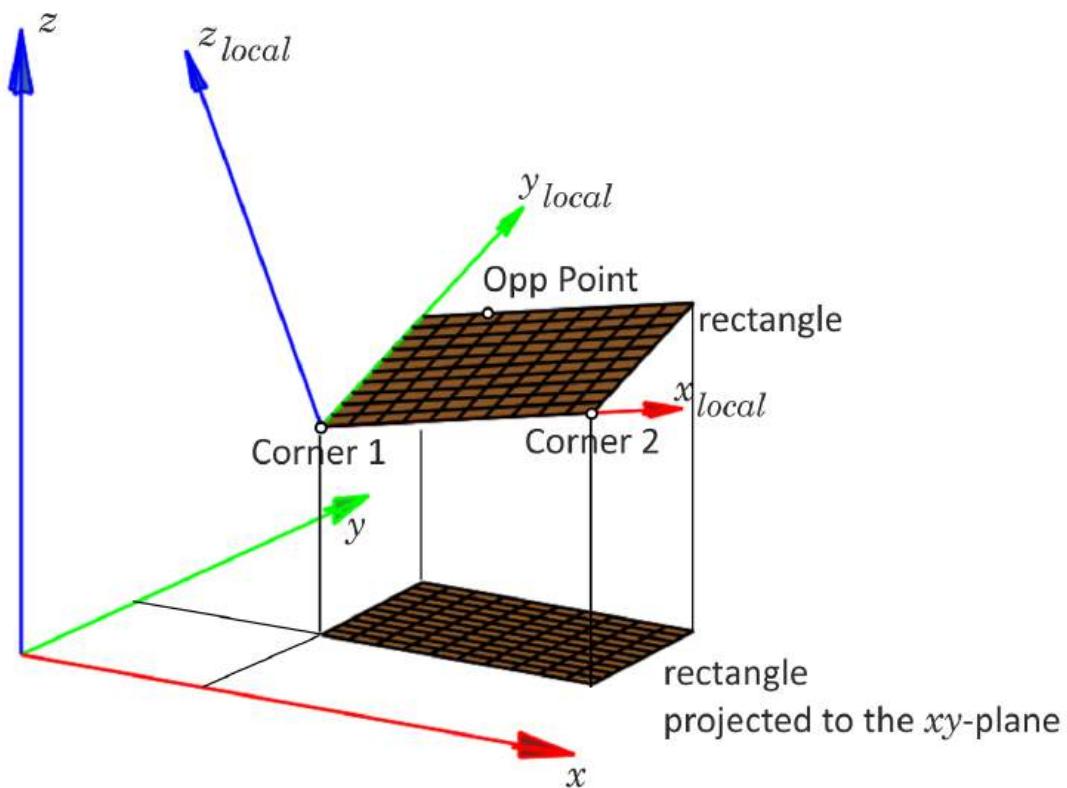


Figure 2

The rectangular plate is defined in the (x, y, z) -coordinates according to the attribute Coordinate System. The projection of the rectangular plate in the xy -plane is also shown. For illustration purposes, a local coordinate system, $(x_{local}, y_{local}, z_{local})$, is shown with the rectangular plate in the (x_{local}, y_{local}) -plane. The normal of the plate is co-parallel to the z_{local} -axis.

PARALLELOGRAM (parallelogram)

Purpose

The class *Parallelogram* is used to define a *Scatterer*, which is a plane with a rim like a parallelogram.

The scattering from a *Parallelogram* may be determined by *PO Analysis* (with *PO*, *Single-Face Scatterer*), *MoM* and *GTD Analysis* (except for plates with materials, see the remarks for details).

Links

Classes→*Geometrical Objects*→*Scatterer*→*Plate*→*Parallelogram*

Remarks

Syntax

```
<object name> parallelogram
(
    coor_sys           : ref(<n>),
    corner             : struct(x:<rl>, y:<rl>, z:<rl>),
    vec_1              : struct(x:<rl>, y:<rl>, z:<rl>),
    vec_2              : struct(x:<rl>, y:<rl>, z:<rl>),
    el_prop            : sequence(ref(<n>), ...)
)
where
<n> = name of an object
<rl> = real number with unit of length
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the parallelogram is specified.

Corner (*corner*) [struct].

Defines the coordinates of one corner of the parallelogram. The corner is also origin in a local coordinate system.

x (x) [real number with unit of length].

x-coordinate of the corner.

y (y) [real number with unit of length].

y-coordinate of the corner.

z (z) [real number with unit of length].

z-coordinate of the corner.

Vec 1 (*vec_1*) [struct].

Vector along an edge (denoted edge 1) of the parallelogram, with origin in *Corner*. The length of the vector equals the length of the edge.

- x (x) [real number with unit of length].
x-coordinate of vector along edge 1.
- y (y) [real number with unit of length].
y-coordinate of vector along edge 1.
- z (z) [real number with unit of length].
z-coordinate of vector along edge 1.

Vec 2 (vec_2) [struct].

Vector along the other edge (denoted edge 4 in the figure below) of the parallelogram, with origin in the *corner* defined above. The length of the vector equals the length of the edge.

- x (x) [real number with unit of length].
x-coordinate of vector along edge 4.
- y (y) [real number with unit of length].
y-coordinate of vector along edge 4.
- z (z) [real number with unit of length].
z-coordinate of vector along edge 4.

Electrical Properties (*el_prop*) [sequence of names of other objects], default: **blank**.

Sequence of one or more references to objects of the class *Electrical Properties*. Each object defines electrical reflection and transmission properties of a layer of surface material.

When no references are given the parallelogram will be assumed a perfectly conducting plate. When at least one reference is given then a conducting layer is not included unless it is specified as one of the layers. When several references are given, the sequence of the *Electrical Properties* are sorted based on the *Displacement* attribute of each *Electrical Properties* object.

Command Types

The commands available for exporting the geometry of a *Parallelogram* to a CAD file are described in *Scatterer*.

Remarks

Analysis methods

The scattering from a *Parallelogram* may be determined by *PO Analysis* (with *PO*, *Single-Face Scatterer*), *MoM* and *GTD Analysis*.

A *Parallelogram* with materials can, however, only be analysed with *PO*, *Single-Face Scatterer* (without PTD) and *GTD Analysis*.

Geometry

The direction of the normal vector to the parallelogram is defined by the vector product $\vec{V}_1 \times \vec{V}_2$ where \vec{V}_1 denotes Vec 1 and \vec{V}_2 denotes Vec 2. The corner and the edge vectors are illustrated in Figure 1.

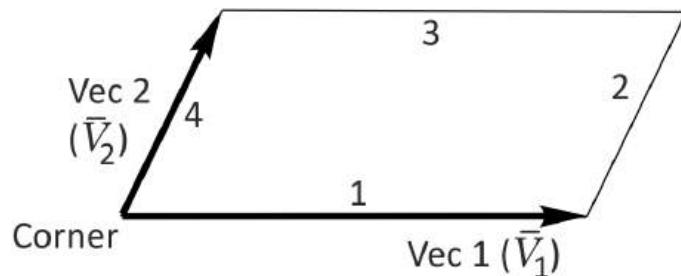


Figure 1 A parallelogram seen from the side of the positive normal.

When the **Parallelogram** is part of a larger structure it is suggested to specify Corner so the normals of all elements of the structure point out of the structure such that the overall direction of the normal does not change from element to element.

The edges of the parallelogram are numbered as shown in Figure 1. This numbering is important when PTD is employed in the field analysis, since it is possible to specify different sampling densities along the individual edges (see the class **PO, Single-Face Scatterer**).

For the purpose to calculate a field across the surface of the parallelogram, a local *xy*-coordinate system is defined in the plane of the parallelogram. The origin of this local coordinate system is located at Corner and the *x*-axis is oriented along Vec 1. The *y*-axis is perpendicular to the *x*-axis and oriented such that the projection of Vec 2 onto the *y*-axis is positive. The *z*-axis is then parallel to the (positive) normal of the parallelogram.

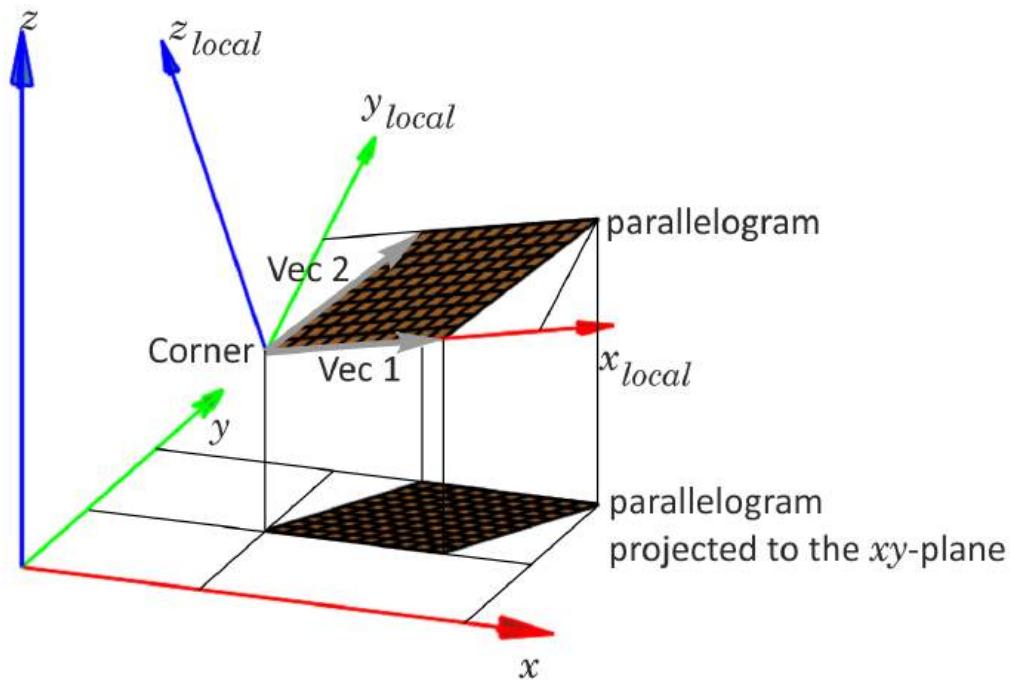


Figure 2

The parallelogram is defined in the (x, y, z) -coordinates according to the attribute Coordinate System. The parallelogram is spanned by the vectors Vec 1 and Vec 2 , shown in grey. The projection of the parallelogram in the xy -plane is also shown. For illustration purposes, a local coordinate system, $(x_{local}, y_{local}, z_{local})$, is shown with the parallelogram in the (x_{local}, y_{local}) -plane. The normal of the parallelogram is co-parallel to the z_{local} -axis.

CIRCULAR PLATE (circular_plate)

Purpose

The class *Circular Plate* is used to define a *Scatterer*, which is a plane with a circular rim.

The scattering from a *Circular Plate* may be determined by *PO Analysis* (e.g. *PO*, *Single-Face Scatterer*), *MoM*, and *GTD Analysis* (except for plates with materials, see the remarks for details).

Links

Classes→*Geometrical Objects*→*Scatterer*→*Plate*→*Circular Plate*

Remarks

Syntax

```
<object name> circular_plate
(
    coor_sys           : ref(<n>),
    radius             : <rl>,
    centre              : struct(x:<rl>, y:<rl>, z:<rl>),
    el_prop            : sequence(ref(<n>), ...)

)
where
<n> = name of an object
<rl> = real number with unit of length
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the circular plate is specified. The plate will be parallel to the *xy*-plane of this coordinate system and the *z*-axis will be the positive normal to the plate.

Radius (*radius*) [real number with unit of length].

Radius of the circular plate

Centre (*centre*) [struct].

Coordinates of the centre of the circular plate.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of the centre.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of the centre.

z (*z*) [real number with unit of length], default: **0**.

z-coordinate of the centre.

Electrical Properties (*el_prop*) [sequence of names of other objects], default: **blank**.

Sequence of one or more references to objects of the class *Electrical Properties*. Each object defines electrical reflection and transmission properties of a layer of surface material.

When no references are given the plate will be assumed perfectly conducting. When at least one reference is given then a conducting layer is not included unless it is specified as one of the layers. When several references are given, the sequence of the *Electrical Properties* are sorted based on the Displacement attribute of each *Electrical Properties* object.

Command Types

The commands available for exporting the geometry of a *Circular Plate* to a CAD file are described in *Scatterer*.

Remarks

When the *Circular Plate* is part of a larger structure it is suggested to specify the Coordinate System so the normals (for the *Circular Plate*: the z -axis) of all elements of the structure point out of the structure such that the overall direction of the normal does not change from element to element.

The scattering from a *Circular Plate* may be determined by *PO Analysis* (e.g. *PO*, *Single-Face Scatterer*), *MoM*, and *GTD Analysis*.

A *Circular Plate* with materials can, however, only be analysed with *PO Analysis* (without PTD) and *GTD Analysis*.

STRUTS

Purpose

In the menu *Struts* it is possible to describe two types of struts:

Circular Struts and

Polygonal Struts

Command Types

The commands available for exporting the geometry of a *Struts* to a CAD file are described in *Scatterer*.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Struts*

CIRCULAR STRUTS (circular_struts)

Purpose

The class *Circular Struts* is used to define a set of all perfectly conducting or all dielectric struts with circular cross-section, as for example used in circular-symmetric dual reflector systems to support the subreflector.

The scattering from *Circular Struts*, if the *Circular Struts* are all perfectly conducting, may be determined by *Strut Analysis*, *Circular Cross Section*, *Strut Analysis*, *Arbitrary Cross Section*, *GTD Circular Struts*, and *MoM*. If the *Circular Struts* are all dielectric, only *Strut Analysis*, *Arbitrary Cross Section* and *MoM* can be applied.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Struts*→*Circular Struts*

Remarks

Syntax

```
<object name> circular_struts
(
    coor_sys           : ref(<n>),
    radius             : <rl>,
    end_points         : sequence(
        struct(
            point1_x:<rl>,
            point1_y:<rl>,
            point1_z:<rl>,
            point2_x:<rl>,
            point2_y:<rl>,
            point2_z:<rl>),
        ...),
    material_parameters : struct(material:<si>,
                                dielectric_constant:<r>,
                                loss_tangent:<r>)
)
where
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the location of the struts is specified. The default is the global coordinate system.

Radius (*radius*) [real number with unit of length].

Radius of the circular strut cross-section.

End Points (*end_points*) [sequence of structs].

Definition of the end-points of each strut, one struct for each strut.

Point1 x (*point1_x*) [real number with unit of length].

x-coordinate of one end of one strut.

Point1 y (*point1_y*) [real number with unit of length].

y-coordinate of one end of one strut.

Point1 z (*point1_z*) [real number with unit of length].

z-coordinate of one end of one strut.

Point2 x (*point2_x*) [real number with unit of length].

x-coordinate of the other end of the same strut.

Point2 y (*point2_y*) [real number with unit of length].

y-coordinate of the other end of the same strut.

Point2 z (*point2_z*) [real number with unit of length].

z-coordinate of the other end of the same strut.

Material Parameters (*material_parameters*) [struct].

Material parameters for the circular struts.

Material (*material*) [item from a list of character strings], default: **PEC**.

Specifies whether the circular strut is a PEC or dielectric strut. If PEC, the values for Dielectric Constant and Loss Tangent will be ignored.

PEC

The strut is a PEC strut.

Dielectric

The strut is a dielectric strut with relative permittivity as specified in Dielectric Constant.

Dielectric Constant (*dielectric_constant*) [real number], default: **1**.

The relative permittivity of the strut. If the strut is a PEC, this attribute will be ignored.

Loss Tangent (*loss_tangent*) [real number], default: **0**.

The loss tangent of the strut. If the strut is a PEC, this attribute will be ignored.

Command Types

The commands available for exporting the geometry of a *Circular Struts* to a CAD file are described in *Scatterer*.

Remarks

Strut end-points

Note that the strut end-points are members of a sequence, so that several struts (with the same radius) can be defined by one and the same object.

When specifying the end points of the struts it should be observed that a strut periphery does not touch (or penetrate) the reflector surface as this may cause difficulties in the field computations.

The end-faces of the struts are not included in the analysis.

Dielectric struts

If a dielectric strut is illuminated by a source at close proximity *Strut Analysis, Arbitrary Cross Section* may lead to inaccurate results. To obtain an accurate result in such cases it is advised to represent the source by a *Plane Wave Expansion*.

POLYGONAL STRUTS (polygonal_struts)

Purpose

The class *Polygonal Struts* specifies struts with a polygonal cross-section. The *Polygonal Struts* can be all perfectly conducting or all dielectric. Several struts with identical cross-section may be specified within a single object.

The scattering from *Polygonal Struts*, if the *Polygonal Struts* is all perfectly conducting, may be determined by *Strut Analysis, Arbitrary Cross Section*, *PO Analysis* (using a *PO, Polygonal Struts (Obsolete)* or *PO, Multi-Face Scatterer* object), and *MoM*. If the *Polygonal Struts* are all dielectric only *Strut Analysis, Arbitrary Cross Section* and *MoM* can be used.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Struts*→*Polygonal Struts*

Remarks

Syntax

```
<object name> polygonal_struts
(
    position : sequence(
        struct(
            coor_sys:ref(<n>),
            z1:<rl>,
            z2:<rl>),
            ...),
    cross_section : sequence(
        struct(x:<rl>,
            y:<rl>),
            ...),
    material_parameters : struct(material:<si>,
        dielectric_constant:<r>,
        loss_tangent:<r>)
)
where
```

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

<si> = item from a list of character strings

Attributes

Position (*position*) [sequence of structs].

Definition of the orientation and the end-points of each strut, one struct for each strut.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the strut cross-section is defined. The length axis of the strut is parallel to the *z*-axis of this coordinate system.

Z1 (z1) [real number with unit of length].

z-coordinate of one end-point of the strut.

Z2 (z2) [real number with unit of length].

z-coordinate of the other end-point of the strut.

Cross Section (*cross_section*) [sequence of structs].

Definition of the polygonal cross-section of the struts (identical for all struts). The cross-sectional polygon is defined by specifying the coordinates of the polygon corners in a local *xy*-plane perpendicular to the axis of the strut. The coordinates must be given in succession (either clockwise or anti-clockwise) and the corners must specify a polygon (which does not need to encircle the origin of the *xy*-plane).

x (x) [real number with unit of length].

x-coordinate of a corner.

y (y) [real number with unit of length].

y-coordinate of the same corner.

Material Parameters (*material_parameters*) [struct].

Material parameters for the polygonal struts.

Material (*material*) [item from a list of character strings], default: **PEC**.

Specifies whether the polygonal strut is a PEC or dielectric strut.

If PEC, the values for Dielectric Constant and Loss Tangent will be ignored.

PEC

The strut is a PEC strut.

Dielectric

The strut is a dielectric strut with relative permittivity as specified in Dielectric Constant.

Dielectric Constant (*dielectric_constant*) [real number], default: **1**.

The relative permittivity of the strut. If the strut is a PEC, this attribute will be ignored.

Loss Tangent (*loss_tangent*) [real number], default: **0**.

The loss tangent of the strut. If the strut is a PEC, this attribute will be ignored.

Command Types

The commands available for exporting the geometry of a *Polygonal Struts* to a CAD file are described in *Scatterer*.

Remarks

Strut end-points

When specifying the end points of the struts it should be observed that a strut periphery does not touch (or penetrate) the reflector surface as this may cause difficulties in the field computations.

The end-faces of the struts are not included in the analysis.

Dielectric struts

If a dielectric strut is illuminated by a source at close proximity *Strut Analysis, Arbitrary Cross Section* may lead to inaccurate results. To obtain an accurate result in such cases it is advised to represent the source by a *Plane Wave Expansion*.

BOX (box)

Purpose

The class **Box** is used to define a *Scatterer* consisting of a perfectly electrically conducting box with planar faces. The scattering from a **Box** may be determined by using *PO*, *Multi-Face Scatterer* and *MoM*.

Links

Classes→*Geometrical Objects*→*Scatterer*→**Box**

Remarks

Syntax

```
<object name> box
(
    coor_sys           : ref(<n>),
    x_length          : <rl>,
    y_length          : <rl>,
    z_length          : <rl>,
    centre            : struct(x:<rl>, y:<rl>, z:<rl>),
    exclude_faces     : sequence(<si>, ...)
)
where
<n> = name of an object
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the box is specified. The edges of the box will be parallel to the axes of this coordinate system. The coordinate system shall be chosen with axes parallel to the edges of the box.

x Length (*x_length*) [real number with unit of length].

The length of the box edges parallel to the *x*-axis.

y Length (*y_length*) [real number with unit of length].

The length of the box edges parallel to the *y*-axis.

z Length (*z_length*) [real number with unit of length].

The length of the box edges parallel to the *z*-axis.

Centre (*centre*) [struct].

The coordinates of the centre of the box.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of the centre.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of the centre.

z (*z*) [real number with unit of length], default: **0**.

z-coordinate of the centre.

Exclude Faces (*exclude_faces*) [sequence of items from a list of character strings].

When using **MoM**: a sequence of faces may be excluded from the modelling of the box. The sequence may be empty in which case the attribute shall not to be specified. See the Remarks section for how the faces to be excluded shall be specified.

Remarks

Analysis methods

The scattering from a **Box** with all six faces may be determined by using **PO**, **Multi-Face Scatterer** (currents will only be determined on the illuminated faces) and **MoM**. If any faces are excluded (using the **Exclude Faces** attribute), the scattering can only be performed using **MoM**.

Geometry

The six faces and 12 edges of the box are numbered as shown in Figure 1(c). Figure 1(b) indicates the names of the faces, as used in the **Exclude Faces** attribute.

This is summarized in the following table which also gives the unit vectors of the outward normal to each of the six faces:

face in figure	name of face	outward normal
face 1	z_max	(0, 0, 1)
face 2	x_max	(1, 0, 0)
face 3	y_max	(0, 1, 0)
face 4	x_min	(-1, 0, 0)
face 5	y_min	(0, -1, 0)
face 6	z_min	(0, 0, -1)

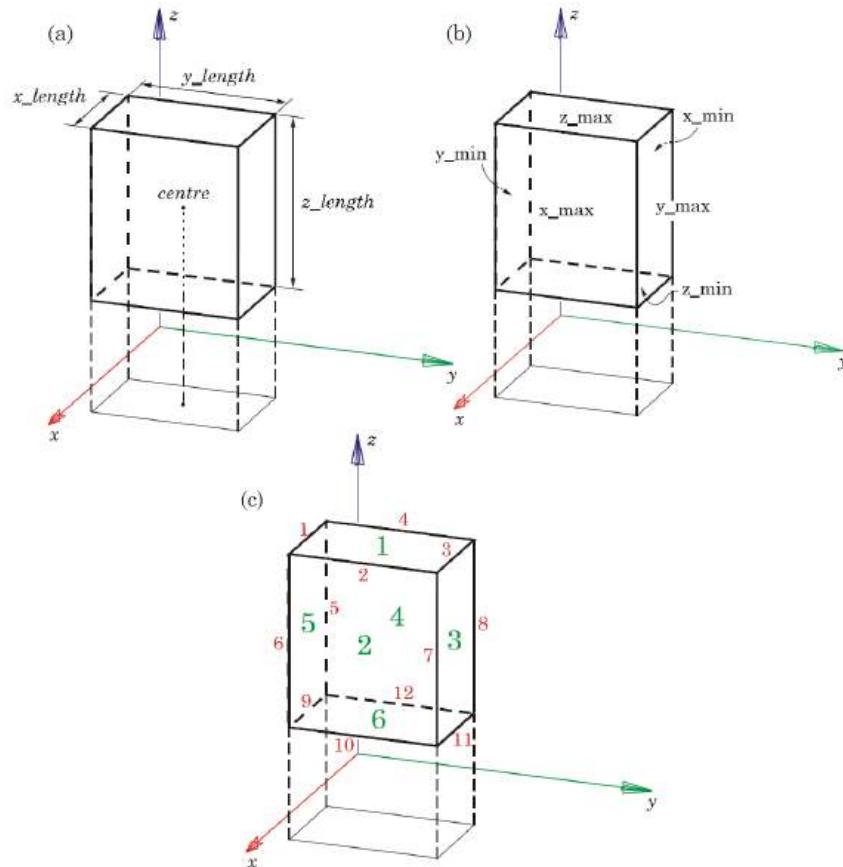


Figure 1

(a) The **Box** is defined by its Centre and the edge lengths x Length, y Length and z Length. The projection of the **Box** on the xy -plane is also shown. (b) The faces are named according to their position relatively to the coordinates. The faces x_min , y_min and z_min are on the backside of the box in this view. The names of the faces are used in the Exclude Faces attribute. (c) The numbering of the faces (in green) and the edges (in red). These numbers are used for identification of faces and edges in *PO, Multi-Face Scatterer*.

SCATTERER CLUSTER (scatterer_cluster)

Purpose

An object of the class *Scatterer Cluster* defines a group of scatterers. This allows a complex scatterer to be defined in terms of a number of more simple scatterers (including other *Scatterer Clusters*). This in turn simplifies the specification of geometries and analyses, thus the *Scatterer Cluster* objects can be referred to from a *PO Analysis* object (*PO*, *Multi-Face Scatterer*) or a *MoM* object and the currents on the *Scatterer Cluster* can be computed by a single *Get Currents* command.

If the currents are computed using a *MoM* object the mutual coupling and the shadow effects between the individual scatterers in a *Scatterer Cluster* can be included. These effects are not included if *MoM* is used on each scatterer separately.

A *Scatterer Cluster* can be analysed by MoM or PO but only if each of the individual scatterers can be analysed by the same method.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Scatterer Cluster*

Remarks

Syntax

```
<object name> scatterer_cluster
(
    scatterers : sequence(ref(<n>), ...)
)
where
<n> = name of an object
```

Attributes

Scatterers (*scatterers*) [sequence of names of other objects].

Sequence of references to objects of one of the *Scatterer* classes.

Remarks

This section contains discussions on the following topics

- Type of scatterers
- Plotting
- Numbering schemes
- MoM analysis on clusters
- PO analysis on clusters

Type of scatterers

Objects of the *Scatterer Cluster* class can be used to group all types of scatterers, including other *Scatterer Clusters*, which allow a complex scatterer

to be defined in a hierarchical way. As an example, a satellite can be defined by grouping a satellite body and a solar panel together in a *Scatterer Cluster*. The satellite body and the solar panel could be individual *Scatterers* or new *Scatterer Clusters*.

Plotting

Note that there is no plot class related to a *Scatterer Cluster*. Instead, the individual scatterers in the cluster are plotted by the relevant individual plot objects.

Numbering schemes

When *Scatterers* are combined to a *Scatterer Cluster* then the number of faces in the *Scatterer Cluster* is the sum of the number of faces on the individual scatterers. The faces are numbered consecutively, e.g., if the first scatterer in the sequence has two faces they will be numbered 1 and 2. The first face on the next scatterer is then face number 3, the second face is face number 4, and so on. The same numbering scheme is used for edges and gaps.

MoM analysis on clusters

The currents on a *Scatterer Cluster* may be obtained using MoM which includes all physical effects such as shadowing and mutual coupling.

The scatterers in a cluster are modelled as being connected, i.e. in physical contact, if proper connectivity rules are fulfilled patch-by-patch. The rules are listed in the description of the *MoM* class. For example, two plates with straight edges are connected if the end-points of two edges conform in pairs within $10^{-5}\lambda$. It is up to the user to ensure proper connectivity of the patches which can be checked using the *MoM Plot* class.

PO analysis on clusters

The currents on a *Scatterer Cluster* may also be obtained using PO (*PO*, *Multi-Face Scatterer*). However, the analysis should be used with caution since shadow and coupling effects are neglected. Thus, if one part of the *Scatterer Cluster* creates a shadow of the incident field on another part of the cluster, this effect is not included. Similarly, PTD should be used with caution since two connected patches which form a wedge will be analyzed as two individual knife-edges. It is up to the user to ensure that PO/PTD is applied in a correct manner.

LOAD (load)

Purpose

The class **Load** is used to terminate a beam path in the Frame Design Tool (see the corresponding manual in the documents directory). If rays are displayed in a geometry plot it also terminates those rays that hits the load. It has no effect on the electromagnetic computations.

Links

[Classes](#)→[Geometrical Objects](#)→[Scatterer](#)→**Load**

Remarks

Syntax

```
<object name> load
(
    coor_sys           : ref(<n>),
    load_size_x        : <rl>,
    load_size_y        : <rl>,
    obsolete_plot_status : <si>
)
```

where

<n> = name of an object

<rl> = real number with unit of length

<si> = item from a list of character strings

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a [Coordinate Systems](#) object in which the load is specified. The plane of the load is the *xy*-plane of the referred coordinate system.

Load Size x (*load_size_x*) [real number with unit of length], default: **0**.

Side length of load in the *x*-direction.

Load Size y (*load_size_y*) [real number with unit of length], default: **0**.

Side length of load in the *y*-direction.

Plot Status (Obsolete) (*obsolete_plot_status*) [item from a list of character strings], default: **on**.

Obsolete attribute maintained for backward compatibility. The setting will be ignored.

on

The load is displayed in the plot.

off

The load is not displayed in the plot.

Remarks

Analysis methods

The load has the shape of a rectangle in the xy -plane of the specified coordinate system within

$$\begin{aligned} -\text{load_size_x}/2 &\leq x \leq \text{load_size_x}/2 , \text{ and} \\ -\text{load_size_y}/2 &\leq y \leq \text{load_size_y}/2 \end{aligned}$$

The load terminates beam paths and rays incoming from either side.

TABULATED MESH (tabulated_mesh)

Purpose

The class ***Tabulated Mesh*** provides a convenient way to define an arbitrary scatterer that may be metallic, dielectric, or composite metallic/dielectric. The ***Tabulated Mesh*** class may be used whenever the standard scatterer classes in GRASP are too restrictive and do not allow a specific scatterer to be defined. The ***Tabulated Mesh*** is also useful for importing data from various CAD meshing tools, e.g. MSC Patran, or other programs based on meshed geometries.

The ***Tabulated Mesh*** scatterer is specified by the (x, y, z) -components of a mesh defining the surfaces of one or more homogeneous regions of which the scatterer is composed. The ***Tabulated Mesh*** may also contain metallic wires.

The scattering from a ***Tabulated Mesh*** scatterer may be determined by MoM using the ***MoM*** class. If the scatterer is a perfect electric conductor, and no wires are present, the scattering may also be determined by PO using the ***PO, Multi-Face Scatterer*** class.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Geometrical Objects](#)→[Scatterer](#)→[Tabulated Mesh](#)

[Remarks](#)

Syntax

```
<object name> tabulated_mesh
(
    coor_sys           : ref(<n>),
    file_name          : <f>,
    unit               : <si>,
    external_command   : ref(<n>)
)
where
<n> = name of an object
<f> = file name
<si> = item from a list of character strings
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the ***Coordinate Systems*** classes, defining the coordinate system in which the (x, y, z) -coordinates are given.

File Name (*file_name*) [file name].

Name of the file containing the tabulated mesh. The recommended file extension is `.msh`, and the required file format is described in the section [Meshed Geometries](#).

Unit (unit) [item from a list of character strings], default: **m**.

Defines the length unit applied in the mesh file for (x, y, z) -coordinates and wire radii (mm, cm, m, km, in, ft).

External Mesh Command (external_command) [name of an object], default: **blank**.

Reference to an object of class [External Command](#). If the command specified by this reference is executed, e.g. if a variable is changed, the tabulated mesh file will be read again. This can be used to create a mesh plugin that updates a complex geometry seamlessly whenever the input changes.

Remarks

This section contains discussions on the following topics:

- Available analysis methods
- Definitions
- Patches
- Wire segments
- Mesh rules
- Mesh accuracy

Available analysis methods

A [Tabulated Mesh](#) scatterer can be analyzed using MoM or PO. The MoM provides a very accurate analysis including shadow and internal coupling effects. The PO can only be applied to perfectly electrically conducting scatterers, and may further yield incorrect results as the shadow and coupling effects are neglected. The scatterer can not be analyzed by GO/GTD.

Definitions

The [Tabulated Mesh](#) object defines a scatterer in terms of a user-defined mesh. The mesh consists of a number of patches and wires segments each defined by a set of nodes.

If the patches are arranged to form a closed volume, this volume is referred to as a region. A region can be assigned material properties (the relative permittivity, the relative permeability and the loss tangent) modelling a volume filled with a homogeneous material with these properties,, e.g. a dielectric. Two closed volumes with the same material properties must be considered as different regions.

Patches

The patches can be flat or curved quadrilaterals or they can be flat or curved triangles. However, the use of triangles is discouraged as the computations

become inefficient. The patches are defined by a number of nodes as described below. The surface of a patch is modelled by polynomials, up to fourth order, through the specified nodes and it is upon this surface the currents may be determined.

A quadrilateral patch can be defined by 4, 9, 16 or 25 nodes resulting in a bilinear, biquadratic, bicubic or 4th order quadrilateral, respectively. A triangular patch can be defined by 3 or 7 nodes resulting in a flat or curved triangle, respectively. The different types of patches are shown in Figure 1.

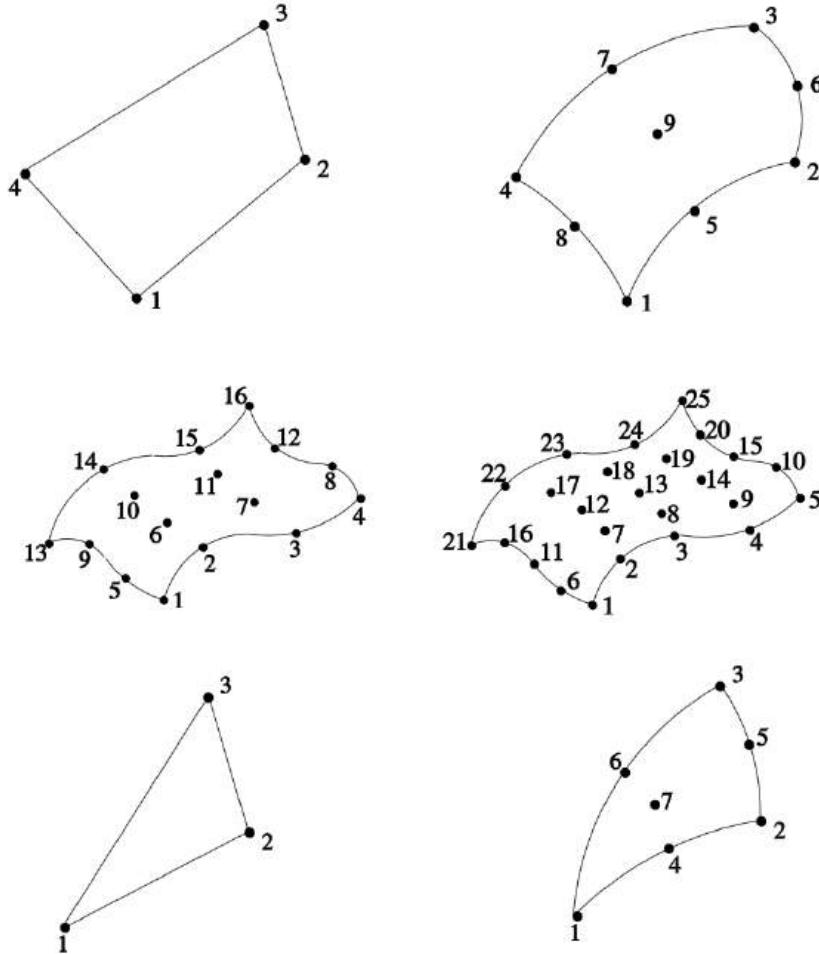


Figure 1 Different types of patches. The quadrilateral patches are defined by 4, 9, 16 or 25 nodes and for triangular patches by 3 or 7 nodes.

Small patches defined by many nodes will in general give the best description of the geometry of a scatterer. On the other hand, the patches should be as large as possible to minimise the number of unknowns in the MoM, and consequently the memory requirement and computation times. Patches larger than 2 wavelengths will internally be subdivided to smaller patches as needed for the MoM computations. Hence, the number of patches used in the mesh should be a trade-off between the modelling accuracy and the resulting number of unknowns.

All patches and nodes are specified in the file referred to in the attribute `file_name`. The required file format as well as an illustrative example are

presented in the section describing the required file format for *Meshed Geometries*.

Wire segments

Wires may be modelled by wire segments which may be straight or curved. A wire segment is defined by a number of nodes through which the segments pass. The shape of a segment between the nodes is modelled by polynomials. When the wire segment is defined by 2, 3 or 4 nodes the polynomial is linear, quadratic or cubic, respectively. The different types of wire segments are shown in Figure 2.

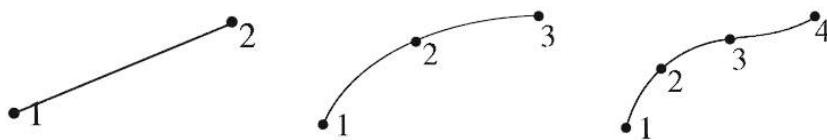


Figure 2 Examples of wire segments defined by 2, 3 or 4 nodes.

As for patches, the best description of the geometry of a wire is by small segments defined by many nodes. On the other hand, the segments should be as large as possible - but not larger than 2 wavelengths - to minimise the number of unknowns in the MoM, and consequently the memory requirement and computation times. Hence, the number of segments used in the mesh should be a trade-off between the modelling accuracy and the resulting number of unknowns.

The wire segments are also specified in the file referred to in the attribute *file_name*. The required file format as well as an illustrative example are presented in the section describing the required file format for *Meshed Geometries*.

Mesh rules

The best connection between patches is obtained by having the same nodes defining the common edge of the patches. The same applies for wire segments, for which it is simple to illustrate: When one segment ends at a given node, say node 127, then the next segment should start at the same node, node 127. This is, however, not mandatory, the next segment may start at, say, node 539, but the position of node 539 should then be the same as the position of node 127 in order to ensure connectivity between the wire segments, cf. the following rules 1 and 5.

The following rules apply for a MoM mesh:

1. Two neighbouring patches are considered connected if an edge on one patch is identical to an edge on the other patch within a tolerance of $10^{-5}\lambda$, λ being the wavelength. This implies that the two edges have the same number of nodes and that the nodes coincide in pairs.

If the rule is fulfilled the edge is modelled as a single shared edge in the MoM computations and the currents are allowed to flow from one patch to the neighbour patch

If the rule is not fulfilled, the edges will be regarded as external edges and the currents are not allowed to flow between the patches.

The *MoM Plot* class can be used to check if edges are considered as connected or external edges.

2. An edge can be shared by no more than 4 patches. An error is issued during the MoM computation if this condition is violated.
3. Dielectric regions must be closed. An error is issued during the MoM computation if this condition is violated.
4. A closed metallic region should be assigned to be region number -1 if it is in contact with a dielectric region. If the region number is not assigned to -1 a significant overhead is introduced in the computations since unnecessary basis functions are used to model the zero currents on the internal side of the closed metallic region.
If the closed metallic region is not in contact with a dielectric region, there is no such overhead in the computations, and the region number -1 needs not to be specified.
5. Two wires are considered connected if a node on one wire is identical to a node on the other wire within a tolerance of $10^{-5}\lambda$. When two wires are connected, the wire current will flow continuously from one wire to the other.
6. If the end of a wire is touching a patch, a wire-plate junction will automatically be inserted and Kirchhoff's laws will be satisfied at the junction.

Mesh accuracy

The surface of a patch is modelled by polynomials through the user-specified nodes. In MoM and PO computations, points on this surface are used. It is up to the user to ensure that the modelled polynomial surface follows the true surface with a sufficient accuracy. That is, a sufficiently fine mesh must be provided as input to the *Tabulated Mesh* class.

WIRES

Purpose

In the menu *Wires* it is possible to describe two types of wires:

Piecewise Straight Wire and

Curved Wire

Links

Classes→*Geometrical Objects*→*Scatterer*→*Wires*

PIECEWISE STRAIGHT WIRE (piecewise_straight_wire)

Purpose

The class *Piecewise Straight Wire* is used to define connected pieces of straight wires of constant radius. The wires are perfectly conducting and the currents on the wires can be computed by *MoM*.

If a *Piecewise Straight Wire* is included in a *Scatterer Cluster*, GRASP-MoM will automatically search for wire-plate junctions at the first and the last node of the *Piecewise Straight Wire* definition. Therefore, wire-plate junctions are automatically introduced if one or both of these end nodes are touching a patch originating from another scatterer in the defined cluster. The optimal location of a wire-plate junction is near the centre of the patch or near the centre point of one of the patch edges.

It is possible to place a *Voltage Generator* at an internal node, between the ends of two wires or at a wire-plate junction.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Geometrical Objects](#)→[Scatterer](#)→[Wires](#)→[Piecewise Straight Wire](#)

Remarks

Syntax

```
<object name> piecewise_straight_wire
(
    coor_sys           : ref(<n>),
    nodes              : sequence(
                            struct(x:<rl>,
                                   y:<rl>,
                                   z:<rl>),
                            ...),
    radius             : <rl>
)
where
<n> = name of an object
<rl> = real number with unit of length
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the location of the wire nodes are specified.

Nodes (*nodes*) [sequence of structs].

A sequence of nodes defining a piecewise straight curve.

x (x) [real number with unit of length].

x-coordinate of the nodes defining the wire.

y (y) [real number with unit of length].

y-coordinate of the nodes defining the wire.

z (z) [real number with unit of length].

z-coordinate of the nodes defining the wire.

Radius (*radius*) [real number with unit of length].

The constant radius of the wire. The wire radius must be positive (greater than zero).

Remarks

Analysis method

Wires can be analyzed using the method of moments, *MoM*. The pieces of straight wires between the defining nodes will automatically be meshed according to the *max_mesh_length* attribute in the *MoM* object. If the wire is a part of a *Scatterer Cluster*, it will automatically be determined if the wire is sharing one or more nodes with other wires within the cluster, or if an end point of the wire is touching a conducting surface within the cluster. In the latter case, a wire-plate junction is automatically formed.

The *MoM* analysis of the wire currents employs the exact Green's function, i.e., the thin-wire kernel is not applied. However, it is assumed that no azimuthally oriented currents are flowing on the wire and that there is no azimuthal variation of the currents flowing along the wire. Please consult the MoM add-on manual for details.

A *Voltage Generator* can be placed at an internal node in the *Piecewise Straight Wire* definition. In addition, a *Voltage Generator* can – if the *Piecewise Straight Wire* is a part of a *Scatterer Cluster* – be placed between an end node of the wire and an end node of another wire in the cluster or the conducting surface at a wire-plate junction.

An example of a circular loop modelled by *Piecewise Straight Wire* is shown in Figure 1. The loop is modelled as two objects of the *Piecewise Straight Wire*. The one object consists of ten straight segments defined by 11 nodes (clockwise from 10 o'clock to 8 o'clock) and has a *radius* of 0.05 of the loop radius. The other object has a larger radius of 0.08 of the loop radius and contains only two pieces (3 nodes) from 8 o'clock to 10 o'clock. The figure is for illustration only; a circular wire is usually better modelled by an object of class *Curved Wire*.

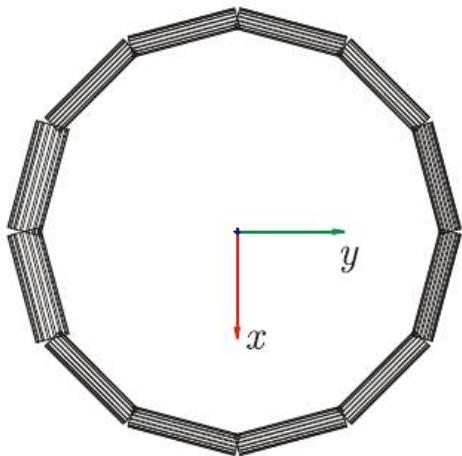


Figure 1

Circular loop in the xy -plane modelled as two objects of the *Piecewise Straight Wire*, the two objects have different *radius* of the wire segments. The nodes are situated at the breaks of the wire.

CURVED WIRE (curved_wire)

Purpose

The class *Curved Wire* is used to define a smooth wire passing through an arbitrary number of interpolation nodes. The curved wire is perfectly conducting and the current on the wire can be computed by *MoM*.

The wire radius can be non-constant and varies then linearly between the nodes defining the wire. The centre line of the curved wire will be described by a cubic spline curve passing through the nodes of the wire. GRASP-MoM will automatically define a MoM mesh for the wire using third-order curved wire pieces, which are in exact agreement with the spline curve.

If a *Curved Wire* is included in a *Scatterer Cluster*, GRASP-MoM will automatically search for wire-plate junctions at the first and the last node of the *Curved Wire* definition. Therefore, wire-plate junctions are automatically introduced if one or both of these end nodes are touching a patch originating from another scatterer in the defined cluster. The optimal location of a wire-plate junction is near the centre of the patch or near the centre point of one of the patch edges.

It is possible to place a *Voltage Generator* at an internal node in a curved wire definition, between the ends of two wires or at a wire-plate junction.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Geometrical Objects](#)→[Scatterer](#)→[Wires](#)→[Curved Wire](#)

Remarks

Syntax

```
<object name> curved_wire
(
    coor_sys           : ref(<n>),
    nodes              : sequence(
                            struct(x:<rl>,
                                    y:<rl>,
                                    z:<rl>,
                                    radius:<rl>),
                            ... )
)
where
<n> = name of an object
<rl> = real number with unit of length
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the nodes of the wire are specified.

Nodes (*nodes*) [sequence of structs].

A sequence of nodes defining a cubic spline curve which is the centre line of the wire, and a radius of the wire at the node.

x (*x*) [real number with unit of length].

x-coordinate of the node.

y (*y*) [real number with unit of length].

y-coordinate of the node.

z (*z*) [real number with unit of length].

z-coordinate of the node.

Radius (*radius*) [real number with unit of length].

The radius of the wire at the node. The radius must be positive.

Remarks

Wires can be analyzed using the method of moments, *MoM*. The curved wire will automatically be meshed using third-order wire segments and according to the `max_mesh_length` attribute in the *MoM* object. If the wire is a part of a *Scatterer Cluster*, it will automatically be determined if the wire is sharing one or more nodes with other wires within the cluster, or if an end point of the wire is touching a conducting surface within the cluster. In the latter case, a wire-plate junction is automatically formed.

The *MoM* analysis of the wire currents employs the exact Green's function, i.e., the thin-wire kernel is not applied. However, it is assumed that no azimuthally oriented currents are flowing on the wire and that there is no azimuthal variation of the currents flowing along the wire. Please consult the MoM add-on manual for details.

A *Voltage Generator* can be placed at an internal node in the *Curved Wire* definition. In addition, a *Voltage Generator* can - if the *Curved Wire* is a part of a *Scatterer Cluster* - be placed between an end node of the wire and an end node of another wire in the cluster or the conducting surface at a wire-plate junction.

An example of a circular loop modelled by *Curved Wire* is shown in the following figure. The loop is modelled by specifying 13 nodes (including identical start and end nodes). The wire radius is 0.05 of the loop radius at all nodes apart from the leftmost node which has radius 0.08 of the loop radius. This illustrates the linear interpolation of the wire radius from node to node.

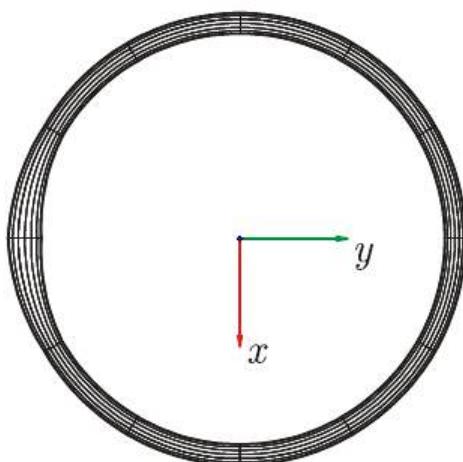


Figure 1

Circular loop in the xy -plane modelled as a *Curved Wire*. The loop is specified by 13 nodes of which the first and the last are identical. The nodes are placed at the centres of the radially oriented lines (which in fact are circles oriented perpendicular to the wire). The radius of the wire at 9 o'clock is specified larger than at the other nodes.

BODY OF REVOLUTION

Purpose

The menu *Body of Revolution* (BoR) is used to describe rotationally symmetric bodies. The rotationally symmetric bodies are traced by a plane curve (the generatrix) which is rotated around an axis positioned in the plane of the curve.

The rotational symmetry is used in *BoR-MoM* such that it is possible to reduce the computation time and to handle larger scatterers than in the usual *MoM*.

The different descriptions of the rotationally symmetric bodies are characterized by the description of the generatrix:

When the generatrix is a piecewise linear curve the scatterer may be described by a

Piecewise Linear Body of Revolution

A general body of revolution scatterer, metallic as well as dielectric, may be described by a

BoR Mesh

When the rotationally symmetric scatterer is a reflector it may be described from the menu

Circular Symmetric Reflector

Links

Classes→*Geometrical Objects*→*Scatterer*→*Body of Revolution*

PIECEWISE LINEAR BODY OF REVOLUTION (piecewise_linear_bor)

Purpose

The class *Piecewise Linear Body of Revolution* is used to define a scatterer consisting of a piecewise linear body of revolution. As the scatterer is rotationally symmetric it may be described by a curve (the generatrix) in a plane containing the axis of rotation. The full geometry of the scatterer is then obtained by rotating this curve around the axis. The curve is described by nodes (points) defining the linear segments of the curve. The scattering by a *Piecewise Linear Body of Revolution* may be determined by *BoR-MoM* and *MoM*.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Geometrical Objects](#)→[Scatterer](#)→[Body of Revolution](#)→[Piecewise Linear Body of Revolution](#)

Remarks

Syntax

```
<object name> piecewise_linear_bor
(
    nodes           : table(
        <r> <r>
        ...),
    coor_sys       : ref(<n>),
    length_unit   : <si>
)
where
<n> = name of an object
<r> = real number
<si> = item from a list of character strings
```

Attributes

Nodes (*nodes*) [table (2,2)].

Defines the piecewise linear curve as a sequence of (z, ρ) points.

z (*z*) [real number].

The *z*-coordinate of the point

rho (*rho*) [real number].

The ρ coordinate of the point. The value has to be equal or greater than 0.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

The piecewise linear body of revolution is defined in this coordinate system, by rotation of the piecewise linear curve around the z-axis of the coordinate system.

Length Unit (*length_unit*) [item from a list of character strings], default: **m**.

Sets the length unit of the coordinates in the Nodes attribute.

Remarks

Objects of the class *Piecewise Linear Body of Revolution* defines a scatterer which is a Body of Revolution (BOR) i.e. the scatterer is rotational symmetry around an axis, the axis of rotation. The surface of the scatterer may be described by a curve (or more curves), the generatrix, in a plane containing the axis of rotation. The surface is generated when this curve is rotated around the axis. It is thus sufficient to specify the curve in order to describe the scatterer. The curve is specified by a set of nodes which are connected by linear segments, linear from node to node. The ρ -values have to be equal or greater than 0.

Application example

In Figure 1, a circular plate is created using a *Piecewise Linear Body of Revolution* using two nodes:

```
nodes      : table
(
  0.0  0.0
  0.0  0.5
)
```

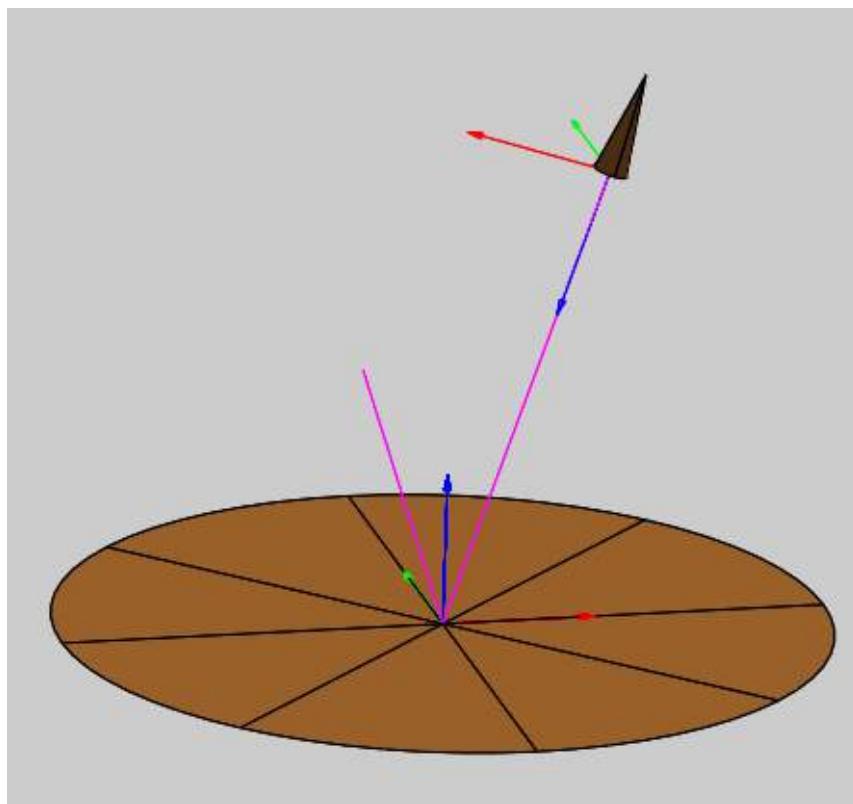


Figure 1

A circular plate created using a piecewise linear bor.

BOR MESH (bor_mesh)

Purpose

By the class *BoR Mesh* it is possible to describe a Body of Revolution (BoR) scatterer. The scatterer may be metallic, dielectric, or composite metallic/dielectric.

As the scatterer is rotationally symmetric it may be described by a curve (the generatrix) in a plane containing the axis of rotation. The full geometry of the scatterer is then obtained by rotating this curve around the axis. The curve is described by nodes (points) defining segments of the curve, either linear or cubic segments.

It is possible to define homogeneous dielectric regions as closed regions enclosed by segments of the curve.

The *BoR Mesh* class may be used to describe general scatterers, e.g. reflectors, as long as these have a rotational symmetry around the same axis.

The scattering by a *BoR Mesh* may be determined by *BoR-MoM* and *MoM*.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Geometrical Objects](#)→[Scatterer](#)→[Body of Revolution](#)→[BoR Mesh](#)

Remarks

Syntax

```

<object name> bor_mesh
(
    coor_sys : ref(<n>),
    regions : table(
        <i> <r> <r> <r>
        ...),
    nodes : table(
        <i> <r> <r>
        ...),
    linear_segments : table(
        <i> <i> <i> <i> <i> <r> <r>
        ...),
    cubic_segments : table(
        <i> <i> <i> <i> <i> <i> <i> <r> <r>
        ...),
    length_unit : <si>,
    coor_order : <si>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<si> = item from a list of character strings

```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

The body of revolution mesh is defined in this coordinate system, by rotation of the linear or cubic segments around the z-axis of the coordinate system.

Regions (*regions*) [table (*,4)].

Table with 4 columns specifying the characteristics of the dielectric region(s).

Region Number (*Region Number*) [integer].

A unique, positive number for the region, apart from a default region with the number 0 representing the free space and closed regions with number -1 representing one or more closed metallic region.

Permittivity (*Permittivity*) [real number].

The relative permittivity (dielectric constant) of the region.

Permeability (*Permeability*) [real number].

The relative permeability of the region.

Loss Tangent (*Loss Tangent*) [real number].

The loss tangent for the region.

Nodes (*nodes*) [table (2,3)].

Table with 3 columns specifying the nodes (the points) of the curve (the generatrix) which describes the scatterer. The table must define at least two nodes. The unit of the coordinates are determined by the attribute Length Unit. In addition, the ρ -values have to be equal or greater than 0.

Node Number (*Node Number*) [integer].

A unique, positive number of the node.

c1 (*c1*) [real number].

First coordinate for the node, according to the attribute Coordinate Order.

c2 (*c2*) [real number].

Second coordinate for the node, according to the attribute Coordinate Order.

Linear Segments (*/linear_segments*) [table (*,7)].

Specification of the linear segments in the specification of the generatrix. The table has 7 columns:

Segment Number (*Segment Number*) [integer].

A unique, positive number of the segment.

Node A (*Node A*) [integer].

The node at the first end of the segment. The segment will run linearly from node A to node B.

Node B (Node B) [integer].

The node at the last end of the segment.

Region 1 (Region 1) [integer].

The number of the region on the one side of the curve segment.

If Region 1 and Region 2 differ, the curve segment may either define a dielectric interface or a conducting surface, which is controlled by the real part of the entry Zs (see below). If Region 1 and Region 2 are identical, the curve segment must define a conducting surface, implying that Zs(real) must be non-negative.

Region 2 (Region 2) [integer].

The number of the region on the second side of the curve segment.

Zs (real) (Zs (real)) [real number].

The real part of Zs, in ohm/meter, Zs being the complex surface impedance of the surface defined by the curve segment. Zs(real) >= 0 specifies a conducting surface; Zs(real) = -1 specifies a dielectric interface and region 1 and 2 must be different.

The default value is Zs = 0 which specifies a perfect conductor.

Zs (imag) (Zs (imag)) [real number].

The imaginary part of Zs, in ohm/meter.

Cubic Segments (cubic_segments) [table (*,9)].

Specification of the third-order segments in the specification of the directrix. The table has 9 columns:

Segment Number (Segment Number) [integer].

A unique, positive number of the segment.

Node A (Node A) [integer].

The node at the first end of the segment. The segment will run from node A, through nodes B and C to node D.

Node B (Node B) [integer].

First intermediate node B of the segment.

Node C (Node C) [integer].

Second intermediate node C of the segment.

Node D (Node D) [integer].

The node at the last end of the segment.

Region 1 (Region 1) [integer].

The number of the region on the one side of the curve segment.

If Region 1 and Region 2 differ, the curve segment may either define a dielectric interface or a conducting surface, which is controlled by the real part of the entry Zs (see below). If Region 1 and Region 2 are identical, the curve segment must define a conducting surface, implying that Zs(real) must be non-negative.

Region 2 (Region 2) [integer].

The number of the region on the second side of the curve segment.

Zs (real) (*Zs (real)*) [real number].

The real part of Zs, in ohm/meter, Zs being the complex surface impedance of the surface defined by the curve segment. Zs(real) >= 0 specifies a conducting surface; Zs(real) = -1 specifies a dielectric interface and region 1 and 2 must be different.

The default value is Zs = 0 which specifies a perfect conductor.

Zs (imag) (*Zs (imag)*) [real number].

The imaginary part of Zs, in ohm/meter.

Length Unit (*length_unit*) [item from a list of character strings], default: **m.**

The length unit to be applied for the coordinates in the specification of the Nodes.

Coordinate Order (*coor_order*) [item from a list of character strings], default: **z_rho.**

Specification of the order of the coordinates z and ρ in the attribute Nodes.

z_rho

z is the first coordinate and ρ is the second coordinate in the specification of the Nodes.

rho_z

ρ is the first coordinate and z is the second coordinate in the specification of the Nodes.

Remarks

Regions, nodes and segments are described in tables. The order of the regions, etc. in the tables is not important.

This section contains discussions on the following topics:

- Definitions
- Segments
- Mesh rules

and, finally, an example on specification of a mesh is given:

- Application example

Definitions

Objects of the class **BoR Mesh** define Body of Revolution (BoR) scatterers, i.e., the scatterer is rotational symmetry around an axis, the axis of rotation. The surface of the scatterer may be described by a curve (or more curves), the generatrix, in a plane containing the axis of rotation. The surface is generated when this curve is rotated around the axis.

It is thus sufficient to specify the curve in order to describe the scatterer. The curve is specified by a set of nodes which are connected either by linear segments, linear from node to node, or by cubic segments following a third

order polynomial through a set of four consecutive nodes. The segments must be defined such that adjacent segments have a node in common.

If the segments are arranged to form a closed volume (when rotated), this volume is referred to as a region. A region can be assigned material properties (the relative permittivity, the relative permeability and the loss tangent) modelling a volume filled with a homogeneous material with these properties, e.g., a dielectric. Two closed volumes with the same material properties must be considered as different regions.

Segments

The segments of which the generatrix is constituted can be linear or curved. The linear segments are defined by two nodes and the curved segments are defined by four nodes. The shape of a curved segment is modelled by a third order polynomial passing through the specified nodes. The number of nodes and segments needed for describing a surface is determined by the curvature of this surface. Short segments defined by many nodes will in general give the best description of the geometry of a scatterer with curved surface.

When the *Body of Revolution* scatterer defined by the BoR-mesh is used in a MoM-calculation, the generatrix is meshed internally. By this meshing, the generatrix of the scatterer is discretized using curved mesh segments which may be up to 2λ . The mesh segments are described by polynomials of up to 3rd order. If the curvature is significant within a single mesh segment this geometrical approximation may be too crude, and the mesh segment size is automatically decreased to reduce this geometrical approximation error.

The two types of curve segments are shown in Figure 1.



Figure 1

The two types of curve segments, the linear segment defined by two nodes, A and B, and the cubic segment defined by four nodes, A, B, C and D.

Mesh rules

Some rules are given below for the segments of the generatrix.

It is most convenient to connect neighbour segments by letting the same node define the common end points of the segments. This is, however, not mandatory. The neighbour segments may start at nodes with different numbers as long as these nodes have the same position. It is important to ensure connectivity between the segments, cf. rule 2 below.

The following rules apply for a *BoR Mesh*:

1. Segments can only be connected at end points.

2. Two curve segments are considered connected if the position of an end node of one segment is identical to the position of an end node of the other segment within a tolerance of $10^{-5}\lambda$, λ being the wavelength.
If the rule is fulfilled the currents are allowed to flow from one segment to the neighbour segment.
If the rule is not fulfilled, the segments will be regarded as independent scatterers and the currents are not allowed to flow between the segments.
3. A node can be shared by no more than 6 segments. An error is issued during the computation if this condition is violated.
4. Dielectric regions must be closed. An error is issued during the computation if this condition is violated.
5. A closed metallic region should be assigned to be region number -1 if it is in contact with a dielectric region. If the region number is not assigned to -1 a significant overhead is introduced in the computations since unnecessary basis functions are used to model the zero currents on the internal side of the closed metallic region.
If the closed metallic region is not in contact with a dielectric region, there is no such overhead in the computations, and the region number -1 needs not to be specified.
6. A dielectric region must be given a positive region number. The segments bordering the region towards another dielectric region and towards the free space must have $Zs(\text{real}) = -1$ in order to flag that it is a dielectric boundary ($Zs(\text{imag})$ is not used). If the segment is conducting then $Zs(\text{real}) \geq 0$; this specifies that the neighbour region here is a conducting region or that the segment is a conducting layer between two dielectric regions.

Application example

For the *BoR Mesh* sets of regions, segments and nodes shall be specified. The sections connect the nodes and enclose one or more regions. In Figure 2, an example of how a mesh may be build is shown. The figure shows an extended hemispherical dielectric lens.

For this example, the enclosed region is a dielectric region with dielectric constant (relative permittivity) of 3.0 and is defined by specifying the Regions attribute:

```
regions : table
(
    1 3.00000E+00 1.00000E+00 0.00000E+00
)
```

Next, the Nodes, from which the Cubic Segments and Linear Segments are determined, are specified as follows:

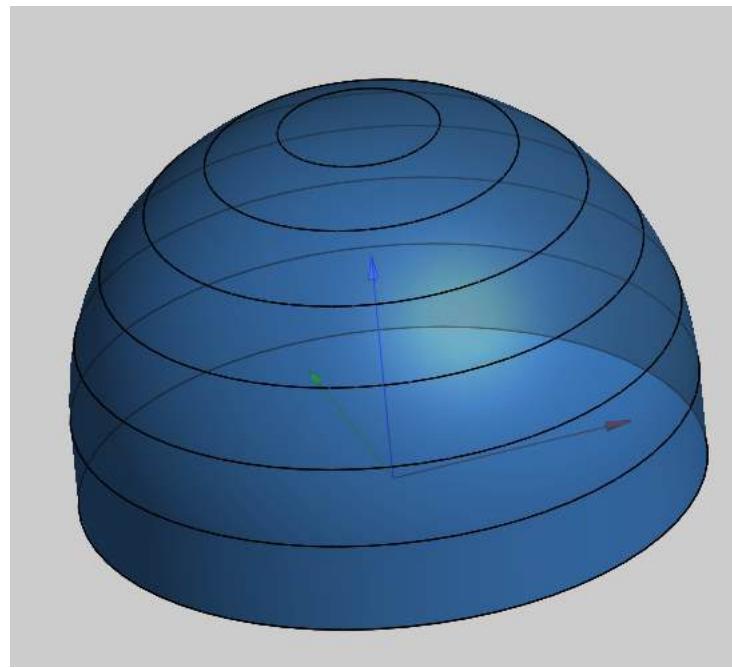


Figure 2

An example of definition of a mesh. An extended hemispherical dielectric lens is created using both linear and cubic segments.

```

nodes          : table
(
  1  0.00           "ref(R)"
  2  0.00           0.00
  3  3.75           "ref(R)"
  4  "3.75+ref(R)*sin(ref(dphi))" "ref(R)*cos(ref(dphi))"
  5  "3.75+ref(R)*sin(2*ref(dphi))" "ref(R)*cos(2*ref(dphi))"
  6  "3.75+ref(R)*sin(3*ref(dphi))" "ref(R)*cos(3*ref(dphi))"
  7  "3.75+ref(R)*sin(4*ref(dphi))" "ref(R)*cos(4*ref(dphi))"
  8  "3.75+ref(R)*sin(5*ref(dphi))" "ref(R)*cos(5*ref(dphi))"
  9  "3.75+ref(R)*sin(6*ref(dphi))" "ref(R)*cos(6*ref(dphi))"
  10 "3.75+ref(R)*sin(7*ref(dphi))" "ref(R)*cos(7*ref(dphi))"
  11 "3.75+ref(R)*sin(8*ref(dphi))" "ref(R)*cos(8*ref(dphi))"
  12 "3.75+ref(R)*sin(9*ref(dphi))" "ref(R)*cos(9*ref(dphi))"
  13 "3.75+ref(R)*sin(10*ref(dphi))" "ref(R)*cos(10*ref(dphi))"
  14 "3.75+ref(R)*sin(11*ref(dphi))" "ref(R)*cos(11*ref(dphi))"
  15 "3.75+ref(R)*sin(12*ref(dphi))" "ref(R)*cos(12*ref(dphi))"
  16 "3.75+ref(R)*sin(13*ref(dphi))" "ref(R)*cos(13*ref(dphi))"
  17 "3.75+ref(R)*sin(14*ref(dphi))" "ref(R)*cos(14*ref(dphi))"
  18 "3.75+ref(R)*sin(15*ref(dphi))" "ref(R)*cos(15*ref(dphi))"
  19 "3.75+ref(R)*sin(16*ref(dphi))" "ref(R)*cos(16*ref(dphi))"
  20 "3.75+ref(R)*sin(17*ref(dphi))" "ref(R)*cos(17*ref(dphi))"
  21 "3.75+ref(R)"           0.00000E+00
)

```

Note that the radius (R) and $\Delta\phi$ ($dphi$) are specified by the *Real Variable* in a way that if R or $dphi$, is changed, then the geometry of the lens will

change accordingly. For the specific case, R=12.5 and dphi=0.08727 (which corresponds to 5 degrees).

Finally, the segments are defined. These specifies how the nodes shall be connected resulting in the surfaces of the structure. Most of the segments are Cubic Segments specifying the curved parts of the dielectric lens. The Cubic Segments are defined by four nodes each, e.g nodes 3 - 4 - 5 - 6 or nodes 15 - 16 - 17 - 18. These segments are specified in the following table:

```
cubic_segments : table
(
  1   3   4   5   6   0   1   -1.00000E+00  0.00000E+00
  2   6   7   8   9   0   1   -1.00000E+00  0.00000E+00
  3   9   10  11  12  0   1   -1.00000E+00  0.00000E+00
  4   12  13  14  15  0   1   -1.00000E+00  0.00000E+00
  5   15  16  17  18  0   1   -1.00000E+00  0.00000E+00
  6   18  19  20  21  0   1   -1.00000E+00  0.00000E+00
)
```

These 6 segments define the curved part of the dielectric lens. Note that Zs (real), 8'th column, is -1 indicating a dielectric interface. In addition to the Cubic Segments, two Linear Segments, which connects two nodes by a straight line, are specified:

```
linear_segments : table
(
  1   1   2   0   1   -1.00000E+00  0.00000E+00
  2   1   3   0   1   -1.00000E+00  0.00000E+00 ) The first linear
```

segment specifies the bottom plate that encloses the lens, whereas the second segment specifies the vertical segment at the bottom.

CIRCULAR SYMMETRIC REFLECTOR

Purpose

The scatterers which may be defined as *Circular Symmetric Reflector* are given by a generatrix which is rotated around the axis of the reflector. The generatrix is specified by a set of points and intermediate points are interpolated.

The intermediate points are determined by a third order spline function:

Spline Reflector

The intermediate points are determined by a cubic interpolation. A tip may be specified at the centre of the reflector:

Tabulated Reflector

This definition of a reflector is restricted to *BoR-MoM*, *MoM*, and *PO*, *Multi-Face Scatterer* calculations of the scattered field. When GTD is to be applied then the reflector shall be specified as a *Reflector* or a *Reflector with Panels*.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Body of Revolution*→*Circular Symmetric Reflector*

SPLINE REFLECTOR (spline_circ_sym_reflector)

Purpose

The class *Spline Reflector* defines a rotational symmetric reflector by a reflector profile given as cubic spline functions passing through a number of data points.

The scattering by a *Spline Reflector* may be determined by *BoR-MoM*, *MoM*, and *PO, Multi-Face Scatterer*.

This class is only available with the MoM add-on.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Body of Revolution*→*Circular Symmetric Reflector*→*Spline Reflector*

Remarks

Syntax

```
<object name> spline_circ_sym_reflector
(
    coor_sys           : ref(<n>),
    z_offset          : <rl>,
    length_unit       : <si>,
    coor_order        : <si>,
    nodes             : table(
                            <r> <r>
                            ...)
```

)

where

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

<si> = item from a list of character strings

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

The spline reflector is defined in this coordinate system, by rotation of the spline segments around the z-axis of the coordinate system.

z-Offset (*z_offset*) [real number with unit of length], default: **0**.

This specifies a constant value, which is added to the z-coordinates of the defined Nodes whereby the exterior is displaced z-Offset along the z-axis (the axis of symmetry).

Length Unit (*length_unit*) [item from a list of character strings], default: **m**.

The length unit to be applied for the coordinates in the specification of the Nodes.

Coordinate Order (*coor_order*) [item from a list of character strings], default: **rho_z**.

Specification of the order of the coordinates z and ρ in the attribute Nodes.

z_rho

z is the first coordinate and ρ is the second coordinate in the specification of the Nodes.

rho_z

ρ is the first coordinate and z is the second coordinate in the specification of the Nodes.

Nodes (*nodes*) [table (2,2)].

Table with 2 columns specifying the coordinates of the points (the nodes) defining the spline curve for the reflector, node by node for increasing ρ -values. The unit of the coordinates are determined by the attribute Length Unit. In addition, the ρ -values have to be equal or greater than 0.

c1 (c1) [real number].

First coordinate for the node, according to the attribute Coordinate Order.

c2 (c2) [real number].

Second coordinate for the node, according to the attribute Coordinate Order.

Remarks

The reflector profile is given as a spline function, $f = f(\rho)$ passing through a number of user-defined data points.

Assume that the user has specified N_p data points, (z_i, ρ_i) , $i = 1, 2, \dots, N_p$, with $\rho_1 < \rho_2 < \dots < \rho_{N_p}$. The points define $N_p - 1$ intervals along the ρ -axis. In each interval, the profile is given by a third-order polynomial. The polynomials are constrained such that the profile function $f(\rho)$ passes through the data points, and has continuous first and second order derivatives with respect to ρ at the data points. At the two end-points, ρ_1 and ρ_{N_p} , the not-a-knot end conditions are used.

An example on a *Spline Reflector* is given in Figure 1. In this example, the reflector is defined using $N_p = 6$ points from $\rho = 0$ m to $\rho = 0.5$ m.

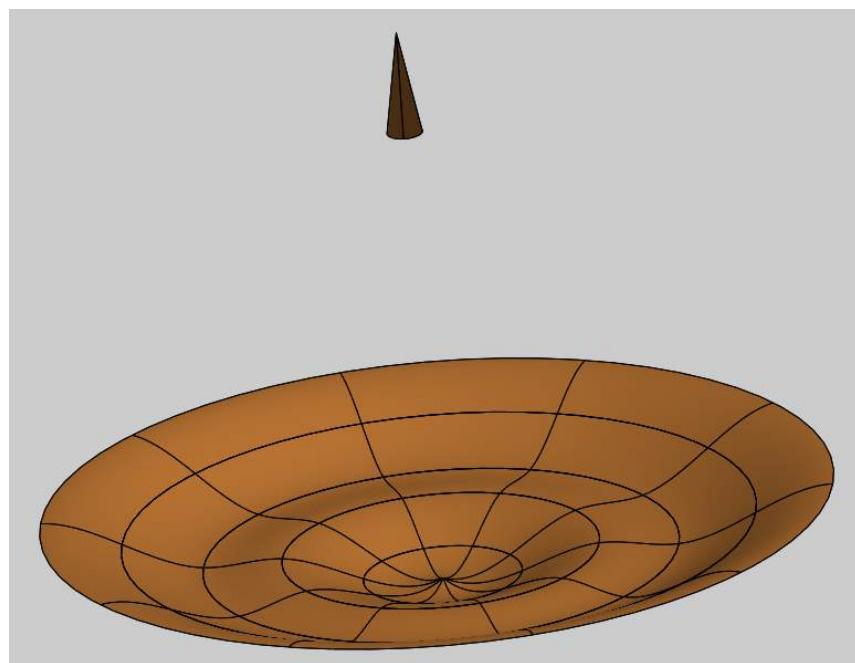


Figure 1

Example of a reflector given as a spline profile using by $N_p = 6$ points from $\rho = 0$ m to $\rho = 0.5$ m.

TABULATED REFLECTOR (tabulated_circ_sym_reflector)

Purpose

The class *Tabulated Reflector* defines a rotationally symmetric reflector by means of tabulated data for a profile of the surface. Data points are given as z -values along a radial surface cut.

The scattering by a *Tabulated Reflector* may be determined by *BoR-MoM*, *MoM*, and *PO, Multi-Face Scatterer*.

This class is only available with the MoM add-on.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Body of Revolution*→*Circular Symmetric Reflector*→*Tabulated Reflector*

Remarks

Syntax

```
<object name> tabulated_circ_sym_reflector
(
    coor_sys           : ref(<n>),
    file_name          : <f>,
    r_unit              : <si>,
    z_unit              : <si>,
    r_factor            : <r>,
    z_factor            : <r>,
    n_points            : <i>,
    tip                 : <si>,
    list                : <si>,
    z_offset             : <rl>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

The tabulated reflector is defined in this coordinate system, by rotation of the tabulated segments around the z -axis of the coordinate system.

File Name (*file_name*) [file name].

The name of the input file containing the data specifying the reflector front side. The format of this file is described in the section *Rotationally Symmetric Surface*.

rho-Unit (*r_unit*) [item from a list of character strings], default: **m**.

The common unit assumed for file data in the radial direction (orthogonal to the axis of symmetry).

z-Unit (*z_unit*) [item from a list of character strings], default: **m**.

The common unit assumed for file data in the axial direction.

rho-Factor (*r_factor*) [real number], default: **1**.

On input, the radial data in the file are scaled by this number.

z-Factor (*z_factor*) [real number], default: **1**.

On input, the axial data in the file are scaled by this number.

Number of Points (*n_points*) [integer], default: **0**.

The data points defining the reflector profile are interpolated into a set of equispaced points. If the attribute Number of Points is set to the default value of 0 the number of equispaced points is the same as the number of data points in the original data set. If the attribute is a positive number the data are interpolated into Number of Points equispaced points.

Tip (*tip*) [item from a list of character strings], default: **defined_in_file**.

Specifies if the reflector has a tip at the centre or not, i.e. determines the local geometry on the axis of symmetry, $\rho = 0$. The setting may be defined in the file:

defined_in_file

The tip is controlled by the parameter KTIP in the data file.

off

The reflector does not have a tip (independently of the setting of KTIP in the data file).

on

The reflector has a tip (independently of the setting of KTIP in the data file).

List (*list*) [item from a list of character strings], default: **off**.

Specifies if the subreflector data should be listed to the standard output file.

off

The data should not be listed.

on

The data should be listed.

z-Offset (*z_offset*) [real number with unit of length], default: **0**.

Defines a displacement of the subreflector along the *z*-axis, i.e. the axis of symmetry.

Remarks

The surface of the reflector is expressed as a function, $z = z(\rho)$, of the radial distance, ρ , from the axis of symmetry.

The file contains n pairs of (z, ρ) -values that define $z(\rho)$:

$$z_i(\rho_i), \quad i = 1, 2, \dots, n, \quad \rho_i \geq 0 \quad (1)$$

At intermediate points, the function is determined by cubic (3rd-order) interpolation in a set of equispaced points. In the end intervals,

$$0 \leq \rho \leq \rho_2 \quad \text{and} \quad \rho > \rho_{n-1}, \quad (2)$$

the interpolation is parabolic (2nd-order). This interpolation will usually result in a tip at $\rho = 0$. If the attribute `Tip` is set to 'off', the cubic interpolation will be continued across $\rho = 0$ applying an additional definition point at $\rho = -\rho_i$ with

$$z(-\rho_i) = z(\rho_i) \quad (3)$$

where ρ_i is the smallest non-zero radial value read from the file.

The file may contain data not equispaced in ρ , in which case the program will pre-interpolate the data into a new, equispaced set. The number of points in the new set is determined by the attribute `Number of Points`.

DGR INTERCOSTALS

Purpose

Intercostals are the distance pieces applied to keep the two surfaces of a dual-gridded reflector (DGR) in place. In the menu *DGR Intercostals* four types of intercostals are available:

Regular intercostals

Circular Stiffeners

Polygonal Stiffeners

a large dielectric circular ring between the two reflector surfaces

Support Ring

And two, or more, dielectric circular rings

Concentric Support Rings

Links

Classes→*Geometrical Objects*→*Scatterer*→*DGR Intercostals*

CIRCULAR STIFFENERS (circular_stiffeners)

Purpose

Objects of the *Circular Stiffeners* class are used to define intercostals in the form of circular cylinders (stiffeners) between the two reflectors in a dual-gridded reflector antenna system.

The *Circular Stiffeners* may be solid or hollow. In the latter case, the stiffeners are tubes with a wall of finite thickness and circular cross section. The two end-faces of each stiffener are automatically made conformal to the surfaces of the two reflectors. The stiffeners may consist of a dielectric material, possibly lossy, or of a perfectly conducting material. Several stiffeners of the same radius may be specified within a single object of this class.

The scattering from *Circular Stiffeners* may be determined by *MoM*.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Geometrical Objects](#)→[Scatterer](#)→[DGR Intercostals](#)→[Circular Stiffeners](#)

Remarks

Syntax

```
<object name> circular_stiffeners
(
    front_reflector           : ref(<n>),
    rear_reflector            : ref(<n>),
    stiffener_type             : <si>,
    coor_sys                  : sequence(ref(<n>), ...),
    centre                     : struct(x:<rl>, y:<rl>),
    outer_radius               : <rl>,
    wall_thickness              : <rl>,
    dielectric_constant        : <r>,
    loss_tangent                : <r>
)
```

where

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

<si> = item from a list of character strings

Attributes

Front Reflector (*front_reflector*) [name of an object].

Reference to a *Reflector* object which then defines the front reflector in a dual-gridded reflector antenna system. The object is also allowed to be of the class *Aperture in Screen*.

Rear Reflector (*rear_reflector*) [name of an object].

Reference to a *Reflector* object which then defines the rear reflector in a dual-gridded reflector antenna system. The object is also allowed to be of the class *Aperture in Screen*.

Stiffener Type (*stiffener_type*) [item from a list of character strings], default: **solid_dielectric**.

The stiffeners may be defined as one of the following types:

solid_dielectric

Each stiffener is solid and dielectric. The dielectric material is defined by the attributes *dielectric_constant* and *loss_tangent*. The attribute *wall_thickness* is not used.

dielectric_tube

Each stiffener is hollow and dielectric. The dielectric material is defined by the attributes *dielectric_constant* and *loss_tangent*. A positive thickness of the wall of the tube must be specified in the attribute *wall_thickness*.

solid_conductor

Each stiffener is solid and perfectly conducting. The attributes *dielectric_constant*, *loss_tangent*, and *wall_thickness* are not used.

conducting_tube

Each stiffener is hollow and perfectly conducting. The attributes *dielectric_constant* and *loss_tangent* are not used. A non-negative thickness of the wall of the tube must be specified in the attribute *wall_thickness*. If *wall_thickness* is specified to zero, the wall is modelled as infinitely thin.

Coordinate System (*coor_sys*) [sequence of names of other objects].

A sequence of references to objects of one of the *Coordinate Systems* classes. Each member of the sequence describes a local coordinate system which defines the position and orientation of one stiffener: The stiffener is parallel to the *z*-axis of that coordinate system. See the remarks below.

The number of stiffeners contained in the *Circular Stiffeners* object is thus equal to the number of specified coordinate systems.

Centre (*centre*) [struct].

The position of the centre of the circular cross section of the stiffeners in the *xy*-plane of the local coordinate systems (given in attribute *coor_sys*) specifying the stiffeners. The relative centre position is identical for all stiffeners of the object.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of the centre.

y (*y*) [real number with unit of length], default: **0**.
y-coordinate of the centre.

Outer Radius (*outer_radius*) [real number with unit of length].

The radius of the circular cross section of each of the stiffeners contained in the object. The radius must be positive.

Wall Thickness (*wall_thickness*) [real number with unit of length], default: **0**.

The wall thickness of the tube when *stiffener_type* is set to *dielectric_tube* or *conducting_tube*. The *wall_thickness* must be positive for a dielectric tube. For the perfectly conducting tube, it is further allowed to specify the *wall_thickness* to zero in which case an infinitely thin wall is modelled.

Dielectric Constant (*dielectric_constant*) [real number], default: **1**.

The relative permittivity of the dielectric medium when *stiffener_type* is set to *dielectric_tube* or *solid_dielectric*. For other cases the value of *dielectric_constant* is not used. For the default value, the stiffener has the characteristics of free space.

Loss Tangent (*loss_tangent*) [real number], default: **0**.

The loss tangent of the dielectric medium when *stiffener_type* is set to *dielectric_tube* or *solid_dielectric*. For other cases the value of *loss_tangent* is not used. For the default value, the dielectric is lossless.

Remarks

The stiffeners specified in an object of the *Circular Stiffeners* class all have the same values for centre displacement, radius and material properties. If circular stiffeners with different values should be modelled, different *Circular Stiffeners* objects must be used.

The number of stiffeners and their orientations are given by specifying a number of local coordinate systems (in attribute Coordinate System), one coordinate system for each stiffener. Each stiffener has its length axis parallel to the *z*-axis of the local coordinate system and is displaced in the local coordinate system as specified by the attribute Centre.

When the stiffeners are defined between two reflectors then the intersections between the stiffeners and the reflectors must be fully within the rim of the reflectors. When one - or both - of the attributes Front Reflector and Rear Reflector are specified to an object of class *Aperture in Screen* then the intersections between the stiffeners and the screen must be fully outside the rim of the aperture.

In the following it is anticipated that the attributes Front Reflector and Rear Reflector refer to true reflectors.

In Figure 1 three stiffeners between a front and a rear reflector are shown. The local coordinate systems are shown as well. In the example, the stiffeners are all oriented nearly perpendicular to the reflector surfaces, but this is not required.

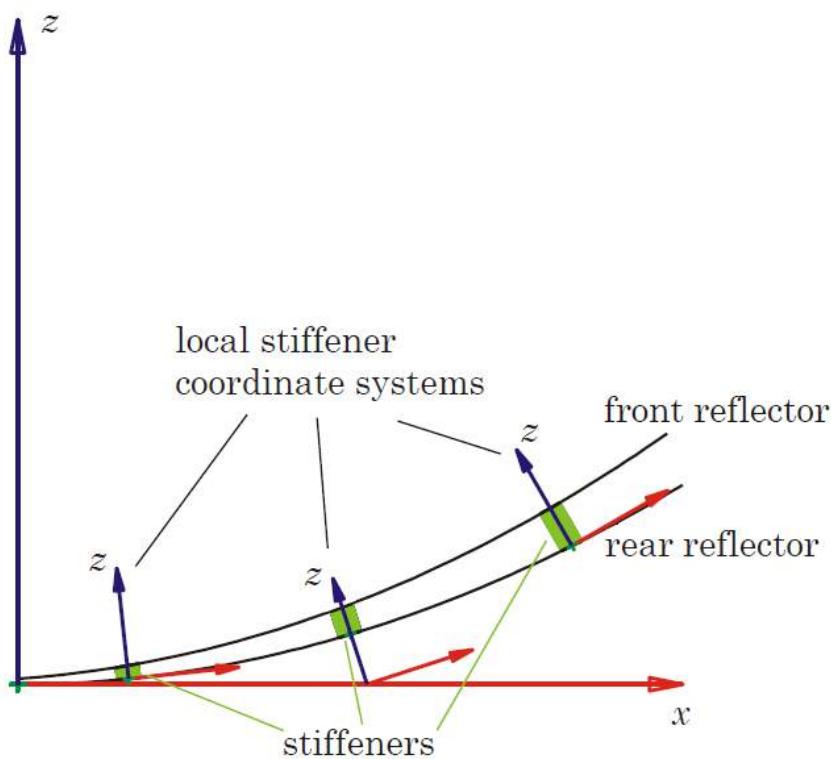


Figure 1

Illustration of three stiffeners between a front and a rear reflector. The local coordinate systems of the stiffeners may be defined to give the most convenient definition of the stiffeners.

Analysis method

Stiffeners can be analyzed using MoM (by applying an object of class *MoM*). A full MoM solution of the front and rear reflectors with the stiffeners in between would be very time-consuming and not worth the computational overhead. Instead, PO should be used for the two reflectors, and MoM should be used for the stiffeners. In order to decouple the stiffeners from the two reflectors, an artificial gap of 0.1λ (at the highest applied frequency) is automatically introduced by GRASP between the stiffeners and the reflectors. This gap has a negligible influence on the field scattered by the reflector-stiffener configuration but the decoupling between PO and MoM simplifies the analysis significantly. The gap also implies that the PO integrals on the front and rear reflectors converge without the need for a time-consuming singularity extraction when field points upon the stiffeners coincide with the reflector surface. Nevertheless, the stiffeners are located very close to the reflector surfaces upon which the PO currents shall be computed and the accuracy should be checked carefully by performing an auto-convergence procedure when evaluating the PO currents on the front and rear reflector.

When plotting, it is the specified geometry without the artificial gaps which is shown.

Other applications

In the examples above the stiffeners are applied as intercostals between two reflectors in a dual reflector system. Stiffeners may, however, also be

applied to model other cylindrical solid or tubular scatterers. For example, a reflector with finite thickness may be modelled as a solid conducting stiffener which is very wide and placed between two close reflector surfaces. In such cases care must be taken for the mentioned artificial gap of 0.1λ applied in the analysis of the stiffener.

POLYGONAL STIFFENERS (polygonal_stiffeners)

Purpose

Objects of the *Polygonal Stiffeners* class are used to define intercostals in the form of polygonal cylinders (stiffeners) between the two reflectors in a dual-gridded reflector antenna system.

The *Polygonal Stiffeners* may be solid or hollow. In the latter case, the stiffeners are tubes with a finite thickness. The two end-faces of each stiffener are automatically made conformal to the surfaces of the two reflectors. The stiffeners may consist of a dielectric material, possibly lossy, or of a perfectly conducting material. Several stiffeners with identical cross section may be specified within a single object of this class.

The scattering from *Polygonal Stiffeners* may be determined by *MoM*.

This class is only available with the MoM add-on.

Links

Classes→*Geometrical Objects*→*Scatterer*→*DGR Intercostals*→*Polygonal Stiffeners*

Remarks

Syntax

```
<object name> polygonal_stiffeners
(
    front_reflector          : ref(<n>),
    rear_reflector           : ref(<n>),
    stiffener_type           : <si>,
    coor_sys                 : sequence(ref(<n>), ...),
    cross_section            : sequence(
                                struct(x:<rl>,
                                       y:<rl>),
                                ...),
    wall_thickness            : <rl>,
    dielectric_constant      : <r>,
    loss_tangent              : <r>
)
```

where

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

<si> = item from a list of character strings

Attributes

Front Reflector (*front_reflector*) [name of an object].

Reference to a *Reflector* object which then defines the front reflector in a dual-gridded reflector antenna system. The object is also allowed to be of the class *Aperture in Screen*.

Rear Reflector (*rear_reflector*) [name of an object].

Reference to a *Reflector* object which then defines the rear reflector in a dual-gridded reflector antenna system. The object is also allowed to be of the class *Aperture in Screen*.

Stiffener Type (*stiffener_type*) [item from a list of character strings], default: **solid_dielectric**.

The stiffeners may be defined as one of the following types:

solid_dielectric

Each stiffener is solid and dielectric. The dielectric material is defined by the attributes *dielectric_constant* and *loss_tangent*. The attribute *wall_thickness* is not used.

dielectric_tube

Each stiffener is hollow and dielectric. The dielectric material is defined by the attributes *dielectric_constant* and *loss_tangent*. A positive thickness of the wall of the tube must be specified in the attribute *wall_thickness*.

solid_conductor

Each stiffener is solid and perfectly conducting. The attributes *dielectric_constant*, *loss_tangent*, and *wall_thickness* are not used.

conducting_tube

Each stiffener is hollow and perfectly conducting. The attributes *dielectric_constant* and *loss_tangent* are not used. A non-negative thickness of the wall of the tube must be specified in the attribute *wall_thickness*. If *wall_thickness* is specified to zero, the wall is modelled as infinitely thin.

Coordinate System (*coor_sys*) [sequence of names of other objects].

A sequence of references to objects of one of the *Coordinate Systems* classes. Each member of the sequence describes a local coordinate system which defines the position and orientation of one stiffener: The stiffener is parallel to the *z*-axis of that coordinate system. See the remarks below. The number of stiffeners contained in the *Polygonal Stiffeners* object is thus equal to the number of specified coordinate systems.

Cross Section (*cross_section*) [sequence of structs].

The polygonal cross-section of the stiffeners in the xy -plane of the local coordinate systems (given in attribute *coor_sys*) specifying the stiffeners. The cross-section is a polygon specified by the (x, y) -coordinates of the corners. The corners must be given in succession (either clockwise or anticlockwise) and specify a convex polygon (which does not need to encircle the origin of the xy -plane). The polygon is automatically closed by the side connecting the last point in the sequence to the first point.

x (*x*) [real number with unit of length].

x-coordinate of a corner.

y (*y*) [real number with unit of length].

y-coordinate of the same corner.

Wall Thickness (*wall_thickness*) [real number with unit of length], default: **0**.

The wall thickness of the tube when *stiffener_type* is set to 'dielectric_tube' or 'conducting_tube'. It is the outer side of the wall which follows the specified *cross_section*. The *wall_thickness* must be positive. For the perfectly conducting tube, it is further allowed to specify the *wall_thickness* to zero in which case an infinitely thin wall is modelled. The *wall_thickness* must not be too large. All the polygonal sides of the tube must have a non-vanishing inner wall parallel to the outer wall.

Dielectric Constant (*dielectric_constant*) [real number], default: **1**.

The relative permittivity of the dielectric medium when *stiffener_type* is set to 'dielectric_tube' or 'solid_dielectric'. For other cases the value of *dielectric_constant* is not used.

Loss Tangent (*loss_tangent*) [real number], default: **0**.

The loss tangent of the dielectric medium when attribute *stiffener_type* is set to "dielectric_tube" or "solid_dielectric". For other cases the value of *loss_tangent* is not used

Remarks

The stiffeners specified in an object of class *Polygonal Stiffeners* all have the same cross section and material properties. If polygonal stiffeners with different cross sections or different material properties should be modelled different *Polygonal Stiffeners* objects must be used.

The number of stiffeners and their orientations are given by specifying a number of local coordinate systems (in attribute Coordinate System), one coordinate system for each stiffener. Each stiffener has its length axis parallel to the *z*-axis of the local coordinate system and its cross-section defined in the local coordinate system.

When the stiffeners are defined between two reflectors then the intersections between the stiffeners and the reflectors must be fully within the rim of the reflectors. When one - or both - of the attributes Front Reflector and Rear Reflector are specified to an object of class *Aperture in Screen*

then the intersections between the stiffeners and the screen must be fully *outside* the rim of the aperture.

In the following it is anticipated that the attributes Front Reflector and Rear Reflector refer to true reflectors

In Figure 1 three stiffeners between a front and a rear reflector are shown. The local coordinate systems are shown as well. In the example, the stiffeners are all oriented nearly perpendicular to the reflector surfaces, but this is not required.

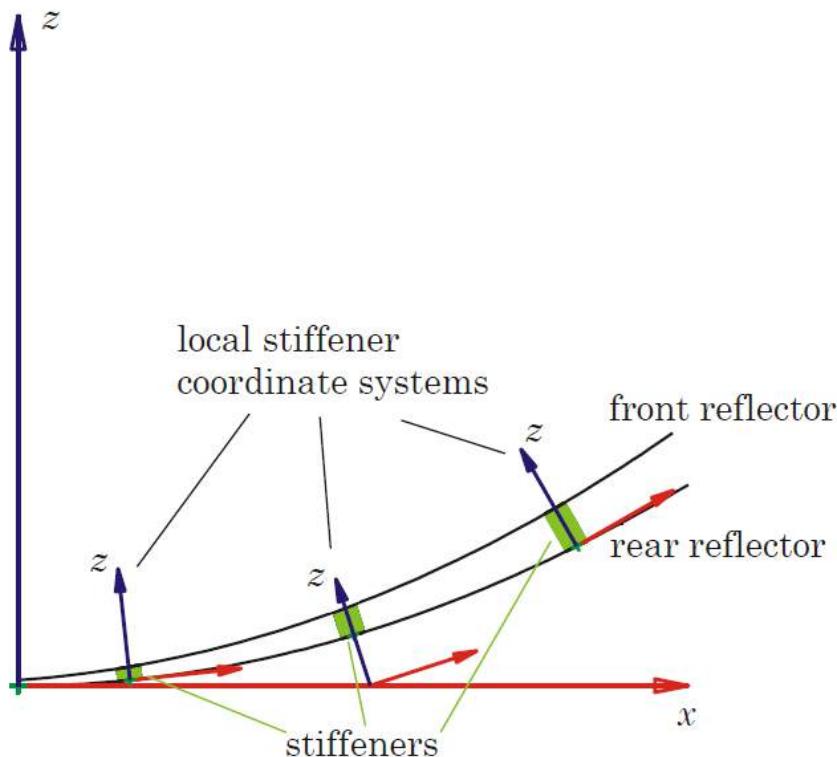


Figure 1

Illustration of three stiffeners between a front and a rear reflector. The local coordinate systems of the stiffeners may be defined to give the most convenient definition of the stiffeners.

Analysis method

Stiffeners can be analyzed using MoM (by applying an object of class *MoM*). A full MoM solution of the front and rear reflectors with the stiffeners in between would be very time-consuming and not worth the computational overhead. Instead, PO should be used for the two reflectors, and MoM should be used for the stiffeners. In order to decouple the stiffeners from the two reflectors, an artificial gap of 0.1λ (at the highest applied frequency) is automatically introduced by GRASP between the stiffeners and the reflectors. This gap has a negligible influence on the field scattered by the reflector-stiffener configuration but the decoupling between PO and MoM simplifies the analysis significantly. The gap also implies that the PO integrals on the front and rear reflectors converge without the need for a time-consuming singularity extraction when field points upon the stiffeners coincide with the reflector surface. Nevertheless, the stiffeners are located very close to the

reflector surfaces upon which the PO currents shall be computed and the accuracy should be checked carefully by performing an auto-convergence procedure when evaluating the PO currents on the front and rear reflector.

When plotting, it is the specified geometry without the artificial gaps which is shown.

Other applications

In the examples above the stiffeners are applied as intercostals between two reflectors in a dual reflector system. Stiffeners may, however, also be applied to model other cylindrical solid or tubular scatterers. For example, a reflector with finite thickness may be modelled as a conducting stiffener which is very wide and placed between two close reflector surfaces. In such cases care must be taken for the mentioned artificial gap of 0.1λ applied in the analysis of the stiffener.

SUPPORT RING (support_ring)

Purpose

Objects of the class *Support Ring* are used to define a large dielectric ring separating the two reflectors in a dual-gridded reflector antenna system. The support ring can be analysed with the Method of Moments, *MoM*, and a special domain-decomposition method is used to keep the memory requirements low. This allows even very large support rings to be analysed on moderately sized computers.

This class is only available with the MoM add-on.

Links

Classes→*Geometrical Objects*→*Scatterer*→*DGR Intercostals*→*Support Ring*

Remarks

Syntax

```
<object name> support_ring
(
    front_reflector           : ref(<n>),
    rear_reflector            : ref(<n>),
    ring_type                 : <si>,
    coor_sys                  : ref(<n>),
    computational_domains     : <i>,
    centre                     : struct(x:<rl>, y:<rl>),
    outer_radius               : <rl>,
    wall_thickness             : <rl>,
    dielectric_constant        : <r>,
    loss_tangent               : <r>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Front Reflector (*front_reflector*) [name of an object].

Reference to a *Reflector* object which then defines the front reflector in a dual-gridded reflector antenna system. The object is also allowed to be of the class *Aperture in Screen*.

Rear Reflector (*rear_reflector*) [name of an object].

Reference to a *Reflector* object which then defines the rear reflector in a dual-gridded reflector antenna system. The object is also allowed to be of the class *Aperture in Screen*.

Ring Type (*ring_type*) [item from a list of character strings], default: **dielectric**.

The support ring may be defined as one of the following types:

dielectric

The support ring is made of a dielectric material as defined by the attributes *dielectric_constant* and *loss_tangent*.

conducting

The support ring is perfectly conducting. The attributes *dielectric_constant* and *loss_tangent* are ignored.

Coordinate System (*coor_sys*) [name of an object].

Reference to an object of one of the *Coordinate Systems* classes.

Computational Domains (*computational_domains*) [integer], default: **8**.

Sets the number of computational sub-domains employed by the domain-decomposition scheme, see the Remarks section for details. A high number of computational domains reduce the memory requirements of the MoM solution but at the same time the error increases.

Centre (*centre*) [struct].

The position of the centre of the circular cross section of the support ring in the *xy*-plane of the local coordinate system specified in the attribute *coor_sys*.

x (x) [real number with unit of length], default: **0**.

x-coordinate of the centre.

y (y) [real number with unit of length], default: **0**.

y-coordinate of the centre.

Outer Radius (*outer_radius*) [real number with unit of length].

The outer radius of the circular cross section of the support ring.

Wall Thickness (*wall_thickness*) [real number with unit of length], default: **0**.

The wall thickness of the ring. The *wall_thickness* must be positive for a dielectric ring. For the perfectly conducting ring, it is further allowed to specify the *wall_thickness* to zero in which case an infinitely thin wall is modelled.

Dielectric Constant (*dielectric_constant*) [real number], default: **1**.

The relative permittivity of the dielectric medium of the ring when *ring_type* is set to 'dielectric'.

Loss Tangent (*loss_tangent*) [real number], default: **0**.

The loss tangent of the dielectric medium when *ring_type* is set to 'dielectric'.

Remarks

The support ring can be analyzed using the Method of Moments, cf. class **MoM**. A full MoM solution of the front and rear reflectors with the support ring in between would be very time-consuming and not worth the computational overhead. Instead, PO should be used on the two reflectors, and MoM on the support ring. In order to decouple the ring from the two reflectors, an artificial gap of 0.1λ (at the highest applied frequency) is automatically introduced by GRASP between the ring and the reflectors. This gap has a negligible influence on the field scattered by the reflector-ring configuration but simplifies the analysis significantly. The gap also implies that the PO integrals on the front and rear reflectors converge without the need for a time-consuming singularity extraction. However, the support ring is located very close to the reflector surfaces and the accuracy should be checked carefully by performing an auto-convergence procedure when computing the PO currents on the front and rear reflector.

When plotting, it is the specified geometry without the artificial gaps which is shown. An example is given in the figure below.

The auto-convergence will usually request a fine integration grid over the part of the reflectors which is close to the support rings. For large reflectors, the front and rear reflectors may with advantage each be subdivided into three reflectors. The integration grid of the middle reflector may be then fine and the grid for the inner and outer reflectors may be coarse which reduces the computational efforts. The outer reflector shall have a centre hole with radius equal to the aperture radius of the middle reflector, and this has a centre hole with radius equal to the aperture radius of the inner reflector. In total the front reflector – as well as the back reflector – has a full solid surface.

Dielectric support rings are usually very large objects in terms of wavelengths and a full MoM solution is generally not possible. However, the important scattering contributions come from the interaction between the reflector shells and the ring whereas the contributions originating in mutual coupling between one part of the ring and another distant part of the ring can be neglected. This can be used to minimise the memory requirements by neglecting far-zone interactions between different parts of the support ring. This is implemented by dividing the actual support ring into a number of sections, sub-domains, which are solved as independent sub-problems each of a manageable size. The domains of the sub-problems are overlapping which allows suppressing surface currents artefacts at the domain boundaries. The number of sub-problems is specified by the attribute *computational_domains* which should be chosen as a trade-off between the memory requirements and the accuracy of the domain-decomposition scheme. Since support rings are normally composed of a weakly scattering dielectric material, the number of sub-domains can be relatively high (and the memory low) without noticeable degradation of the results. GRASP-MoM automatically performs the domain decomposition, runs the analysis for each domain, and finally assembles a set of surface currents without artefacts at the domain boundaries. These surface currents of the full support ring can then be used as a source in further computations.

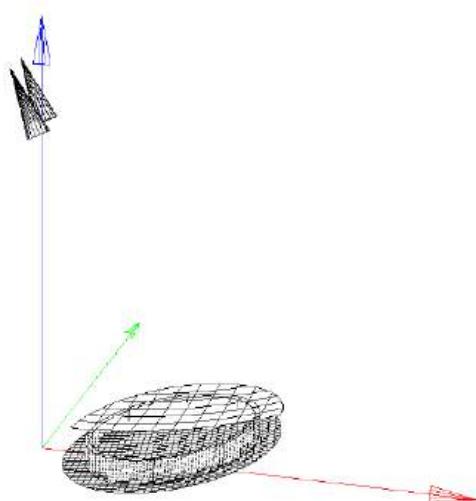


Figure 1

A dual gridded reflector in which the distance between the two reflectors is controlled by a support ring. (only the half of the top reflector is shown).

CONCENTRIC SUPPORT RINGS (concentric_support_rings)

Purpose

Objects of the class *Concentric Support Rings* are used to define a set of dielectric circular concentric rings of large diameters, separating the two reflectors in a dual-gridded reflector antenna system. The rings in *Concentric Support Rings* are not allowed to be in physical contact. The support rings are analysed with the Method of Moments, *MoM*, and a special domain-decomposition method is used to keep the memory requirements low. This allows even very large support rings to be analysed on moderately sized computers.

The class differs from the class *Support Ring* by the ability to set up more than one support ring so that modelling of a circular sandwich structure becomes easier for the user. Furthermore, a thin-sheet option is implemented which allows a very fast analysis of thin dielectric rings modelling the surface layers in a sandwich structure. Normally, these layers have a thickness of a fraction of a wavelength and can be treated as thin sheets.

For technical reasons *Concentric Support Rings* class has been limited to one frequency to be computed in each analysis task. Multiple frequencies must be computed by running separate analysis commands for each frequency.

This class is only available with the MoM add-on.

Links

Classes→*Geometrical Objects*→*Scatterer*→*DGR Intercostals*→*Concentric Support Rings*

Remarks

Syntax

```
<object name> concentric_support_rings
(
    front_reflector           : ref(<n>),
    rear_reflector            : ref(<n>),
    coor_sys                  : ref(<n>),
    computational_domains     : <i>,
    thickness_model           : <si>,
    centre                    : struct(x:<rl>, y:<rl>),
    rings                     : sequence(
                                struct(
                                    outer_radius:<rl>,
                                    wall_thickness:<rl>,
                                    dielectric_constant:<r>,
                                    loss_tangent:<r>),
                                ...))
)
where
<i> = integer
<n> = name of an object
```

$\langle r \rangle$ = real number

$\langle rl \rangle$ = real number with unit of length

$\langle si \rangle$ = item from a list of character strings

Attributes

Front Reflector (*front_reflector*) [name of an object].

Reference to a *Reflector* object which then defines the front reflector in a dual-gridded reflector antenna system. The object is also allowed to be of the class *Aperture in Screen*.

Rear Reflector (*rear_reflector*) [name of an object].

Reference to a *Reflector* object which then defines the rear reflector in a dual-gridded reflector antenna system. The object is also allowed to be of the class *Aperture in Screen*.

Coordinate System (*coor_sys*) [name of an object].

Reference to an object of one of the *Coordinate Systems* classes.

Computational Domains (*computational_domains*) [integer], default: **8**.

Sets the number of computational sub-domains employed by the domain-decomposition scheme, see the Remarks section for details. A high number of computational domains reduce the memory requirements of the MoM solution but at the same time the error increases.

Thickness Model (*thickness_model*) [item from a list of character strings], default: **automatic**.

The method for the MoM analysis may be chosen according to the thickness of the support rings. Below, λ is the free-space wavelength.

automatic

The MoM analysis method is selected automatically. If the *wall-thickness* is less than $\lambda/10$ the *approximate* method is used. Otherwise the *actual* thickness is used.

actual

The actual thickness of the support rings are used in the MoM analysis.

approximate

An approximate thin-sheet MoM analysis is used. This option significantly reduces the computation time and should be used if the thickness of the material is a small fraction of the wavelength, e.g. thickness $\leq \lambda/10$.

Centre (*centre*) [struct].

The position of the centre of the circular cross section of the support ring in the xy -plane of the local coordinate system specified in the attribute *coor_sys*.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of the centre.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of the centre.

Rings (*rings*) [sequence of structs].

Each struct in the sequence defines a support ring and specifies details of it.

Outer Radius (*outer_radius*) [real number with unit of length].

The outer radius of the circular cross section of the support ring.

Wall Thickness (*wall_thickness*) [real number with unit of length].

The wall thickness of the ring. The *wall_thickness* must be positive.

Dielectric Constant (*dielectric_constant*) [real number], default: **1**.

The relative permittivity of the dielectric medium of the ring.

Loss Tangent (*loss_tangent*) [real number], default: **0**.

The loss tangent of the dielectric medium.

Remarks

The class *Concentric Support Rings* is useful for specifying a support ring built as a sandwich structure. Such a structure is often made up of two thin outer layers with a high dielectric constant and a much thicker middle layer with a dielectric constant close to one. Since the rings in *Concentric Support Rings* are not allowed to be in physical contact, the sandwich structure is conveniently modelled by the two surface layers only. It is then assumed that the inner layer is free space which is normally a very good approximation.

The concentric support rings can be analyzed using the Method of Moments (*MoM*). A full MoM solution of the front and rear reflectors with the support rings in between would be very time-consuming and not worth the computational overhead. Instead, PO should be used on the two reflectors, and MoM on the support rings. In order to decouple the rings from the two reflectors, an artificial gap of 0.1λ is automatically introduced by GRASP between the rings and the reflectors. This gap has a negligible influence on the field scattered by the reflector-ring configurations but simplifies the analysis significantly. The gap also implies that the PO integrals on the front and rear reflectors converge without the need for a time-consuming singularity extraction. However, the support rings are located very close to the reflector surfaces and the accuracy should be checked carefully by performing an auto-convergence procedure when computing the PO currents on the front and rear reflector.

When plotting, it is the specified geometry without the artificial gaps which is shown. A typical dual gridded reflector with two closely spaced concentric support rings is shown below.

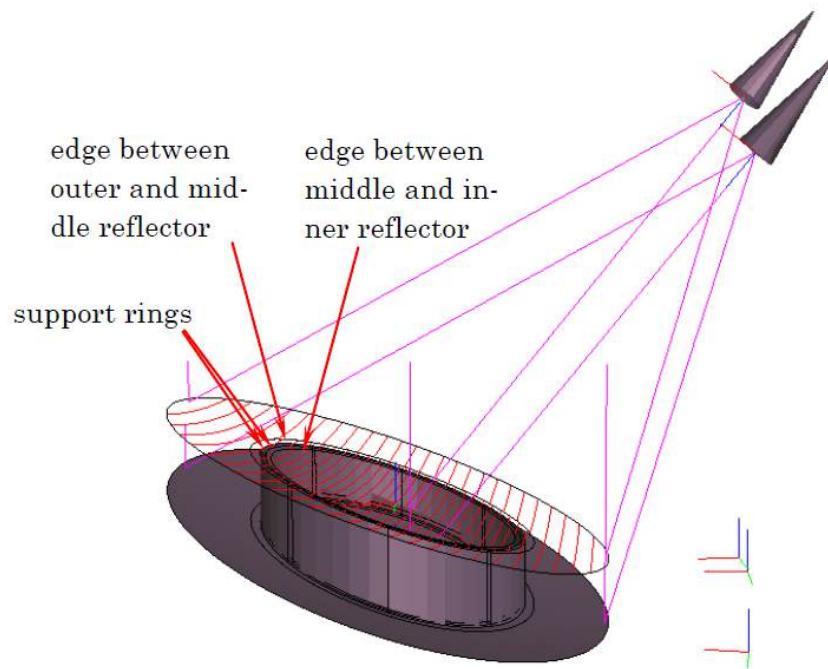


Figure 1

A dual gridded reflector with two support rings between the two reflectors. Each reflector is here subdivided into three parts, an outer, a middle and an inner part, which can be seen when the grid lines of the front reflector are followed. The width of the middle reflector is 1.5λ centred at the support rings.

The auto-convergence will usually request a fine integration grid over the part of the reflectors which is close to the support rings and this will dictate the grid for the full reflectors. When the reflector is large, the front and rear reflectors may with advantage each be subdivided into three concentric reflectors. The outer reflector shall have a centre hole with a radius equal to the aperture radius of the middle reflector, and this has a centre hole with radius equal to the aperture radius of the inner reflector. In total the front reflector – as well as the back reflector – has a full solid surface for which the integration grid of the middle reflector may be fine and the grid for the inner and outer reflectors may be coarse whereby the computational efforts are reduced. Such a method has been applied in the case shown in the figure above.

Dielectric support rings are usually very large objects in terms of wavelengths and a full MoM solution is generally not possible. However, the important scattering contributions come from the interaction between the reflector shells and closely spaced parts of the rings, whereas the contributions originating in mutual coupling between one part of the rings and another distant part of the rings can be neglected. This can be used to minimise the memory requirements by neglecting far-zone interactions between different parts of the support rings. This is implemented by dividing the actual support rings into a number of sections, sub-domains, which are solved as independent sub-problems each of a manageable size. The domains of the sub-problems are overlapping which allows suppressing surface currents

artefacts at the domain boundaries. The number of sub-problems on each ring is specified by the attribute *computational_domains* which should be chosen as a trade-off between the memory requirements and the accuracy of the domain-decomposition scheme. Since support rings are normally composed of a weakly scattering dielectric material, the number of sub-domains can be relatively high (and the memory low) without noticeable degradation of the results. GRASP-MoM automatically performs the domain decomposition, runs the analysis for each domain, and finally assembles a set of surface currents without artefacts at the domain boundaries. These surface currents of the full support ring can then be used as a source in further computations.

APERTURE IN SCREEN (aperture_in_screen)

Purpose

This class defines an aperture in an infinite conducting screen of zero thickness. This is useful for modelling of e.g. an optical aperture stop (an iris diaphragm). The scattering from an Aperture in Screen is determined by means of Babinet's principle.

This class is only available with the Quast add-on.

Links

[Classes](#)→[Geometrical Objects](#)→[Scatterer](#)→[Aperture in Screen](#)

Remarks

Syntax

```
<object name> aperture_in_screen
(
    coor_sys           : ref(<n>),
    rim                : ref(<n>)
)
where
<n> = name of an object
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a [Coordinate Systems](#) object in the xy-plane of which the screen with the aperture is specified.

Rim (*rim*) [name of an object].

Reference to a [Rim](#) object that defines the shape of the aperture.

Remarks

The analysis of an aperture in a screen is performed by means of Babinet's principle as described in the class [PO, Aperture in Screen](#). A plot can be generated by the class [Aperture in Screen Plot](#) as shown in Figure 1.

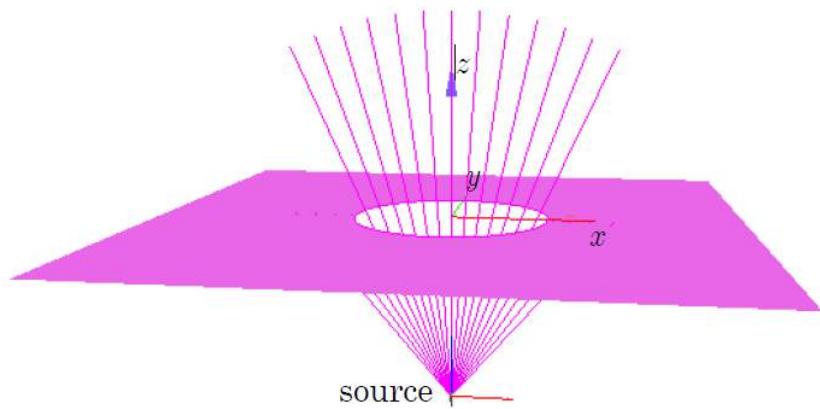


Figure 1

Plot of a source radiating up towards and through an aperture in a screen. The axes of the Aperture in Screen coordinate system defined in the attribute Coordinate System, are shown as x , y and z .

ROOFTOP MIRROR (rooftop_mirror)

Purpose

The class *Rooftop Mirror* is used to define a *Scatterer*, which is composed of two planar mirrors joined at a right angle. See the remarks below for a drawing and a definition of the geometrical parameters. The *Rooftop Mirror* is a frequently used component in quasioptical interferometers. The scattering from a *Rooftop Mirror* may be determined by *PO Analysis* and *MoM*.

This class is only available with the Quast add-on.

Links

[Classes](#)→[Geometrical Objects](#)→[Scatterer](#)→[Rooftop Mirror](#)

Remarks

Syntax

```
<object name> rooftop_mirror
(
    coor_sys : ref(<n>),
    aperture_width : <rl>,
    aperture_height : <rl>,
    z_displacement : <rl>,
    z_rotation : <r>
)
```

where

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the Rooftop Mirror is specified. See the remarks below for details about how the mirror is oriented relative to the coordinate system.

Aperture Width (*aperture_width*) [real number with unit of length].

Width of the aperture measured along the *x*-axis of the coordinate system specified in attribute *coor_sys*.

Aperture Height (*aperture_height*) [real number with unit of length].

Height of the aperture measured along the *y*-axis of the coordinate system specified in attribute *coor_sys*.

z Displacement (*z_displacement*) [real number with unit of length], default: **0**.

Displacement of the mirror along the z -axis of the coordinate system specified in attribute `coor_sys`.

`z` Rotation (`z_rotation`) [real number], default: **0**.

Rotation around the z -axis of the mirror.

Remarks

Analysis methods

The scattering from a Rooftop Mirror may be determined by *PO Analysis* (with *PO*, *Multi-Face Scatterer*) or by *MoM*.

Geometry

A plot of the geometry is shown below in Figure 1.

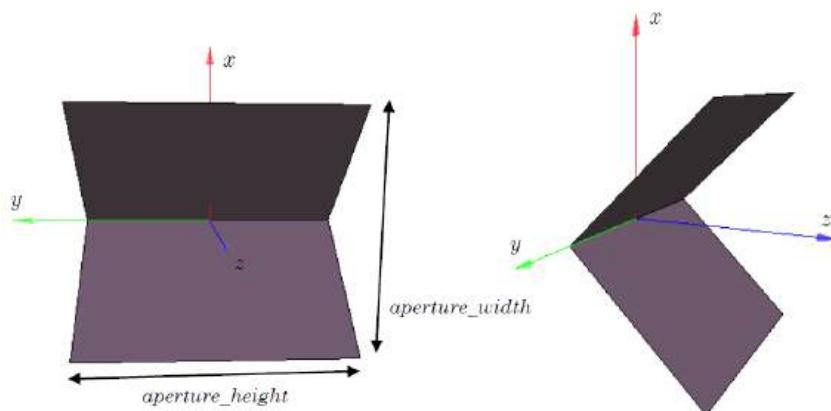


Figure 1 Rooftop mirror and default location in its coordinate system, front view (left) and oblique view (right).

The default location of the rooftop mirror is that the roof line (where the two faces join) coincides with the y -axis and the centre of the roof line is located at the origin of the coordinate system. The xz -plane is orthogonal to both mirror faces and the z -axis bisects the angle between the faces. The `z` Displacement translates the rooftop mirror in the direction of the z -axis and the `z` Rotation rotates the mirror around the z -axis (positive from the x -axis towards the y -axis). In Figure 2, the mirror is translated with a positive `z` Displacement and rotated with a positive `z` Rotation.

In Figure 3, the reflection of a Gaussian beam is illustrated. Normally, the beam will be centred perpendicularly at the centre of the roof line, whereby the reflected beam follows the same centre line as the incoming beam, but that the beam reflects back into itself is difficult to see in an illustration.

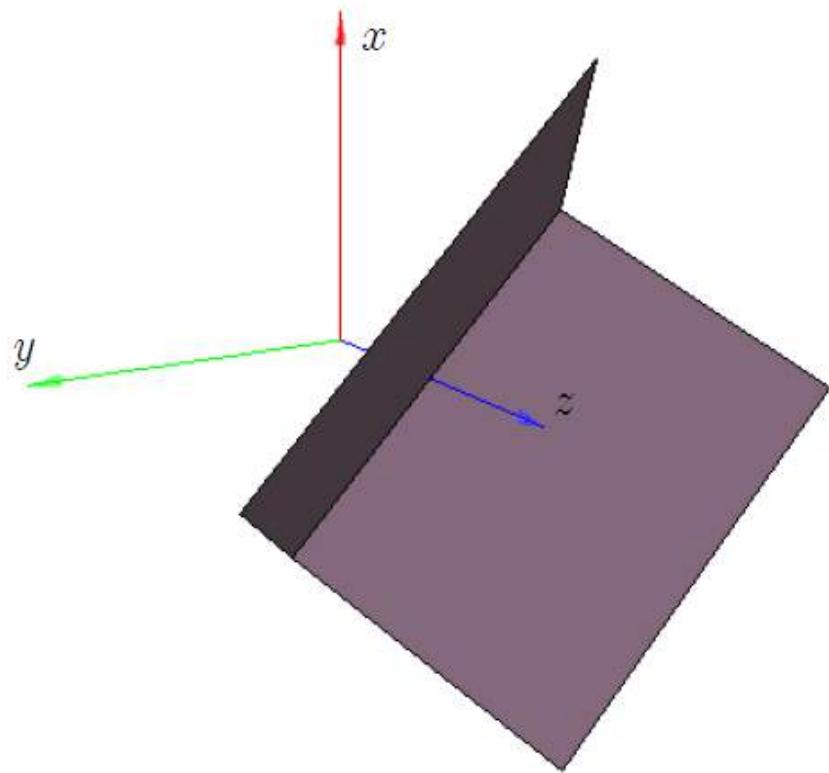


Figure 2 Translated and rotated rooftop mirror.

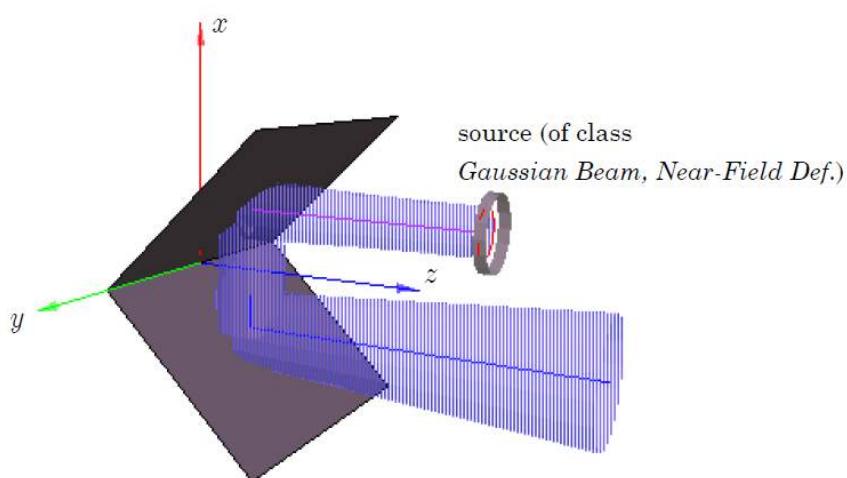


Figure 3 Gaussian beam reflection in a rooftop mirror at its default position.

SIMPLE LENS (simple_lens)

Purpose

The class *Simple Lens* specifies a homogeneous dielectric Scatterer, with rotationally symmetric surfaces. The class is useful for modelling a classical dielectric lens.

The scattering from a *Simple Lens* is calculated by the class *PO, Lens* by means of a combination of Physical Optics and Geometrical Optics.

This class is only available with the Quast add-on.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Simple Lens*

Remarks

Syntax

```
<object name> simple_lens
(
    coor_sys           : ref(<n>),
    diameter          : <rl>,
    refractive_index   : <r>,
    loss_tangent       : <r>,
    r1                 : <rl>,
    r2                 : <rl>,
    bs1                : <r>,
    bs2                : <r>,
    thickness          : <rl>,
    surface1_file      : <f>,
    surface2_file      : <f>,
    length_unit_in_files : <si>
)
```

where

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

<f> = file name

<si> = item from a list of character strings

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to a *Coordinate Systems* object in which the lens is specified. See the definition in the remarks below.

Diameter (*diameter*) [real number with unit of length].

Diameter of lens.

Refractive Index (*refractive_index*) [real number].

The Refractive Index n is related to the relative dielectric constant through $n = \sqrt{\epsilon_r}$, where ϵ_r is the real part of the complex relative dielectric constant $\epsilon = \epsilon_r - j\epsilon_i$.

Loss Tangent (*loss_tangent*) [real number], default: **0**.

The Loss Tangent is defined by ϵ_i/ϵ_r , which is related to the complex relative dielectric constant ϵ by $\epsilon = \epsilon_r - j\epsilon_i$.

r1 (*r1*) [real number with unit of length].

The radius of curvature of lens face 1 on the axis of revolution. A positive sign of *r1* means that the face is convex seen from the exterior of the lens, whereas a negative sign of *r1* means that the face is concave. For a planar face *r1* shall be specified to zero. See the remarks below for details. The attribute is not used if a Surface File 1 is specified.

r2 (*r2*) [real number with unit of length].

The radius of curvature of lens face 2 on the axis of revolution. A positive sign of *r2* means that the face is convex seen from the exterior of the lens, whereas a negative sign of *r2* means that the face is concave. For a planar face *r2* shall be specified to zero. See the remarks below for details. The attribute is not used if a Surface File 2 is specified.

bs1 (*bs1*) [real number], default: **0**.

Conic constant of lens face 1. The conic constant must be zero or negative, see the remarks below for more information. The attribute is not used if a Surface File 1 is specified.

bs2 (*bs2*) [real number], default: **0**.

Conic constant of lens face 2. The conic constant must be zero or negative, see the remarks below for more information. The attribute is not used if a Surface File 2 is specified.

Thickness (*thickness*) [real number with unit of length], default: **0**.

The thickness of the lens on the axis of revolution. By default Thickness is set to zero which has the special meaning that the thickness is automatically set to the minimum possible value. For a biconvex or a plano-convex lens this means that the two faces of the lens will meet at the rim of the lens. For a biconcave lens the minimum thickness occurs when the two faces meet on the axis of revolution. If a positive value of Thickness is specified it must be greater than the minimum thickness to take effect. See also the remarks below.

Surface File 1 (*surface1_file*) [file name].

Name of file with tabulated data containing a numerical description of face 1 of the lens. If a file name is given the data in the file takes

precedence over `r1` and `bs1` for defining face 1. The format of the file is described in the section *Rotationally Symmetric Surface*.

Surface File 2 (`surface2_file`) [file name].

Name of file with tabulated data containing a numerical description of face 2 of the lens. If a file name is given the data in the file takes precedence over `r2` and `bs2` for defining face 2. The format of the file is described in the section *Rotationally Symmetric Surface*.

Length Unit in Files (`length_unit_in_files`) [item from a list of character strings], default: **m**.

Defines the unit of length (mm, cm, m, km, in, ft) for the data in the files Surface File 1 and Surface File 2.

Remarks

Geometry of a lens

In Figure 1 below a biconvex and a plano-convex lens are shown

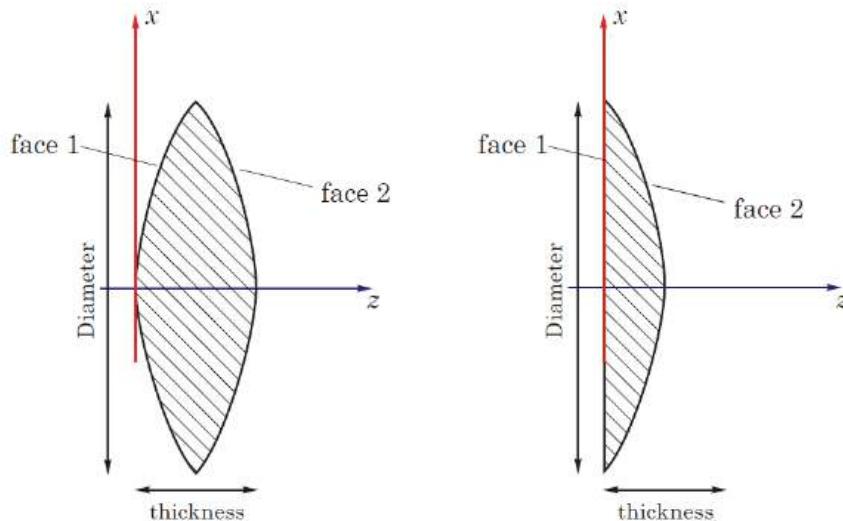


Figure 1 Biconvex lens (left) and plano-convex lens (right).

The figure illustrates the orientation of the lens relative to its coordinate system. The z -axis is the axis of revolution, and the centre point of face 1 is located at the origin of the coordinate system. It follows that the xy -plane will always be a tangent plane to the lens. Furthermore, the centre point of face 2 is located on the positive z -axis. In Figure 1 the two faces of the lenses meet at the rim so that the thickness is as small as possible for the given diameter and shape of the faces. If `Thickness` is specified to a positive value, it must be larger than the minimum value and has the effect shown in Figure 2 (left).

The thickness is measured along the axis of rotation. For a biconcave lens the minimum thickness becomes zero. The following Figure 3 illustrates a convex lens with diameter 100 mm and `Thickness` specified to 5 mm.

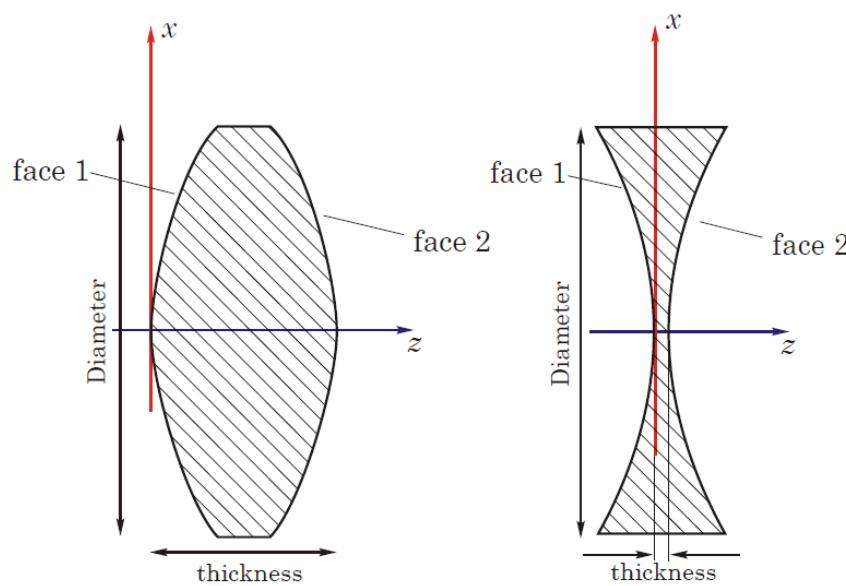


Figure 2 Biconcave thick lens (left) and convex lens with nonzero thickness (right).

Specification of the faces of a lens

The lens faces may be given either analytically or by tabulated numerical data on files as described in the attributes section. It is possible to specify one face analytically and the other numerically.

Each face is specified in a coordinate system with origin at the centre of the face and the z -axis coinciding with the axis of revolution. The z -axis is directed from the face towards the interior of the lens. The x -axis is parallel to the x -axis of the lens coordinate system defined in Figure 1. The coordinates are denoted (x_1, y_1, z_1) and (x_2, y_2, z_2) for face 1 and face 2, respectively. Thus, coordinate system (x_1, y_1, z_1) is identical to the lens coordinate system, and coordinate system (x_2, y_2, z_2) has origin on the opposite face and the y - and z -axes are reversed with respect to the lens coordinate system, Figure 3.

Analytical surface descriptions

If the face is described analytically, the surface is defined by the radius of curvature R on the z -axis and the conic constant b_s (attributes $r1, bs1$ for face 1 and $r2, bs2$ for face 2). If $R > 0$ the face becomes convex seen from the exterior of the lens, and if $R < 0$ the face becomes concave. The value $R = 0$ has the special meaning that the face is planar. For $R \neq 0$ the face becomes a conic surface (a quadric surface of revolution with two focal points) with eccentricity e given by $b_s = -e^2$ so that

$$b_s = 0 \quad \text{specifies a spherical surface} \tag{1}$$

$$-1 < b_s < 0 \quad \text{specifies a ellipsoidal surface} \tag{2}$$

$$b_s = -1 \quad \text{specifies a paraboloidal surface} \tag{3}$$

$$b_s < -1 \quad \text{specifies a hyperboloidal surface} \tag{4}$$

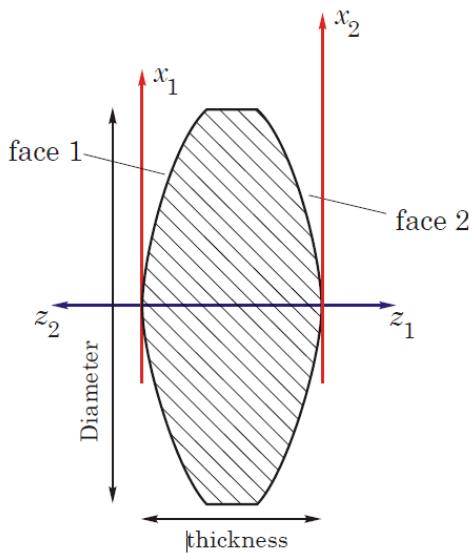


Figure 3

Coordinate systems for the two faces of the lens.

The explicit surface expression for the face is given by:

$$z = \frac{\kappa \rho^2}{1 + \sqrt{1 - (1 + b_s) \kappa^2 \rho^2}} \quad (5)$$

with

$$\kappa = \frac{1}{R}, \quad \text{and} \quad \rho = \sqrt{x^2 + y^2} \quad (6)$$

which can be shown to define a conic surface. In these expressions (x, y, z) shall be replaced by (x_1, y_1, z_1) for face 1 and by (x_2, y_2, z_2) for face 2.

Tabulated surfaces

The surfaces of a lens may be described numerically in two files, one for each surface. Each file contains a table of points (ρ, z) defining the rotationally symmetric surface, cf. Section *Rotationally Symmetric Surface*. The data are interpolated to give a smooth surface of the form $z = f(\rho)$ that pass through the given points. The first point must be $(\rho, z) = (0, 0)$ such that the surface passes through the origin of the coordinate system. Coordinate system $x_1y_1z_1$ shall be applied for face 1 and coordinate system $x_2y_2z_2$ shall be applied for face 2, see the definitions of the coordinate systems above.

MARTIN PUPLETT INTERFEROMETER (martin_puplett_interferometer)

Purpose

Objects of the class *Martin Puplett Interferometer* are used in the Frame Design Tool to model a Martin Puplett Interferometer. It is a rather complicated 4-port component consisting of two roof-top mirrors and three polarization grids. Because of its complexity it is recommended to define, modify and analyse this component through the Frame Design Tool.

This class is only available with the Quast add-on.

Links

Classes→*Geometrical Objects*→*Scatterer*→*Martin Puplett Interferometer*

Remarks

Syntax

```
<object name> martin_puplett_interferometer
(
    center_polarizer           : ref(<n>),
    roof_top_mirror_a          : ref(<n>),
    roof_top_mirror_b          : ref(<n>),
    port_a_polarizer           : ref(<n>),
    port_b_polarizer           : ref(<n>)
)
where
<n> = name of an object
```

Attributes

Center Polarizer (*center_polarizer*) [name of an object].

Reference to an object of class *Reflector*. In a Martin-Puplett interferometer this reflector must be a polarization grid. See the remarks below.

Roof Top Mirror A (*roof_top_mirror_a*) [name of an object].

Reference to an object of class *Rooftop Mirror*.

Roof Top Mirror B (*roof_top_mirror_b*) [name of an object].

Reference to an object of class *Rooftop Mirror*.

Port A Polarizer (*port_a_polarizer*) [name of an object].

Reference to an object of class *Reflector*. In a Martin-Puplett interferometer this reflector must be a polarization grid

Port B Polarizer (*port_b_polarizer*) [name of an object].

Reference to an object of class *Reflector*. In a Martin-Puplett interferometer this reflector must be a polarization grid

Command Types

The available commands for generating plots are described in [Plot Settings \(Obsolete\)](#).

Remarks

The Martin-Puplett interferometer is implemented in the Frame Design Tool (see the GRASP documentation). It is a rather complicated 4-port component consisting of two roof-top mirrors and three polarization grids. It is recommended to define, modify and analyse this component through the Frame Design Tool. A schematic drawing is shown below in Figure 1.

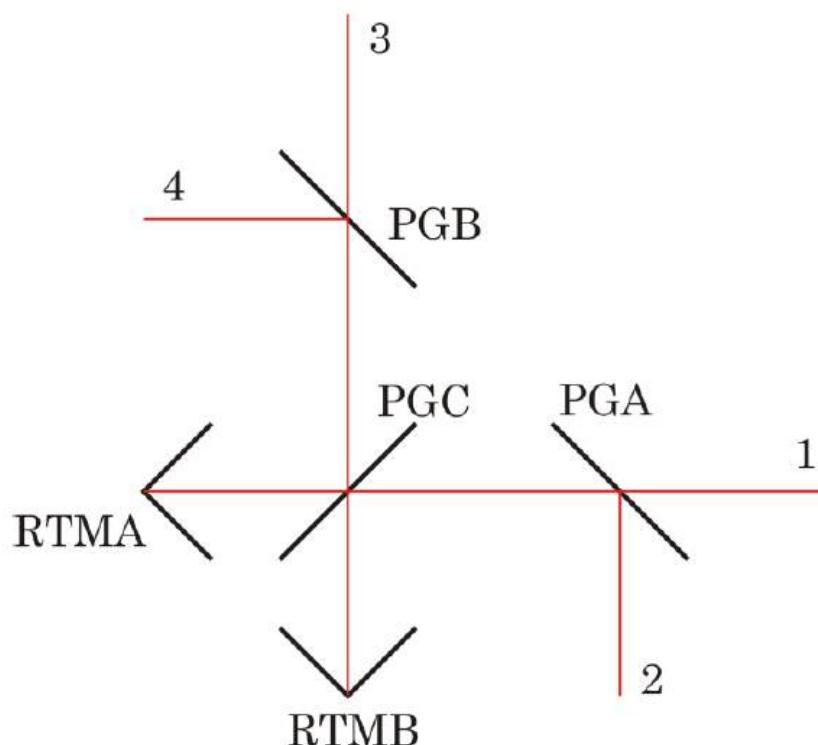


Figure 1 Schematic view of Martin-Puplett interferometer.

The four ports are numbered from 1 to 4 and the two roof-top mirrors are denoted RTMA and RTMB. The components PGA and PGB are polarization grids and the wires in the grid can be either parallel or orthogonal to the plane of the paper. Finally, the central component PGC is also a polarization grid in which the wires, see from a beam direction, makes an angle of 45° with the plane of the paper. The effect of this arrangement is that input signals coupled to the ports 1 and 2 will be combined and transmitted through ports 3 and 4 without any return loss back to port No. 1 and 2. It is possible to adjust the positions of the roof-top mirrors along their respective beam directions by which the combined signal can be directed solely to either ports 3 or 4. For a detailed discussion of this interferometer the reader is referred to the book P.F. Goldsmith (P.F. Goldsmith, 'Quasioptical Systems', IEEE press, New York 1998.)

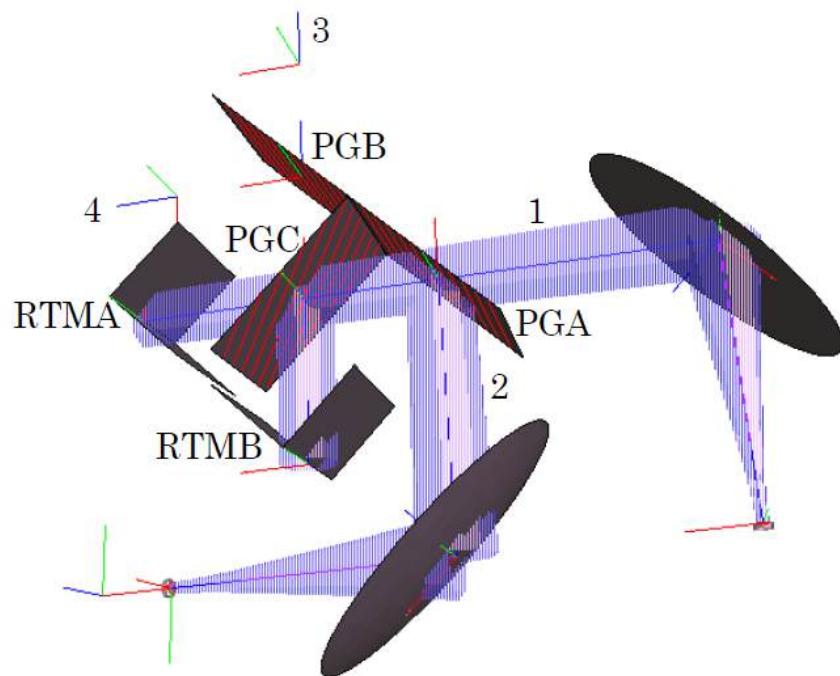


Figure 2

View of Martin-Puplett interferometer. The ports are numbered from 1 through 4 as in Figure 1. Beams from two feeds are fed into the interferometer through ports 1 and 2 and shown as beam tubes in light blue.

SURFACE

Purpose

This class describes, in conjunction with a *Rim*, the *Surface* of a *Reflector*, which is a special type of a *Scatterer*. The following surface types are available.

Simple reflector surfaces:

Paraboloid

Hyperboloid

Ellipsoid

Plane

Conic Mirror Surface

More complex surfaces given by a second order polynomial are found under:

Other Quadrics

Surfaces given in a tabulated form are defined under:

Tabulated Surface

Surface deviations may be added by a

Imperfection Modelling

Other surface types are:

Unfurlable Surface

Pyramidal Surface

Links

Classes→*Geometrical Objects*→*Surface*

PARABOLOID (paraboloid)

Purpose

The class *Paraboloid* defines a paraboloidal surface, obtained by rotating a parabola around its axis. The vertex of the paraboloid is the point where its axis intersects the paraboloid.

Links

Classes→*Geometrical Objects*→*Surface*→*Paraboloid*

Remarks

Syntax

```
<object name> paraboloid
(
    focal_length           : <rl>,
    vertex                 : struct(x:<rl>, y:<rl>, z:<rl>)
)
where
<rl> = real number with unit of length
```

Attributes

Focal Length (*focal_length*) [real number with unit of length].

Focal length of paraboloid. i.e. the distance from the focal point to the vertex.

Vertex (*vertex*) [struct].

Defines the position of the vertex in the reflector coordinate system.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of the vertex.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of the vertex.

z (*z*) [real number with unit of length], default: **0**.

z-coordinate of the vertex.

Remarks

The surface is defined in the reflector coordinate system by the equation

$$z = \frac{(x - x_0)^2 + (y - y_0)^2}{4f} + z_0$$

where *f* is the *focal_length* and *x₀*, *y₀* and *z₀* are the coordinates of the vertex. The axis of the paraboloid is parallel to the *z*-axis, and the opening of the paraboloid is in the direction of positive *z*.

HYPERBOLOID (hyperboloid)

Purpose

The class *Hyperboloid* defines a hyperboloidal surface, obtained by rotating a hyperbola around the axis through its two foci. The intersections between this axis and the two branches of the hyperbola are the two vertex points.

Links

Classes→*Geometrical Objects*→*Surface*→*Hyperboloid*

Remarks

Syntax

```
<object name> hyperboloid
(
    vertex_distance : <rl>,
    foci_distance : <rl>
)
where
<rl> = real number with unit of length
```

Attributes

Vertex Distance (*vertex_distance*) [real number with unit of length].

Distance between the two vertices.

Foci Distance (*foci_distance*) [real number with unit of length].

Distance between the foci.

Remarks

The hyperboloidal surface is defined by

$$\frac{(z - c)^2}{a^2} - \frac{x^2 + y^2}{b^2} = 1 \quad (1)$$

in the coordinate system in which the reflector is specified. Here, a is half the *vertex_distance* and c is half the *foci_distance* and

$$b = \sqrt{c^2 - a^2} \quad (2)$$

The eccentricity, e , of the hyperboloid is calculated as $e = c/a$.

The surface is rotationally symmetric around the z -axis.

In general, a surface must be given by a function of the form $z = f(x, y)$. Of the two solutions that can be obtained from Eq. (1) the following function is used:

$$z = c - \frac{a}{b} \sqrt{b^2 + x^2 + y^2} \quad (3)$$

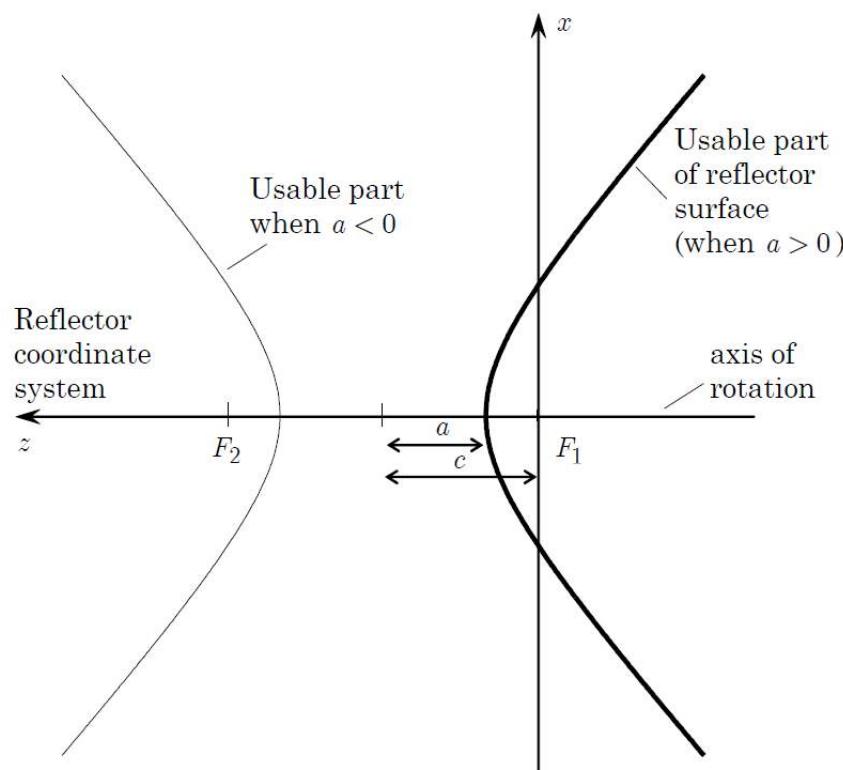


Figure 1

Sketch of a hyperboloid. Rays from the one focal point will reflect as coming from the other focal point. The *vertex_distance* is $2a$ and the *foci_distance* is $2c$.

This equation gives the positive sheet of the hyperboloid expressed in a coordinate system with its origin at F_1 . The positive sheet of the hyperboloid is shown in heavy line in the figure.

This definition corresponds to the 'normal' hyperboloidal subreflector as used in a Cassegrainian dual reflector system. The feed shall then be placed at the focal point F_2 , and the field scattered from the subreflector will have a virtual focal point (phase centre) at F_1 , which shall coincide with the main reflector focal point.

If the other, negative, sheet of the hyperboloid shall be used as reflector, this may be obtained by specifying a negative value of the half vertex distance, a . With the feed still at F_2 , the field scattered from this part of the hyperboloid will also have a virtual focal point at F_1 .

ELLIPSOID (ellipsoid)

Purpose

The class *Ellipsoid* defines an ellipsoidal surface, obtained by rotating an ellipse around its major axis.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→*Ellipsoid*

Remarks

Syntax

```
<object name> ellipsoid
(
    vertex_distance      : <rl>,
    foci_distance        : <rl>,
    axis_angle           : <r>
)
where
<r> = real number
<rl> = real number with unit of length
```

Attributes

Vertex Distance (*vertex_distance*) [real number with unit of length].

Distance between the two vertices, i.e. the ellipsoid major axis.

Foci Distance (*foci_distance*) [real number with unit of length].

Distance between the foci. The Foci Distance shall be specified greater than zero. A spherical surface may be specified by class [Spherical Surface](#).

Axis Angle (*axis_angle*) [real number], default: **0**.

Angle (in degrees) for rotation of the ellipsoid, negative from *z*-axis towards *x*-axis (see the remarks below).

Remarks

The surface is defined by an ellipsoid, where a is half the Vertex Distance and c half the Foci Distance. The ellipsoid is placed with one focal point (F_1 in Figure 1) at the origin of the coordinate system in which the reflector is specified.

The eccentricity, e , of the ellipsoid is calculated as $e = c/a$.

Rays from the one focal point will reflect towards the other focal point. The Vertex Distance is $2a$ and the Foci Distance is $2c$.

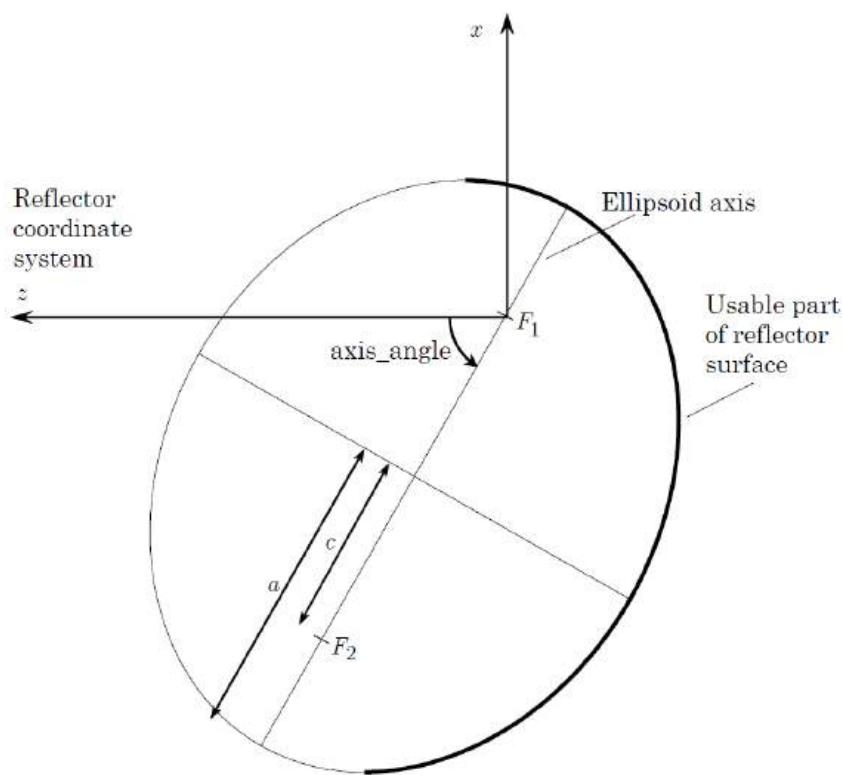


Figure 1

Sketch of an ellipsoid. The ellipsoid axis may be rotated an angle, Axis Angle, around F_1 . The other focal point, F_2 , is on the positive z -axis of the reflector coordinate system before the rotation. A positive rotation moves F_2 towards the negative x -axis. The usable part of the ellipsoid is the section with its concave side in the direction of the positive z -axis, cf. the figure. The applied part of the ellipsoid is that defined by its *Rim*.

PLANE (plane)

Purpose

This class defines the surface of a plane.

Note: Some simple *Reflectors* with a plane *Surface* may be modelled as a *Plate*. Calculations with PO/PTD are then more accurate than when modelled as a plane *Reflector*. The difference is most important for electrically small plates, see *Plate* for more information and the Technical Description of GRASP (PTD) for details.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Plane](#)

[Remarks](#)

Syntax

```
<object name> plane
(
    point : struct(x:<rl>, y:<rl>, z:<rl>),
    normal : struct(x:<r>, y:<r>, z:<r>)
)
where
<r> = real number
<rl> = real number with unit of length
```

Attributes

Point (*point*) [struct].

A point on the plane.

x (x) [real number with unit of length], default: **0**.

x-coordinate of the point.

y (y) [real number with unit of length], default: **0**.

y-coordinate of the point.

z (z) [real number with unit of length], default: **0**.

z-coordinate of the point.

Normal (*normal*) [struct].

Vector in the direction of a normal to the plane. To produce a surface, $z = f(x, y)$, the normal must not be parallel to the xy -plane, i.e. it must have a non-zero z -component.

x (x) [real number], default: **0**.

x-coordinate of the vector normal.

y (y) [real number], default: **0**.

y-coordinate of the vector normal.

z (z) [real number], default: **1**.

z-coordinate of the vector normal.

Remarks

The polynomial $z = f(x, y)$ is defined by

$$z = \frac{-n_x(x - p_x) - n_y(y - p_y)}{n_z} + p_z \quad (1)$$

where the attributes *point* and *normal* define

$$(p_x, p_y, p_z) = \textit{point} \quad (2)$$

$$(n_x, n_y, n_z) = \textit{normal} \quad (3)$$

The surface is defined in the coordinate system of the reflector object that uses the surface. The front side of the plane will, as for any reflector surface, be the side for which its normal has a positive component along the z -axis of the reflector coordinate system. Therefore the front side does not depend on the direction of the normal (specified by the attribute *normal*) which defines the plane.

CONIC MIRROR SURFACE (conical_surface)

Purpose

This class defines ellipsoidal, paraboloidal and hyperboloidal surfaces of revolution in an alternative way to the classic definitions in the *Ellipsoid*, *Hyperboloid* and *Paraboloid* surface classes. The definition is commonly used in the design of beam waveguides and quasi-optical networks, and is extensively used by the Frame design Tool.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Conic Mirror Surface](#)

Remarks

Syntax

```
<object name> conical_surface
(
    r1                      : <rl>,
    r2                      : <rl>,
    theta_i                  : <r>,
    theta_n                  : <r>
)
where
<r> = real number
<rl> = real number with unit of length
```

Attributes

R1 (*r1*) [real number with unit of length].

The distance from the reflection point to focal point No. 1 measured with sign along the axial ray (negative when the focal point is on the continuation of the ray behind the reflector). When the focal point is at infinite *r1* shall be specified to zero. See the remarks below for definition of the axial ray and the numbering of the focal points.

R2 (*r2*) [real number with unit of length].

The distance from the reflection point to focal point No. 2 measured with sign along the axial ray (negative when the focal point is on the continuation of the ray behind the reflector). When the focal point is at infinite *r2* shall be specified to zero.

theta I (*theta_i*) [real number].

The angle from the surface normal to the axial ray, $0^\circ \leq \text{theta_i} < 90^\circ$

theta N (*theta_n*) [real number].

The direction of the surface normal at the reflection point given as the angle from the *z*-axis to the normal, measured positive towards the *x*-axis (the normal is restricted to the *zx*-plane).

Remarks

The surface definition

Because the definition of the surface is closely related to Gaussian beam tracing in quasi-optical beam waveguides, the conic mirror surface is defined with respect to an axial ray. This axial ray is incident and reflected at a central point, R , on the surface and the angle of incidence is theta_i . The surface normal \hat{n} is for definition purpose pointed into the half-space of the axial ray. R is at the origin of the coordinate system in which the surface is given, and the two foci are located in the xz -plane, see Figure 1.

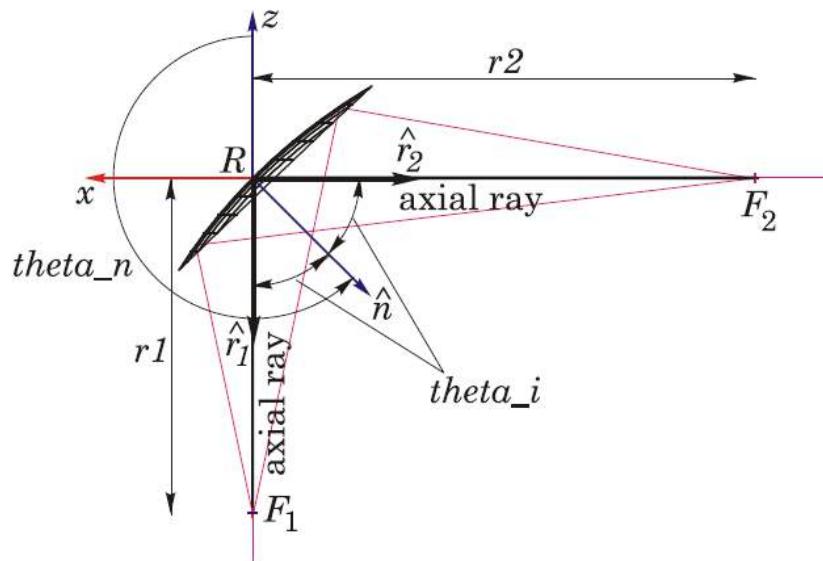


Figure 1

The attributes for defining a *Conic Mirror Surface* when $r1 > 0$ and $r2 > 0$; theta_n is positive. In order to obtain the correct surface it is always important to perform the following check: The rotation from \hat{r}_1 over \hat{n} to \hat{r}_2 is positive around the y -axis (counter clockwise in this figure) and less than 180° .

We define from R directions along the axial ray by the unit vectors \hat{r}_1 and \hat{r}_2 such that the angle from \hat{r}_1 to \hat{r}_2 is positive and less than 180° when measured in a direction from the z -axis towards the x -axis (i.e. positive around the y -axis).

The foci, F_1 and F_2 , are located at distances $r1$ and $r2$ along \hat{r}_1 and \hat{r}_2 , respectively. A negative distance indicates that the focal point is behind the surface relative to the axial ray, see Figure 2. Thus

$$\begin{aligned} RF_1 &= r1 \hat{r}_1 \\ RF_2 &= r2 \hat{r}_2 \end{aligned}$$

It should be noted that the orientation of the above normal, \hat{n} , is strictly related to the definition here. In calculations where the surface normal is of importance, for example when placing different layers of surface materials

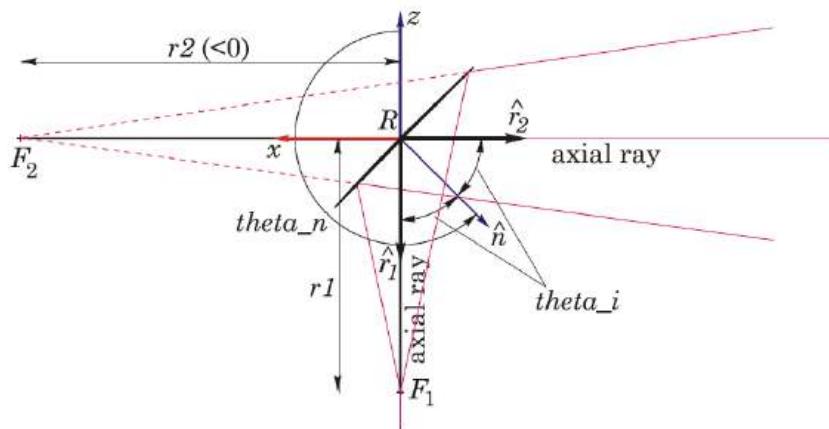


Figure 2

The attributes for defining the *Conic Mirror Surface* when one focal distance (here r_2) is negative. Check: The rotation from \hat{r}_1 over \hat{n} to \hat{r}_2 is positive around the y -axis (counter clockwise in this figure) and less than 180° .

on a reflector, the direction of the normal is always towards the half-space $z > 0$. This still applies for the present surface.

The axial ray in direction of focus F_1 (along \hat{r}_1) will pass through the focal point when $r_1 > 0$ but when $r_1 < 0$ the focal point is on the other side of the reflector. In the same way an axial ray in direction of focus F_2 (along \hat{r}_2) will pass through the focal point when $r_2 > 0$, cf. Figure 1. The focal point is on the other side of the reflector when $r_2 < 0$, Figure 2.

Objects of this class will typically be generated by the Frame design Tool. the *Conic Mirror Surface* will belong to a *Reflector* which in such cases is defined in a coordinate system oriented such that the rim of the reflector is plane and parallel to the xy -plane. This implies that the angle θ_n is small. The *Rim* of a *Reflector* is always described in the xy -plane of the reflector coordinate system but in general the rim is not parallel to the xy -plane, see the examples below.

The defined surface is, according to the definition of the two foci, a conical surface which is rotationally symmetric around the line connecting the two focal points.

Examples

The sign of r_1 and r_2 determines the position of the focal points relative to the surface. When both r_1 and r_2 are positive the foci are both on the positive side of the surface as defined by the direction of \hat{n} . With a feed in the first focus, F_1 , the field will be reflected and concentrated at the second focus, F_2 . The surface is an ellipsoid illuminated on its concave side as shown in Figure 3. The ellipsoidal surface is here outlined as an ellipse which shall be rotated around its axis F_1F_2 to generate the full surface. A reflector is drawn in the region of R .

If both r_1 and r_2 are negative the foci are behind the surface with respect to \hat{n} and the fields. The surface is an ellipsoid illuminated on its convex side. An incoming field focused towards F_1 it will be reflected as if it originates in F_2 .

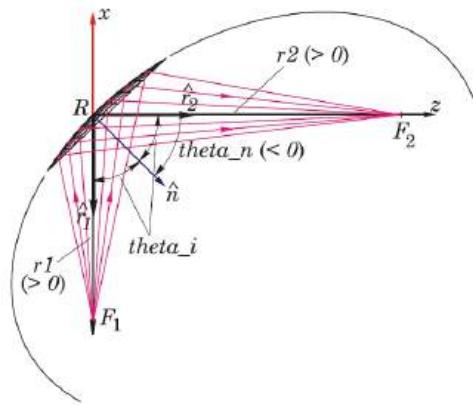


Figure 3

Ellipsoid illuminated on its concave side, focal points F_1 and F_2 . Rays radiated from a feed at F_1 will optically be reflected and converge at F_2 . The normal \hat{n} is positive on the concave side and $r_1 = 40\lambda$ and $r_2 = 60\lambda$. The z -axis has been chosen to point towards F_2 whereby $\text{theta}_n = -45^\circ$. The angle of incidence for the axial ray is $\text{theta}_i = 45^\circ$. Check: The rotation from \hat{r}_1 over \hat{n} to \hat{r}_2 is positive around the y -axis (counter clockwise in this figure) and less than 180° .

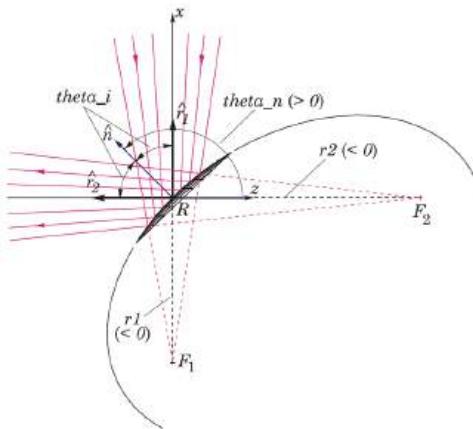


Figure 4

The same ellipsoid as in Figure 3 illuminated on its convex side, focal points F_1 and F_2 . Converging rays radiated towards F_1 will optically be reflected in directions as if the rays originate in F_2 . The normal \hat{n} is positive on the convex side and r_1 and r_2 are both negative, $r_1 = -40\lambda$ and $r_2 = -60\lambda$. The z -axis has been chosen to point towards F_2 whereby $\text{theta}_n = 135^\circ$. The angle of incidence for the central ray is $\text{theta}_i = 45^\circ$. Check: The rotation from \hat{r}_1 over \hat{n} to \hat{r}_2 is positive around the y -axis (counter clockwise in this figure) and less than 180° .

If one of r_1 or r_2 is zero the corresponding focal point is infinitely far away and the surface is a paraboloid. The sign of the non-zero distance determines if it is the concave or the convex side which is illuminated. The full paraboloidal surface is generated by rotating the outlined parabola around

its axis.

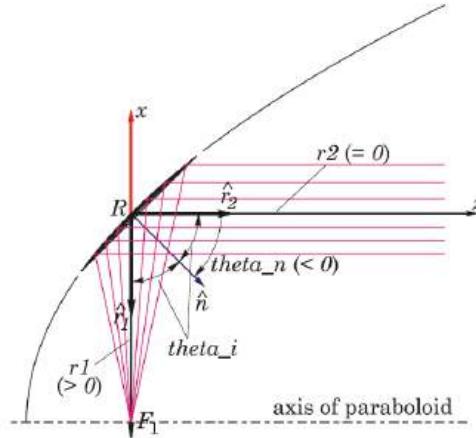


Figure 5

Paraboloid illuminated on its concave side, focal points F_1 . Rays radiated from a feed at F_1 will optically be reflected and result in rays parallel to the axis of the paraboloid. The normal \hat{n} is positive on the concave side and $r_1 = 40\lambda$. As F_2 is infinitely far then $r_2 = 0$. The z -axis has been chosen to point along the parabola axis whereby $\theta_n = -45^\circ$. The angle of incidence for the central ray is $\theta_i = 45^\circ$. Check: The rotation from \hat{r}_1 over \hat{n} to \hat{r}_2 is positive around the y -axis (counter clockwise in this figure) and less than 180° .

If r_1 and r_2 have different signs, then the foci are on opposite sides of the surface which implies that the surface is hyperboloidal. In that case the reflection will occur in the branch closest to F_1 when $|r_1| < |r_2|$ and in the branch closest to F_2 when $|r_2| < |r_1|$.

The hyperboloid defined in Figure 6 is generated by rotating the two branches of the shown hyperbola around the axis F_1F_2 .

An example in which the far branch of the hyperboloid is illuminated is shown in Figure 7. The defined surface is the same as in the preceding example but now with the feed at focal point F_1 . As $|r_2| < |r_1|$ it is again the branch closest to F_2 which is reflecting. It will be the convex side of the hyperboloid which is illuminated so the normal must be shifted to that side.

The hyperboloid is again generated by rotating the two branches of the shown hyperbola around the axis F_1F_2 . As it is the far (from F_1) branch of the hyperboloid which is illuminated the near branch is not active.

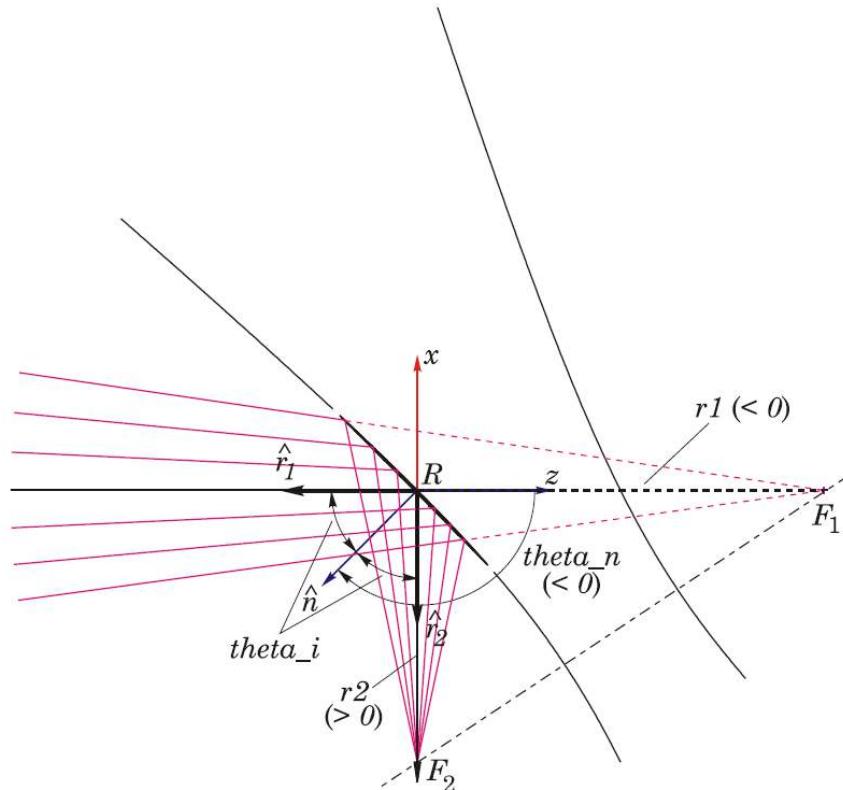


Figure 6

A hyperboloid illuminated from focal point F_2 , $r_2 > 0$. The rays are reflected in directions as if they originate in the other focal point F_1 (thus $r_1 < 0$). As $|r_2| < |r_1|$ it is the branch closest to F_2 which is reflecting. The normal \hat{n} is positive on the concave side and $r_1 = -60\lambda$ and $r_2 = 40\lambda$. The z -axis has been chosen to point towards F_1 whereby $\theta_n = -135^\circ$. The angle of incidence for the central ray is $\theta_i = 45^\circ$. Check: The rotation from \hat{r}_1 over \hat{n} to \hat{r}_2 is positive around the y -axis (counter clockwise in this figure) and less than 180° .

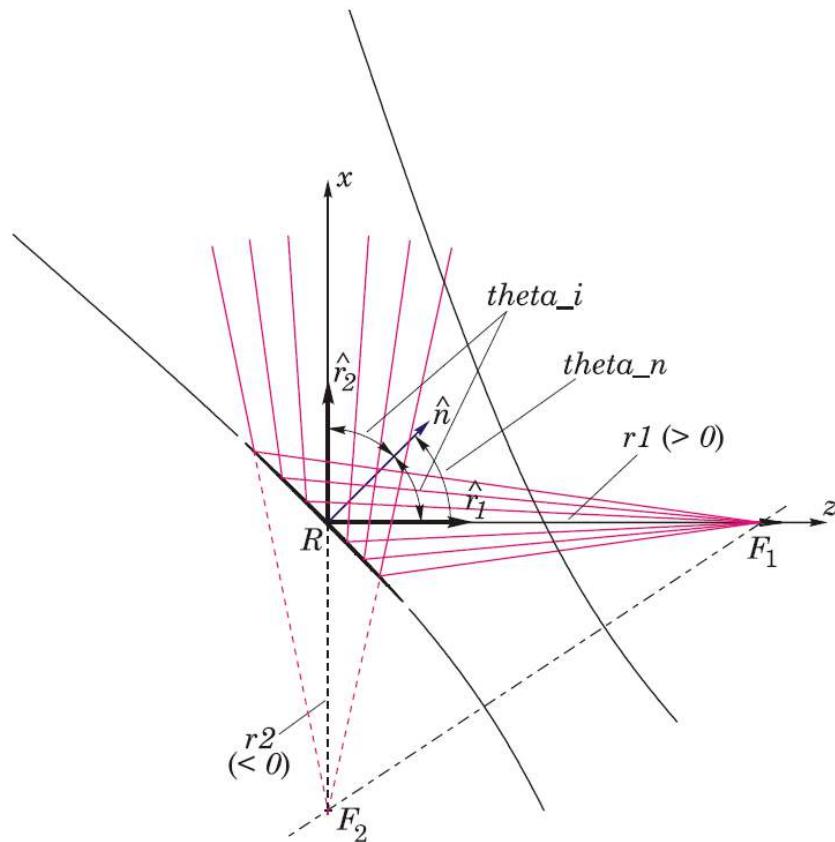


Figure 7

A hyperboloid illuminated from focal point F_1 and thus $r1 > 0$. The rays are reflected in directions as if they originate in the other focal point F_2 , thus $r2 < 0$. As $|r2| < |r1|$ it is the branch closest to F_2 which is reflecting. The normal \hat{n} is positive on the convex side and $r1 = 60\lambda$ and $r2 = -40\lambda$. The z -axis has been chosen to point towards F_1 whereby $\text{theta_n} = 45^\circ$. The angle of incidence for the central ray is $\text{theta_i} = 45^\circ$. Check: The rotation from \hat{r}_1 over \hat{n} to \hat{r}_2 is positive around the y -axis (counter clockwise in this figure) and less than 180° .

OTHER QUADRICS

Purpose

The menu *Other Quadrics* consists of a group of surfaces (within the class *Surface*), described by a second order polynomial. The polynomial determines the surface z as a function of x and y . The following types of *Other Quadrics* are available.

Spherical Surface

Circular Cone

Circular Cylinder Surface

Second Order Polynomial

Polynomial (Obsolete) (obsolete, use *Second Order Polynomial* instead)

Links

Classes→*Geometrical Objects*→*Surface*→*Other Quadrics*

SPHERICAL SURFACE (*spherical_surface*)

Purpose

This class defines the surface of a sphere.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Other Quadrics](#)→[Spherical Surface](#)

Remarks

Syntax

```
<object name> spherical_surface
(
    radius : <rl>,
    origin : struct(x:<rl>, y:<rl>, z:<rl>),
    largest_root : <si>
)
where
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Radius (*radius*) [real number with unit of length].

The radius of the sphere.

Origin (*origin*) [struct].

Origin of the sphere.

x (x) [real number with unit of length], default: **0**.

x-coordinate of the point.

y (y) [real number with unit of length], default: **0**.

y-coordinate of the point.

z (z) [real number with unit of length], default: **0**.

z-coordinate of the point.

Largest Root (*largest_root*) [item from a list of character strings], default: **on**.

The sphere will have two surface values for each (x, y) , corresponding to two solutions of the second order equation $z = f(x, y)$, i.e. two roots for z are possible.

on

The larger root is selected.

off

The smaller root is selected.

Remarks

The spherical surface is defined by

$$z = \pm \sqrt{r^2 - (x - O_x)^2 - (y - O_y)^2} + O_z$$

where r is given by the attribute *radius*,

$$r = \textit{radius}$$

and the attribute *origin* defines (O_x, O_y, O_z)

$$(O_x, O_y, O_z) = \textit{origin}$$

A surface must be given by a function of the form $z = f(x, y)$, but the equation above has two solutions for z . Either the smallest or the largest solution must be selected by the *largest_root* attribute.

The surface is defined in the coordinate system of the reflector object that uses the surface.

CIRCULAR CONE (circular_cone)

Purpose

This class defines the surface of a circular cone.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Other Quadrics](#)→[Circular Cone](#)

Remarks

Syntax

```
<object name> circular_cone
(
    half_cone_angle           : <r>,
    vertex                    : struct(x:<rl>, y:<rl>, z:<rl>),
    axis                      : struct(x:<r>, y:<r>, z:<r>),
    largest_root              : <si>
)
where
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Half Cone Angle (*half_cone_angle*) [real number].

Half the opening angle of the cone, in degrees.

Vertex (*vertex*) [struct].

Defines the position of the cone vertex in the reflector coordinate system.

x (x) [real number with unit of length], default: **0**.
x-coordinate of the vertex.

y (y) [real number with unit of length], default: **0**.
y-coordinate of the vertex.

z (z) [real number with unit of length], default: **0**.
z-coordinate of the vertex.

Axis (*axis*) [struct].

Vector in the direction of the cone axis.

x (x) [real number], default: **0**.
x-coordinate of the direction vector.
y (y) [real number], default: **0**.
y-coordinate of the direction vector.

z (z) [real number], default: **1**.

z-coordinate of the direction vector.

Largest Root (*largest_root*) [item from a list of character strings], default: **on**.

The cone will have two surface values for each (x, y) , corresponding to two solutions of the second order equation $z = f(x, y)$, i.e. two roots for z are possible.

on

The larger root is selected.

off

The smaller root is selected.

Remarks

In a coordinate system (x', y', z') , with vertex in the origin and the z' -axis as the cone axis, the circular cone surface is defined by

$$z' = \pm \frac{\sqrt{x'^2 + y'^2}}{\tan \alpha}$$

where

$$\alpha = \text{half_cone_angle}$$

Transformed to the reflector coordinate system (x, y, z) , the equation is changed to one of the type:

$$A_{zz}z_v^2 + A_{xz}x_vz_v + A_{yz}y_vz_v = A_{xx}x_v^2 + A_{xy}x_vy_v + A_{yy}y_v^2 \quad (1)$$

where

$$(x_v, y_v, z_v) = (x - V_x, y - V_y, z - V_z) \quad (2)$$

and

$$\text{vertex} = (V_x, V_y, V_z) \quad (3)$$

A surface must be given by a function of the form $z = f(x, y)$, but the equation above will usually have two solutions for z . Either the smallest or the largest solution must be selected by the attribute *largest_root*.

In the example above with the cone axis parallel to the z -axis, '*largest_root: on*' will result in the cone with its opening towards positive z ; while '*largest_root: off*' results in the cone with the opening towards negative z .

The surface is defined in the coordinate system of the reflector object that uses the surface.

CIRCULAR CYLINDER SURFACE (circular_cylinder_surface)

Purpose

This class defines the surface of a circular cylinder.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Other Quadrics](#)→[Circular Cylinder Surface](#)

Remarks

Syntax

```
<object name> circular_cylinder_surface
(
    radius : <rl>,
    origin : struct(x:<rl>, y:<rl>, z:<rl>),
    axis : struct(x:<r>, y:<r>, z:<r>),
    largest_root : <si>
)
where
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Radius (*radius*) [real number with unit of length].

The radius of the circular cross-section.

Origin (*origin*) [struct].

A point on the axis of the circular cylinder.

x (x) [real number with unit of length], default: **0**.
x-coordinate of the origin.

y (y) [real number with unit of length], default: **0**.
y-coordinate of the origin.

z (z) [real number with unit of length], default: **0**.
z-coordinate of the origin.

Axis (*axis*) [struct].

Vector in the direction of the axis of the cylinder. To produce a surface, $z = f(x, y)$, the axis must not be parallel to the z -axis.

x (x) [real number], default: **1**.
x-coordinate of the direction vector.
y (y) [real number], default: **0**.
y-coordinate of the direction vector.

z (z) [real number], default: **0**.

z-coordinate of the direction vector.

Largest Root (*largest_root*) [item from a list of character strings], default: **on**.

The cylinder will have two surface values for each (x, y) , corresponding to two solutions of the second order equation $z = f(x, y)$, i.e. two roots for z are possible.

on

The larger root is selected.

off

The smaller root is selected.

Remarks

Define a coordinate system (x', y', z') , with origin and z -axis as specified by the attributes *origin* and *axis*. In this coordinate system the circular cylinder surface is defined by

$$r^2 = x'^2 + y'^2$$

where r is given by the attribute *radius*,

$$r = \text{radius} \tag{1}$$

Transformed to the reflector coordinate system (x, y, z) , the equation is changed to one of the type:

$$A_{zz}z_o^2 + A_{xz}x_oz_o + A_{yz}y_oz_o = A_{xx}x_o^2 + A_{xy}x_oy_o + A_{yy}y_o^2 + A_c$$

where

$$(x_o, y_o, z_o) = (x - O_x, y - O_y, z - O_z)$$

and

$$\text{origin} = (O_x, O_y, O_z) \tag{2}$$

A surface must be given by a function of the form $z = f(x, y)$, but the equation above will usually have two solutions for z . Either the smallest or the largest solution must be selected by the *largest_root* attribute.

The surface is defined in the coordinate system of the reflector object that uses the surface.

SECOND ORDER POLYNOMIAL (polynomial_2nd_order)

Purpose

This class defines a surface by a general second order polynomial.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Other Quadrics](#)→[Second Order Polynomial](#)

Remarks

Syntax

```
<object name> polynomial_2nd_order
(
    coefficients : struct(xx:<r>, xy:<r>, yy:<r>, x:<r>,
                           y:<r>, c:<r>, zz:<r>, z:<r>,
                           xz:<r>, yz:<r>, unit:<si>),
    origin : struct(x:<rl>, y:<rl>, z:<rl>),
    rotations : sequence(
        struct(around:<si>,
               angle:<r>),
        ...),
    largest_root : <si>,
    scale_factor : <r>
)
where
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Coefficients (*coefficients*) [struct].

Coefficients of a second order polynomial, which define the surface (see the remarks below):

xx (xx) [real number].

Coefficient A_{xx} to x^2 .

xy (xy) [real number].

Coefficient A_{xy} to xy .

yy (yy) [real number].

Coefficient A_{yy} to y^2 .

x (x) [real number].

Coefficient A_x to x .

y (y) [real number].

Coefficient A_y to y .

c (c) [real number].

The constant A_c in the polynomial.

zz (zz) [real number].

Coefficient A_{zz} to z^2 .

z (z) [real number].

Coefficient A_z to z .

xz (xz) [real number].

Coefficient A_{xz} to xz .

yz (yz) [real number].

Coefficient A_{yz} to yz .

Unit (unit) [item from a list of character strings], default: **m**.

Defines the unit of length for x , y and z (mm, cm, m, km, in, ft).

Origin (origin) [struct].

Defines the coordinates (in the reflector coordinate system) of the origin of the coordinate system in which the polynomial is given.

x (x) [real number with unit of length], default: **0**.

x -coordinate of the origin.

y (y) [real number with unit of length], default: **0**.

y -coordinate of the origin.

z (z) [real number with unit of length], default: **0**.

z -coordinate of the origin.

Rotations (rotations) [sequence of structs].

A sequence of rotations of the polynomial. Each rotation is around an axis originally parallel to an axis of the reflector coordinate system, going through **Origin**. The order of rotations follows the order in the sequence. The type of rotation is specified by the struct member **around**, and the angle of rotation by the member **angle**.

Around (around) [item from a list of character strings], default: **x_original**.

Rotation around the specified axis.

x_original

Rotation around an axis parallel to the original x -axis, positive from the y -axis towards the z -axis.

y_original

Rotation around an axis parallel to the original y -axis, positive from the z -axis towards the x -axis.

z_original

Rotation around an axis parallel to the original z -axis, positive from the x -axis towards the y -axis.

x_rotated

Rotation around the new rotated x -axis, positive from the y -axis towards the z -axis.

y_rotated

Rotation around the new rotated y -axis, positive from the z -axis towards the x -axis.

`z_rotated`

Rotation around the new rotated z -axis, positive from the x -axis towards the y -axis.

Angle (*angle*) [real number], default: **0**.

The angle of rotation in degrees.

Largest Root (*largest_root*) [item from a list of character strings], default: **on**.

The solution of the second order equation may result in two values (roots) for z .

on

The larger root is selected.

off

The smaller root is selected.

Scale Factor (*scale_factor*) [real number], default: **1**.

The z -values will be scaled with this factor.

Remarks

The second order polynomial is defined by

$$A_{xx}x^2 + A_{xy}xy + A_{yy}y^2 + A_x x + A_y y + A_c = A_{zz}z^2 + A_z z + A_{xz}xz + A_{yz}yz$$

A surface must be given by a function of the form $z = f(x, y)$. However, for some values of the coefficients the equation above will have two solutions for z . Either the smallest or the largest solution must be selected by the Largest Root attribute. If the equation has only one solution the value of the Largest Root attribute is immaterial.

The surface is defined in the coordinate system of its corresponding reflector object. It is possible to displace and re-orient the surface relative to the reflector coordinate system by the Origin and Rotations attributes.

First, the coordinate system, in which the surface is specified, is translated in the reflector coordinate system so that its origin is positioned as specified by the Origin attribute. The axes of this translated coordinate system are those denoted the original axes.

Next, the rotations are carried out in the order as given in the sequence of the attribute Rotations. At each rotation, the coordinate system defining the surface is rotated around its origin. The rotations may be specified around either the original axes or the axes of the rotated system.

POLYNOMIAL (OBSOLETE) (polynomial)

Purpose

The class *Polynomial (Obsolete)* is an obsolete class used to define a surface given by a general second order polynomial.

It is included for backwards compatibility only. Instead it is recommended to use the more versatile class *Second Order Polynomial*.

Links

Classes→*Geometrical Objects*→*Surface*→*Other Quadrics*→*Polynomial (Obsolete)*

Remarks

Syntax

```
<object name> polynomial
(
    coefficients : struct(xx:<r>, xy:<r>, yy:<r>, x:<r>,
                           y:<r>, c:<r>, zz:<r>, z:<r>,
                           xz:<r>, yz:<r>, unit:<si>),
    y_rotation : <r>,
    z_rotation : <r>,
    translation : struct(x:<rl>, y:<rl>, z:<rl>),
    largest_root : <si>
)
where
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Coefficients (*coefficients*) [struct].

Coefficients of a second order polynomial, which define the surface (see the remarks below)

xx (xx) [real number].

Coefficient A_{xx} to x^2 .

xy (xy) [real number].

Coefficient A_{xy} to xy .

yy (yy) [real number].

Coefficient A_{yy} to y^2 .

x (x) [real number].

Coefficient A_x to x .

y (y) [real number].

Coefficient A_y to y .

c (c) [real number].

The constant A_c in the polynomial.

zz (zz) [real number].

Coefficient A_{zz} to z^2 .

z (z) [real number].

Coefficient A_z to z .

xz (xz) [real number].

Coefficient A_{xz} to xz .

yz (yz) [real number].

Coefficient A_{yz} to yz .

Unit (*unit*) [item from a list of character strings], default: **m**.

Defines the unit of length for x , y and z (mm, cm, m, km, in, ft).

y-Rotation (*y_rotation*) [real number], default: **0**.

Rotation of the surface around the y -axis of the reflector coordinate system, positive from the z -axis towards the x -axis.

z-Rotation (*z_rotation*) [real number], default: **0**.

Rotation of the surface around the z -axis of the reflector coordinate system, positive from the x -axis towards the y -axis.

Translation (*translation*) [struct].

The surface may be translated in the reflector coordinate system as specified by this vector.

x (x) [real number with unit of length], default: **0**.

x -coordinate of the translation vector.

y (y) [real number with unit of length], default: **0**.

x -coordinate of the translation vector.

z (z) [real number with unit of length], default: **0**.

x -coordinate of the translation vector.

Largest Root (*largest_root*) [item from a list of character strings], default: **on**.

The solution of the second order equation may result in two values (roots) for z :

on

The larger root is selected.

off

The smaller root is selected.

Remarks

The second order polynomial is defined by

$$A_{xx}x^2 + A_{xy}xy + A_{yy}y^2 + A_x x + A_y y + A_c = A_{zz}z^2 + A_z z + A_{xz}xz + A_{yz}yz$$

A surface must be given by a function of the form $z = f(x, y)$. For some values of the coefficients the equation above will have two solutions for z . Either the smallest or the largest solution must be selected by the *largest_root* attribute. If the equation has only one solution the value of the *largest_root* attribute is immaterial.

The surface is defined in the coordinate system of its corresponding reflector object. It is possible to displace and re-orient the surface relative to the reflector coordinate system by the rotations and translation attributes. This will be carried out in the following order: First, the surface is rotated around the *y*-axis of the reflector coordinate system according to the *y_rotation* attribute. This defines a new, rotated coordinate system. Subsequently, the surface is rotated around the *z*-axis of the new coordinate system according to the *z_rotation* attribute. Finally, the surface is translated by the vector defined in the *translation* attribute. The translation is carried out in the original reflector coordinate system.

TABULATED SURFACE

Purpose

This menu lists a series of ways in which a surface may be given in tabular forms. The methods have different characteristics such as

requirements to the position of the tabulated points and
the method of interpolation applied between these points.

Alternatively, the surface may be specified as a sum of a class of polynomials. The surface is then specified by

a set of coefficients to the specific polynomials.

Regular xy-Grid - the tabulated surface values shall be given in a regular grid in the xy -plane, the surface is given by cubic interpolation between the tabulated points.

Spline Surface - the surface is given by coefficients to a set of bi-cubic spline functions. Each spline function contributes to a local region of the surface.

Zernike Surface - the surface is given by coefficients to a set of Zernike polynomials. Each Zernike polynomial contributes to the entire surface.

Rotationally Symmetric - a rotationally symmetric surface specified by tabulated data along a radial profile.

Irregular xy-Grid, Triangulation - irregularly positioned surface points, polynomial interpolation in triangles with continuous surface values and first and second order derivatives.

Irregular xy-Grid, Pseudo Splines - irregularly positioned surface points, each surface point contributes to the full surface according to the distance to the surface point.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Tabulated Surface](#)

REGULAR XY-GRID (regular_xy_grid)

Purpose

This class defines a surface through a set of tabulated z -values in a regular xy -grid across the surface area. Cubic interpolation is used to determine the surface between the tabulated points.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Tabulated Surface](#)→[Regular xy-Grid](#)

Remarks

Syntax

```
<object name> regular_xy_grid
(
    file_name           : <f>,
    xy_unit            : <si>,
    z_unit              : <si>,
    xy_factor          : <r>,
    z_factor            : <r>,
    list                : <si>,
    translation         : struct(x:<rl>, y:<rl>, z:<rl>)
)
where
<r> = real number
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

File Name (*file_name*) [file name].

Name of the file containing the tabulated surface values in a regular xy -grid. The structure of the file is described in [Surface Data in Rectangular Grid](#).

xy-Unit (*xy_unit*) [item from a list of character strings], default: **m**.

Defines the unit of length for the x -and y -values (mm, cm, m, km, in, ft).

z-Unit (*z_unit*) [item from a list of character strings], default: **m**.

Defines the unit of length for the z -values (mm, cm, m, km, in, ft).

xy Factor (*xy_factor*) [real number], default: **1**.

Scale factor to be multiplied on the x and y grid values. The grid limits (see the remarks below) x_{\min} , y_{\min} , x_{\max} , y_{\max} will also be multiplied by this factor.

z-Factor (*z_factor*) [real number], default: **1**.

Scale factor to be multiplied on the *z*-values of the surface.

List (*list*) [item from a list of character strings], default: **off**.

A list of the read surface grid data may be produced in the standard output file.

Translation (*translation*) [struct].

The surface may be translated in the reflector coordinate system as specified by this vector.

x (x) [real number with unit of length], default: **0**.

x-coordinate of the translation vector.

y (y) [real number with unit of length], default: **0**.

y-coordinate of the translation vector.

z (z) [real number with unit of length], default: **0**.

z-coordinate of the translation vector.

Remarks

The surface is defined by tabulated *z*-values specified in a regular *xy*-grid in the reflector coordinate system. Cubic interpolation is used to determine the surface between the tabulated points.

In the first 3 lines the file contains : 1) an initial header string, 2) the grid limits x_{\min} , y_{\min} , x_{\max} , y_{\max} , and 3) the number of grid points along *x* and *y*: n_x and n_y . In the following lines the *z*-values of the surface are given, numbered by the two indices *i* and *j*, so that z_{ij} denotes the *z*-value of the surface at the *x* and *y* values

$$x_i = x_{\min} + \Delta x (i - 1), \quad i = 1, 2, \dots, n_x$$

$$y_j = y_{\min} + \Delta y (j - 1), \quad j = 1, 2, \dots, n_y$$

where the spacings in *x* and *y* are given by

$$\Delta x = \frac{x_{\max} - x_{\min}}{n_x - 1}$$

$$\Delta y = \frac{y_{\max} - y_{\min}}{n_y - 1}$$

Notice that the *z*-values must be given in the following sequence :

$$z_{11}, z_{12}, z_{13}, \dots, z_{1n_y}, z_{21}, z_{22}, z_{23}, \dots, z_{2n_y}, \dots, z_{n_x 1}, z_{n_x 2}, z_{n_x 3}, \dots, z_{n_x n_y}$$

with at least one *z*-value on each data line.

A detailed description of the file format and its contents are given in [Surface Data in Rectangular Grid](#).

The above (x, y, z) values describes the surface tabulated in the file. In the reflector coordinate system (x_r, y_r, z_r) the surface is given by

$$x_r = x + x_t$$

$$y_r = y + y_t$$

$$z_r = z + z_t$$

where (x_t, y_t, z_t) is given by the attribute *translation*.

SPLINE SURFACE (spline_surface)

Purpose

The class **Spline Surface** defines a surface through the coefficients to a set of bi-cubic B-spline functions. The coefficients are read from a file. Evaluation of the spline expansion determines the z -values of the surface as a function of x and y , within the limits for x and y specified in the file.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Tabulated Surface](#)→[Spline Surface](#)

Remarks

Syntax

```
<object name> spline_surface
(
    file_name           : <f>,
    xy_unit            : <si>,
    coef_unit          : <si>,
    list               : <si>,
    translation        : struct(x:<rl>, y:<rl>, z:<rl>)
)
where
<rl> = real number with unit of length
<f>  = file name
<si> = item from a list of character strings
```

Attributes

File Name (*file_name*) [file name].

Name of a file containing the B-spline coefficients. The contents and format of the file are described in [Surface Defined by Cubic Splines](#).

xy-Unit (*xy_unit*) [item from a list of character strings], default: **m**.

Defines the unit of length for the xy -limits(mm, cm, m, km, in, ft).

Coefficient Unit (*coef_unit*) [item from a list of character strings], default: **m**.

Defines the unit of length for the B-spline coefficients (mm, cm, m, km, in, ft).

List (*list*) [item from a list of character strings], default: **off**.

A list of the read B-spline coefficients may be produced in the standard output file.

Translation (*translation*) [struct].

The surface may be translated in the reflector coordinate system as specified by this vector.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of the translation vector.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of the translation vector.

z (*z*) [real number with unit of length], default: **0**.

z-coordinate of the translation vector.

Remarks

The spline functions are defined in a regular *xy*-grid across the surface and each spline function is non-zero only within a 4 by 4 rectangle in this grid (smaller rectangles are applied near the edges of the region of definition). Therefore, each coefficient controls only a local area of the surface.

The surface in the file is given as a function $z = z(x, y)$. This surface is in the reflector coordinate system (x_r, y_r, z_r) given by

$$x_r = x + x_t$$

$$y_r = y + y_t$$

$$z_r = z + z_t$$

where (x_t, y_t, z_t) are given by the attribute *translation*.

ZERNIKE SURFACE (zernike_surface)

Purpose

The class **Zernike Surface** defines a surface through a weighted sum of Zernike polynomials. The coefficients to the polynomials are read from a file. Evaluation of the Zernike expansion determines the z -values of the surface as a function of x and y . Each polynomial is characterized through two indices, m and n . The surface is defined within a circular or elliptical region.

In line with common usage, 'Zernike polynomials' and 'Zernike modes' are used indiscriminately in the documentation of GRASP.

Note: A surface description based on Zernike polynomials of high orders may result in unnecessarily large surface variations along the rim of the surface. Instead it is recommended to describe the surface as a **Spline Surface**.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Tabulated Surface](#)→[Zernike Surface](#)

Remarks

Syntax

```
<object name> zernike_surface
(
    file_name : <f>,
    headline : <i>,
    mode_number_format : struct(line:<i>, column:<i>),
    mode_values_line : <i>,
    mode_values_column : struct(m:<i>, n:<i>, x:<i>, y:<i>),
    mode_definition : <si>,
    mode_unit : <si>,
    centre : struct(x:<rl>, y:<rl>),
    half_axis : struct(x:<rl>, y:<rl>),
    rotation : <r>,
    weight_factor : <r>,
    max_m_mode_index : <i>,
    max_n_mode_index : <i>,
    list : <si>
)
where
<i> = integer
<r> = real number
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

File Name (*file_name*) [file name].

Name of the file containing tabulated coefficients to Zernike modes. The contents and format of the file are described in [Surface Data as Zernike Modes](#), and may be controlled by the following specifiers.

Headline (*headline*) [integer], default: **1**.

Number of line in file which holds a string identifier.

Mode Number Format (*mode_number_format*) [struct].

Describes where to read the number of modes in the file.

Line (*line*) [integer], default: **2**.

The line in the file where the number of Zernike modes shall be found.

Column (*column*) [integer], default: **1**.

The total number of Zernike modes is given as the *column*'th number on the above defined line.

Mode Value Line (*mode_values_line*) [integer], default: **3**.

The number of the line in the file holding the first Zernike mode coefficients.

Mode Values Column (*mode_values_column*) [struct].

The Zernike mode coefficients are organised in the file with one coefficient per line, each consisting of 4 columns with the following content.

M (*m*) [integer], default: **1**.

Column *m* contains the *m*-index of the mode.

N (*n*) [integer], default: **2**.

Column *n* contains the *n*-index of the mode.

x (*x*) [integer], default: **3**.

Column *x* contains the first coefficient value (see Mode Definition below).

y (*y*) [integer], default: **4**.

Column *y* contains the second coefficient value (see Mode Definition below).

Mode Definition (*mode_definition*) [item from a list of character strings], default: **amp_deg**.

Definition of coefficient values (see also remarks below).

amp_deg

The coefficients are given by their amplitude (value 1) and a reference direction (value 2) (in degrees), corresponding to the TICRA-format.

even_odd

The coefficients are given as amplitudes for even (value 1) and odd (value 2) modes, in accordance with the format used by previous versions of the TICRA software package POS.

Mode Unit (*mode_unit*) [item from a list of character strings], default: **m**.

Specifies the unit of length for the amplitudes of the coefficients (mm, cm, m, km, in, ft).

Centre (*centre*) [struct].

Specifies the centre of the area in which the Zernike modes are defined.

x (x) [real number with unit of length], default: **0**.

x-coordinate of the centre.

y (y) [real number with unit of length], default: **0**.

y-coordinate of the centre.

Half Axis (*half_axis*) [struct].

Specifies the lengths of the half axes of the ellipse over which the Zernike modes are defined (see also remarks below).

x (x) [real number with unit of length].

Length of half axis in the *x*-direction.

y (y) [real number with unit of length].

Length of half axis in the *y*-direction.

Rotation (*rotation*) [real number], default: **0**.

The Zernike modes may be rotated through an angle, positive from the *x*-axis towards the *y*-axis. The rotation is in the *x'y'*-plane (in degrees), see the remarks below.

Weight Factor (*weight_factor*) [real number], default: **1**.

The amplitudes of the coefficients in the file will be multiplied by this factor. If the coefficients originate from POS4 (or later versions) the weight factor shall be equal to 1. If the coefficients originate from POS3 (or POD2 or earlier versions) they have been normalised by the mean value of the half axes, *a* and *b*, of the region over which they are defined. Thus the coefficients are multiplied by a weight factor of $(a + b)/2$, where *a* and *b* shall be given in the same unit of length as employed in POS/POD.

Max m-Mode Index (*max_m_mode_index*) [integer], default: **-1**.

The Zernike mode expansion is truncated in the index *m* at Max m-Mode Index, i.e. modes with a larger *m* are excluded. All *m*-modes are included if Max m-Mode Index is negative.

Max n-Mode Index (*max_n_mode_index*) [integer], default: **-1**.

The Zernike mode expansion is truncated in the index *n* at Max n-Mode Index, i.e. modes with a larger *n* are excluded. All *n*-modes are included if Max n-Mode Index is negative.

List (*list*) [item from a list of character strings], default: **off**.

A list of the read Zernike coefficients may be produced in the standard output file.

Remarks

As it is seen from the following equations each Zernike polynomial contributes to the entire surface shape.

The surface is expressed as a function $z = z(x, y)$ in a coordinate system co-parallel to the reflector coordinate system and with origin as described in the attribute `Centre` relative to the reflector coordinate system. If the surface shall be displaced in the z -direction this may be done by adding a contribution of the basic Zernike polynomials z_0^0 which equals unity. This displacement has to be added as a mode coefficient in the file given by `File Name`.

The surface is generated by a weighted sum of Zernike polynomials, defined by

$$z_n^m(x', y') = R_n^m(\rho) \cos(m(\phi - \phi_n^m)) \quad (1)$$

where ρ and ϕ are usual polar coordinates in the $x'y'$ -plane, i.e. $x' = \rho \cos \phi$ and $y' = \rho \sin \phi$. The radial polynomials $R_n^m(\rho)$ are defined in the Technical Description.

The Zernike polynomials in Eq. (1) are only useful for surface approximations within the unit circle $0 \leq \rho \leq 1$ in the $x'y'$ -plane. Therefore a transformation from the unit circle to the elliptical reflector area in the xy -plane is necessary.

The surface may then be expressed by

$$z(x, y) = w \sum_{m,n} \{a_n^m R_n^m(\rho) \cos(m(\phi - \phi_n^m))\} \quad (2)$$

where w is the `weight_factor`, a_n^m is the amplitude of the Zernike mode (m,n) , $R_n^m(\rho)$ is the radial Zernike polynomial, and ϕ_n^m is the reference direction. Note that ρ , as well as ϕ and ϕ_n^m , are given in the $x'y'$ -coordinate system.

The attribute `rotation` is specified as an angle in the xy -plane.

If `mode_definition` is specified to `amp_deg`, then a_n^m is the first coefficient value (value 1) and ϕ_n^m is the second coefficient value (value 2).

If `mode_definition` is specified to `even_odd`, then Eq. (2) is re-written as

$$z(x, y) = w \sum_{m,n} \{a_n^m \cos(m\phi_n^m) R_n^m(\rho) \cos(m\phi) + a_n^m \sin(m\phi_n^m) R_n^m(\rho) \sin(m\phi)\} \quad (3)$$

where $a_n^m \cos(m\phi_n^m)$ is the first coefficient value (value 1) and $a_n^m \sin(m\phi_n^m)$ is the second coefficient value (value 2).

The default values for the format of the data file correspond to the format described in [Surface Data as Zernike Modes](#).

ROTATIONALLY SYMMETRIC (rotationally_symmetric)

Purpose

The class *Rotationally Symmetric* defines a rotationally symmetric surface by means of tabulated data for a profile of the surface. Data points are given as z -values along a radial surface cut. The radial surface cut for an already defined reflector surface may be generated by an object of class *Surface Data Output*.

For a rotationally symmetric *reflector*, the scatterer can be specified from *Circular Symmetric Reflector* and calculated using *BoR-MoM*. The rotational symmetry is then exploited such that it is possible to handle larger reflectors than in the standard *MoM*.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Tabulated Surface](#)→[Rotationally Symmetric](#)

Remarks

Syntax

```
<object name> rotationally_symmetric
(
    nodes          : table(
        <r> <r>
        ...),
    file_name     : <f>,
    rho_unit      : <si>,
    z_unit        : <si>,
    rho_factor    : <r>,
    z_factor      : <r>,
    n_points      : <i>,
    tip           : <si>,
    list          : <si>,
    origin        : struct(x:<rl>, y:<rl>, z:<rl>)
)
where
<i> = integer
<r> = real number
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

Nodes (*nodes*) [table (*,2)].

The profile defining the rotational symmetric surface is generated by interpolation between nodes given in a table. The nodes shall be specified as consecutive sets of (ρ, z) -values. There is no upper limit

for the number of specified nodes. The attribute File Name will be ignored when nodes are specified.

In the GUI the nodes are specified in a table with two columns. When many nodes shall be specified then a table may be inserted in the tor-file, see the remarks for details.

rho (*rho*) [real number].

The radius ρ of an interpolation node

z (*z*) [real number].

The *z*-coordinate of the same interpolation node.

File Name (*file_name*) [file name].

Name of the file containing the tabulated surface values for a rotationally symmetric surface. The contents and the format of the file are described in *Rotationally Symmetric Surface*. The file will be neglected if interpolation points have been specified in the attribute Nodes.

rho-Unit (*rho_unit*) [item from a list of character strings], default: **m**.

Defines the unit of length for the radial values(mm, cm, m, km, in, ft).

z-Unit (*z_unit*) [item from a list of character strings], default: **m**.

Defines the unit of length for the *z*-values(mm, cm, m, km, in, ft).

rho-Factor (*rho_factor*) [real number], default: **1**.

Scale factor to be multiplied on the radial values.

z-Factor (*z_factor*) [real number], default: **1**.

Scale factor to be multiplied on the *z*-values of the surface.

N-Points (*n_points*) [integer], default: **0**.

>0: The data points defining the surface are interpolated into a set of N-Points equispaced points

=0: Irregular data are interpolated into a set of equispaced points with the same number of data points as in the original set

Tip (*tip*) [item from a list of character strings], default: **defined_in_file**.

Determines the local geometry on the axis of rotation, $\rho = 0$:

defined_in_file

When the surface profile is read from file the tip is controlled by a parameter KTIP, see the file description in *Rotationally Symmetric Surface*. The instructions 'on' and 'off' take precedence over the value of KTIP in the data file.

This setting is not valid when the surface profile is specified in the attribute Nodes.

off

The surface has a tangent parallel to the *xy*-plane at $\rho = 0$.

on

The surface has a tip at $\rho = 0$.

List (*list*) [item from a list of character strings], default: **off**.

A list of the read surface data may be produced in the standard output file.

Origin (*origin*) [struct].

Defines the coordinates (in the reflector coordinate system) of the origin of the coordinate system in which the rotation is given.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of the origin.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of the origin.

z (*z*) [real number with unit of length], default: **0..**

z-coordinate of the origin.

Remarks

The surface is expressed as a function of x and y , $z = z(x, y)$, in the reflector coordinate system. Since the surface is rotationally symmetric it may be described by a single radial cut, $z = z(\rho)$ where

$$\rho = \sqrt{x^2 + y^2}$$

The file contains n_p pairs of (ρ, z) -values - the nodes of the profile - that define $z(\rho)$:

$$z_i(\rho_i), i = 1, 2, \dots, n, \rho_i \geq 0. \quad (1)$$

At intermediate points, the function is determined by cubic (3rd order) interpolation in a set of equispaced points. In the end intervals (and outside these)

$$0 \leq \rho \leq \rho_j \text{ and } \rho > \rho_{n-1}$$

where ρ_j is the smallest specified non-zero radial value, the interpolation is parabolic (2nd order). This interpolation will usually result in a tip at $\rho = 0$. If the attribute **Tip** is set to 'off' the cubic interpolation will be continued across $\rho = 0$ applying an additional definition point at $\rho = -\rho_j$ with

$$z(-\rho_j) = z(\rho_j) \quad (2)$$

The specification may contain data not equispaced in ρ , in which case the program will pre-interpolate the data into a new, equispaced set. The number of points in the new set is determined by the attribute **N-Points**.

The nodes may be specified in a file defined by the attribute **File Name**. For this case, a complete description of the file format and its contents may be found in the section *Rotationally Symmetric Surface*.

Alternatively, the nodes may be specified in the GUI or in the *tor*-file by a table. In the GUI the values shall be given one by one which is unpractically when the number of nodes is large. In such a case it is easier to specify a few nodes in the GUI, save the project and open the *tor*-file in a text editor. Here the actual object of class *Rotationally Symmetric* shall be searched for. With three nodes specified, the object specification may look like

```
My_surface rotationally_symmetric
(
    nodes          : table
    (
        1.00000E+00  0.00000E+00
        2.00000E+00  2.40000E-01
        3.00000E+00  3.90000E-01
    )
    ,
    file_name      : " ",
    tip            : on
)
```

The three GUI-specified nodes appear on the three lines between the lines with start- and end-parentheses for the attribute *nodes*. These lines may be replaced with actual values pasted from a table in which the user has the data, one node given by (ρ, z) per line.

IRREGULAR XY-GRID, TRIANGULATION (irregular_xy_grid_triangulation)

Purpose

This class defines a surface through interpolation in triangles with corner positions given by tabulated irregularly spaced (x, y, z)-points read from a data file.

Links

Classes→*Geometrical Objects*→*Surface*→*Tabulated Surface*→*Irregular xy-Grid, Triangulation*

Remarks

Syntax

```
<object name> irregular_xy_grid_triangulation
(
    file_name : <f>,
    headline : <i>,
    file_xyz_number : struct(in_file:<si>, line:<i>, column:<i>),
    file_xyz_values : struct(start_line:<i>, x_column:<i>,
                                y_column:<i>, z_column:<i>),
    xy_unit : <si>,
    z_unit : <si>,
    number_of_points : <i>,
    change_reference : struct(status:<si>,
                                definition_coor_sys:ref(<n>),
                                new_reference_coor_sys:ref(<n>)),
    xy_factor : <r>,
    z_factor : <r>,
    min_distance : <rl>,
    concave_border : struct(status:<si>,
                                minimum_height_over_baseline:<r>),
    interpolation : <si>,
    list : <si>,
    plot_points : struct(symbol:<si>, size:<rl>),
    plot_triangles : <si>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

File Name (*file_name*) [file name].

Name of file containing tabulated surface values specified at irregularly spaced points. The contents and format of the file are described in [Surface Data in Irregular Points](#), but may be relaxed according to the following format specifiers.

Headline (*headline*) [integer], default: **1**.

Number of line in the file which holds a string identifier (will be printed in the standard output file).

File xyz Number (*file_xyz_number*) [struct].

Specification on how to determine the number of data points in the file.

In File (*in_file*) [item from a list of character strings], default: **yes**.

Determines if the number of points is read in the file.

yes

The number of points is read in the file as specified by *line* and *column*.

no

The number of points is not read, but the file is scanned to determine the number of points. The values of *line* and *column* are immaterial.

Line (*line*) [integer], default: **2**.

The line in the file where the number of data points shall be found.

Not used for *in_file*: no.

Column (*column*) [integer], default: **1**.

The column (or variable/word number) on the above line in which the number of data points shall be found. Not used for *in_file*: no.

File xyz Values (*file_xyz_values*) [struct].

Specification of where the data points will be located in the file.

Start Line (*start_line*) [integer], default: **3**.

Number of the first line in the data file with *x*-, *y*- and *z*-values.

x Column (*x_column*) [integer], default: **1**.

The column on the line where the *x*-value will be found.

y Column (*y_column*) [integer], default: **2**.

The column on the line where the *y*-value will be found.

z Column (*z_column*) [integer], default: **3**.

The column on the line where the *z*-value will be found.

xy-Unit (*xy_unit*) [item from a list of character strings], default: **m**.

Specifies the unit of length for the *x*-and *y*-values in the file(mm, cm, m, km, in, ft).

z-Unit (*z_unit*) [item from a list of character strings], default: **m**.

Specifies the unit of length for the z -values in the file(mm, cm, m, km, in, ft).

Number of Points (*number_of_points*) [integer], default: **0**.

Number of surface points to be read for the surface representation. If zero, all points in the data file are included. The number of points must be larger than 7.

Change Reference (*change_reference*) [struct].

The surface data points should be given in the coordinate system of the associated reflector. If that is not the case, it is necessary to transform the data to the reflector coordinate system. This is indicated by the *change_reference* attribute.

Status (*status*) [item from a list of character strings], default: **off**.

Control of the coordinate-system operation.

off

No change. The points in the file are already given in the reflector coordinate system.

on

The irregular points shall be transformed to a new coordinate system.

Definition Coor Sys (*definition_coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate System* specifying the coordinate system in which the surface points in the file are given. Not used when *status* is specified to 'off'.

New Reference Coor Sys (*new_reference_coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate System* specifying the reflector coordinate system. Not used when *status* is specified to 'off'.

xy Factor (*xy_factor*) [real number], default: **1**.

Scale factor to be multiplied on the x - and y -values. The scaling is performed in the reflector coordinate system.

z Factor (*z_factor*) [real number], default: **1**.

Scale factor to be multiplied on the z -values of the surface. The scaling is performed in the reflector coordinate system.

Min Distance (*min_distance*) [real number with unit of length], default: **0**.

Minimum distance between points in the xy -plane. Points that are spaced closer to each other will be automatically removed.

Concave Border (*concave_border*) [struct].

Generation of a concave border (edge).

Status (*status*) [item from a list of character strings], default: **on**.

Determines if too long border triangles should be eliminated.

off

No triangle elimination maintaining a convex border.

on

Elimination of too long border triangles

Minimum Height Over Baseline (*minimum_height_over_baseline*) [real number], default: **0.044**.

limit for h/c of border triangles where c is triangle border length and h is the triangle height. If status is specified to 'on' then triangles along the border are neglected when h/c is less than specified here.

Interpolation (*interpolation*) [item from a list of character strings], default: **smooth**.

When a point on the surface shall be used in a computation, its z -value is found by interpolation in the generated triangles. The interpolation may be carried out in either of three ways. See the Remarks section for details.

smooth

A local fifth order interpolation with continuous surface derivatives.

linear

The surface is locally plane over each triangel.

smooth_global

A global fifth order interpolation with continuous surface derivatives.

List (*list*) [item from a list of character strings], default: **off**.

A list of the (x, y, z) -coordinates to the irregular surface points may be produced in the standard output file.

off

No list is generated.

on

A list is generated of the coordinates of the surface points as given in the input file and as applied, i.e. in the reflector coordinate system after a possible scaling.

Plot Points (*plot_points*) [struct].

The sample points may be plotted as part of the reflector in [Reflector Plot](#).

Symbol (*symbol*) [item from a list of character strings], default: **off**.

Control of the symbol to be plotted.

off

No plot of sample points

cross

The sample points are plotted as crosses.

Size (*size*) [real number with unit of length], default: **0**.

Size of the cross lines.

Plot Triangles (*plot_triangles*) [item from a list of character strings], default: **off**.

The generated interpolation triangles may be plotted as part of the reflector in [Reflector Plot](#).

Remarks

The program makes a triangulation over the reflector surface with the specified data points as corners and it is strongly recommended to inspect a plot of the triangles by specifying Plot Triangles to 'on'. The interpolation will be best in regions where the points are dense and evenly spaced in all directions.

When the attribute Interpolation is set to 'smooth' or 'smooth_global' then in each triangle a fifth degree polynomial is fitted to the data subject to the requirement that the surface value and the first and second order derivatives become continuous from one triangle to the neighbour triangles. When 'smooth' is chosen then only the neighbour triangles are used for determining the derivatives; when 'smooth_global' is chosen then all triangles are used with a weight which decreases with the distance to the actual triangle. The method is able to interpolate very accurately inside the border points.

The default values for the format of the data file correspond to the general data file to GRASP, as specified in [Surface Data in Irregular Points](#).

If the irregular points are defined in an inconvenient coordinate system the points may be transformed to another coordinate system using the Change Reference attribute. A typical case is when the points are given in a mechanical measurement coordinate system and shall be redefined to the reflector coordinate system. Here the attribute New Reference Coor Sys must be the associated reflector coordinate system.

Limitations

Low accuracy of the sampling data may generate fluctuating 2'nd order derivatives. In that case an alternative solution may be a surface generated with an object of class [Irregular xy-Grid, Pseudo Splines](#). For ray optics (GO and GTD calculations) it is recommended to apply the surface model in [Irregular xy-Grid, Pseudo Splines](#).

It is possible to reduce unrealistic surface fluctuations in the modelled surface by calculating interpolated surface values in a regular grid and then print these values on a file using class *Surface Data Output*. The grid spacing shall be chosen with care in order to exclude the undesired inaccuracies and still maintain the characteristics of the surface. The generated file with grid data may then be applied as surface definition in *Regular xy-Grid*.

IRREGULAR XY-GRID, PSEUDO SPLINES (irregular_xy_grid_pseudo_splines)

Purpose

This class defines a surface through interpolation between tabulated z -values, given at irregularly spaced (x, y) -points read from a data file.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Tabulated Surface](#)→[Irregular xy-Grid, Pseudo Splines](#)

Remarks

Syntax

```
<object name> irregular_xy_grid_pseudo_splines
(
    file_name : <f>,
    headline : <i>,
    xyz_number_format : struct(line:<i>, column:<i>),
    xyz_values_line : <i>,
    xyz_values_column : struct(x:<i>, y:<i>, z:<i>),
    xy_unit : <si>,
    z_unit : <si>,
    number_of_points : <i>,
    change_reference : struct(status:<si>,
                               definition_coor_sys:ref(<n>),
                               new_reference_coor_sys:ref(<n>)),
    xy_factor : <r>,
    z_factor : <r>,
    min_distance : <rl>,
    list : <si>,
    plot_points : struct(symbol:<si>, size:<rl>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

File Name (*file_name*) [file name].

Name of file containing tabulated surface values specified at irregularly spaced points. The contents and format of the file are described by the following format specifiers. The default values of these correspond to the format described in [Surface Data in Irregular Points](#).

Headline (*headline*) [integer], default: **1**.

Number of line in file which holds a string identifier (will be printed in the standard output file).

xyz Number Format (*xyz_number_format*) [struct].

Describes where to read the number of tabulated surface points in the data file.

Line (*line*) [integer], default: **2.**

The line in the file where the number of points shall be found.

Column (*column*) [integer], default: **1.**

The number of points is given as the *column*'th number/word on the above defined *line*.

xyz Values Line (*xyz_values_line*) [integer], default: **3.**

The number of the line in the file holding coordinates for the first surface point.

xyz Values Column (*xyz_values_column*) [struct].

The tabulated surface points are organised in the file with one (x, y, z) -coordinate set per line, each consisting of 3 columns with the content (x, y, z) .

x (*x*) [integer], default: **1.**

Column *x* contains the *x*-coordinate for the surface point.

y (*y*) [integer], default: **2.**

Column *y* contains the *y*-coordinate for the surface point.

z (*z*) [integer], default: **3.**

Column *z* contains the *z*-coordinate for the surface point.

xy-Unit (*xy_unit*) [item from a list of character strings], default: **m.**

Specifies the unit of length for the *x*- and *y*-values in the file (mm, cm, m, km, in, ft).

z-Unit (*z_unit*) [item from a list of character strings], default: **m.**

Specifies the unit of length for the *z*-values in the file (mm, cm, m, km, in, ft).

Number of Points (*number_of_points*) [integer], default: **0.**

Number of surface points to be used in the surface representation. If zero, all points in the data file are included. The number of points must be larger than 5.

Change Reference (*change_reference*) [struct].

The surface data points should be given in the coordinate system of the associated reflector. If that is not the case, it is necessary to transform the data to the reflector coordinate system. This is indicated by the *change_reference* attribute.

Status (*status*) [item from a list of character strings], default: **off.**

Control of the coordinate-system operation.

off

No change. The points in the file are already given in the reflector coordinate system.

on

The irregular points shall be transformed to a new coordinate system.

Definition Coor Sys (*definition_coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate System* specifying the coordinate system in which the surface points in the file are given. Not used when *status* is specified to 'off'.

New Reference Coor Sys (*new_reference_coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate System* specifying the reflector coordinate system. Not used when *status* is specified to 'off'.

xy Factor (*xy_factor*) [real number], default: **1**.

Scale factor to be multiplied on the *x*- and *y*-values. The scaling is performed in the reflector coordinate system.

z Factor (*z_factor*) [real number], default: **1**.

Scale factor to be multiplied on the *z*-values of the surface. The scaling is performed in the reflector coordinate system.

Min Distance (*min_distance*) [real number with unit of length], default: **0**.

Minimum distance between points in the *xy*-plane. Points that are spaced closer to each other will be automatically removed.

List (*list*) [item from a list of character strings], default: **off**.

A list of the (*x*, *y*, *z*)-coordinates to the irregular surface points may be produced in the standard output file.

off

No list is generated.

on

A list is generated of the coordinates of the surface points as given in the input file and as applied, i.e. in the reflector coordinate system after a possible scaling.

Plot Points (*plot_points*) [struct].

The sample points may be plotted as part of the reflector in *Reflector Plot*.

Symbol (*symbol*) [item from a list of character strings], default: **off**.

Control of the symbol to be plotted.

off

No plot of sample points.

cross

The sample points are plotted as crosses.

Size (size) [real number with unit of length], default: **0**.

Size of the cross lines.

Remarks

A quintic pseudo spline (QPS) algorithm for interpolation of irregularly tabulated points and smooth extrapolation is employed to generate the surface. The algorithm involves solving a system of $N + 6$ linear equations, where N is the number of irregularly spaced surface points. The requirement of continuous second order derivatives results in the QPS expression :

$$z(u, v) = \sum_{i=1}^N a_i [r_i(u, v)]^5 + b_1 u^2 + b_2 u v + b_3 v^2 + b_4 u + b_5 v + b_6 \quad (1)$$

where the set (u, v) corresponds to (x, y) scaled to the region $-1 \leq u \leq 1$, $-1 \leq v \leq 1$, and where r_i is the distance to data point No. i :

$$r_i(u, v) = \sqrt{(u - u_i)^2 + (v - v_i)^2} \quad (2)$$

The default values for the format of the data file correspond to the general data file to GRASP, as specified in [Surface Data in Irregular Points](#).

If the irregular points are defined in an inconvenient coordinate system the points may be transformed to another coordinate system using the *change_reference* attribute. A typical case is when the points are given in a mechanical measurement coordinate system and shall be redefined to the reflector coordinate system. Here the attribute *new_reference_coor_sys* must be the associated reflector coordinate system.

Limitations

The density of the sampling points must not vary 'too much' over the surface. If the sample spacing is very irregular, then ripples, generated in an area with dense spacing, may spread to adjacent areas with a wide spacing.

Therefore it is strongly recommended to make a plot of the reflector surface prior to a field analysis. This plot will show the resulting interpolated surface and an inappropriate interpolation may be identified.

The present surface modelling generates a surface with continuous second order derivatives. A more smooth surface with also continuous third order derivatives may be generated by [Irregular xy-Grid, Triangulation](#).

IMPERFECTION MODELLING

Purpose

Some surface imperfections may be modelled and added to an ideal *Reflector* surface.

Pseudo random surface variations may be modelled by

Random Surface

And the quilting effect which is due to the core structure of light weight reflectors may be modelled by

Quilting Surface

Links

Classes→*Geometrical Objects*→*Surface*→*Imperfection Modelling*

RANDOM SURFACE (random_surface)

Purpose

This class defines a surface in a regular xy -grid, by means of random values for the z -coordinate within a user-specified interval.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Imperfection Modelling](#)→[Random Surface](#)

[Remarks](#)

Syntax

```
<object name> random_surface
(
    x_range           : struct(start:<rl>, end:<rl>, np:<i>),
    y_range           : struct(start:<rl>, end:<rl>, np:<i>),
    peak              : <rl>,
    seed              : <i>,
    list              : <si>
)
where
<i> = integer
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

x-Range (*x_range*) [struct].

Grid specification in x range.

Start (*start*) [real number with unit of length].

Start value of the x range.

End (*end*) [real number with unit of length].

End value of the x range.

Np (*np*) [integer].

Number of values in the x range.

y-Range (*y_range*) [struct].

Grid specification in y range.

Start (*start*) [real number with unit of length].

Start value of the y range.

End (*end*) [real number with unit of length].

End value of the y range.

Np (*np*) [integer].

Number of values in the y range.

Peak (*peak*) [real number with unit of length].

Specifies the peak of the surface *z*-values at any grid point, so that
-Peak $\leq z \leq$ +Peak.

Seed (*seed*) [integer], default: **1**.

Seed for the random number generator. Each Seed generates a new surface, but the same Seed will always result in the same surface.

List (*list*) [item from a list of character strings], default: **off**.

A list of the generated random grid data may be produced in the standard output file.

Remarks

The surface is defined by random *z*-values at the grid points specified by x-Range and y-Range. At intermediate points, the surface is determined by a cubic interpolation procedure. In order to perform the interpolation near the edges of the grid, the algorithm will add additional points, one spacing outside the specified grid. These grid points are also assigned a random *z*-value, and will be included in the output file list, when *list* is 'on'.

The surface is a part of a *Reflector* definition. Care must be taken that the specified grid (given by x-Range and y-Range) includes the full *Rim* of the reflector.

The rms-value of the generated surface is, due to the cubic interpolation, given by

$$\text{rms} = .47 \cdot \text{Peak} \quad (1)$$

It shall be noted that the grid points defining the surface will have an rms-value which is about 20% higher than the rms-value for the full surface.

QUILTING SURFACE (quilting_surface)

Purpose

This class models a surface deviation of bulges (or dimples) originating in a back structure (the reflector core) which forms a regular polygonal grid, a so-called quilted structure. The surface is undisturbed along the edges of the polygons but is raised (or depressed) inside the polygons for the bulges (the dimples, respectively). The effect is also known as core print-through.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Imperfection Modelling](#)→[Quilting Surface](#)

Remarks

Syntax

```
<object name> quilting_surface
(
    side_length      : <rl>,
    peak             : <rl>,
    type             : <si>,
    grid_orientation : <r>
)
where
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Side Length (*side_length*) [real number with unit of length].

Length of one side of the polygons in the grid structure.

Peak (*peak*) [real number with unit of length].

Peak deviation of the surface at the centres of the polygons. The deviation is positive along the *z*-axis of the reflector coordinate system, i.e. a positive value results in bulges while a negative value results in dimples (the surface is pressed up or down, respectively).

Type (*type*) [item from a list of character strings], default: **hexagonal**.

The quilting of the back structure forms regular polygons over the entire *xy*-plane of the reflector coordinate system. The polygons may be one of the following types (see also the remarks below).

hexagonal

Grid with hexagonal elements.

square

Grid with square elements.

triangular

Grid with equilateral triangle elements.

Grid Orientation (*grid_orientation*) [real number], default: **0**.

Rotation angle of the grid in the *xy*-plane of the reflector coordinate system. The angle shall be given in degrees, measured positive from the *x*-axis towards the *y*-axis (see the remarks below).

Remarks

The geometry of the surface deviation is based on the geometry of the polygons of which the core of the reflector is build. The polygons are regular in the *xy*-plane of the reflector coordinate system and projected on the reflector surface. The surface deviation is determined along the *z*-axis of the same coordinate system.

The modelled deviations are build by a combination of sine and cosine functions and are one-sided according to the sign of the attribute *peak*. This definition results in a surface deviation which is continuous and has continuous derivatives.

The surface is a part of a *Reflector* definition and represents a distortion which is added to the ideal reflector surface.

The quilting surface may be based on three different polygonal grids as specified by the attribute *type*: the hexagonal, the square or the triangular grid. These will be illustrated by examples in the following. In these examples also the definition of the attributes *side_length* and *grid_orientation* are illustrated.

Hexagonal structure

Equal hexagonal polygons fit over the *xy*-plane of the reflector coordinate system. One polygon is centred at the origin of the *xy*-plane and the *grid_orientation* is measured from the *x*-axis to a line of symmetry through a corner of the central polygon, cf. the following figure. The value of the attribute *side_length* is given by the length of a side of the hexagonal polygons.

Square structure

The square structure is formed as a regular grid in the *xy*-plane of the reflector coordinate system. One of the squares has centre at the origin of the *xy*-plane, cf. the following figure. The *grid_orientation* is a rotation angle measured from the *x*-axis to a line parallel to the lines of the square grid and through the centre of the central square. The value of *side_length* is given by the length of a side of the squares.

Triangular structure

The triangular structure is formed as an infinite grid of triangles in the *xy*-plane of the reflector coordinate system. At each grid point six triangles

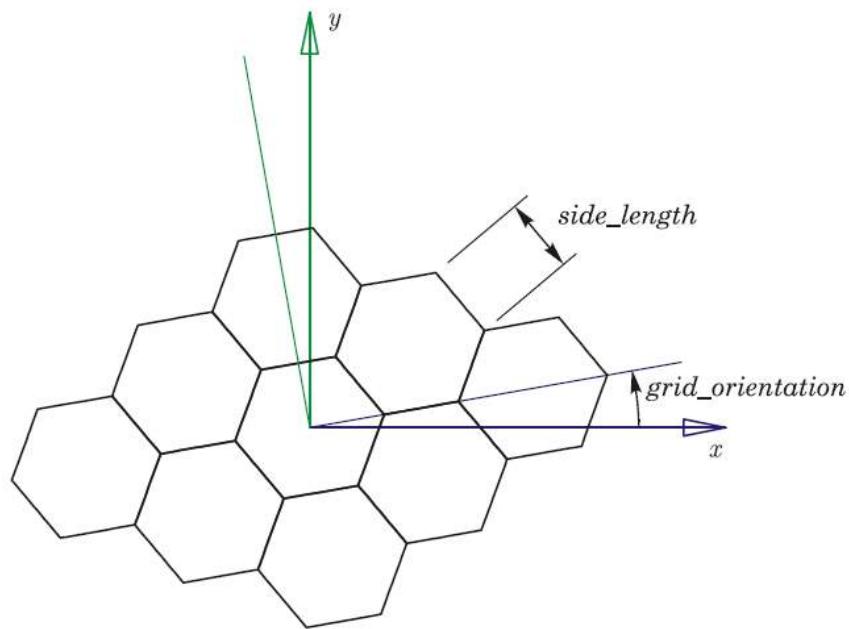


Figure 1 Hexagonal back structure. Some of the hexagonal polygons forming the hexagonal back structure are shown in the *xy*-plane of the reflector coordinate system.

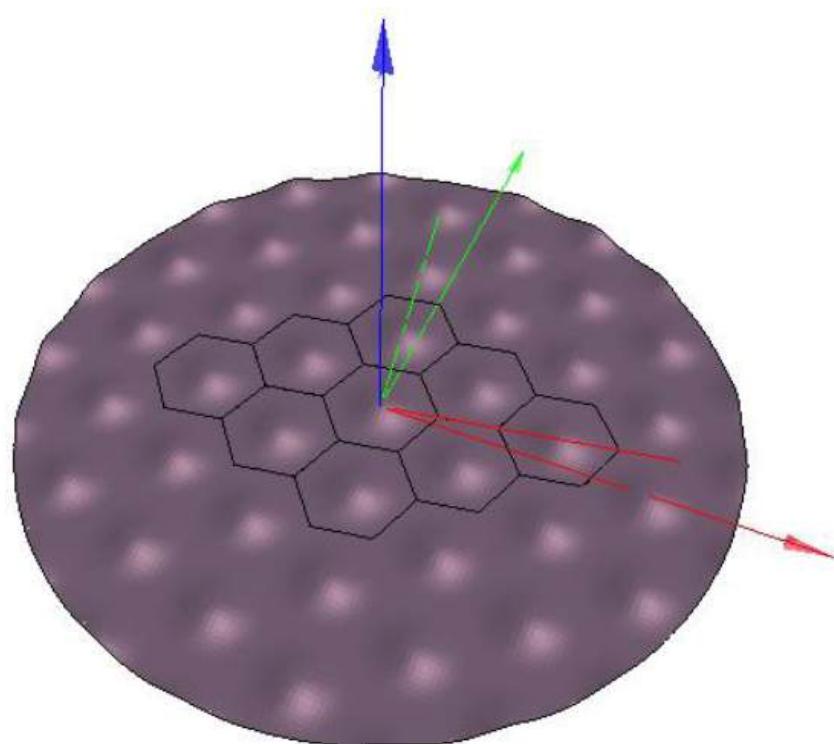


Figure 2 3D view (generated by the command *Get OpenGL Plot (Obsolete)*) of a plane reflector with bulges based on the hexagonal back structure. A part of the hexagonal grid is also shown.

meet and such a point constitutes the origin of the *xy*-plane, cf. the following figure. The *grid_orientation* is a rotation angle measured from the *x*-axis to

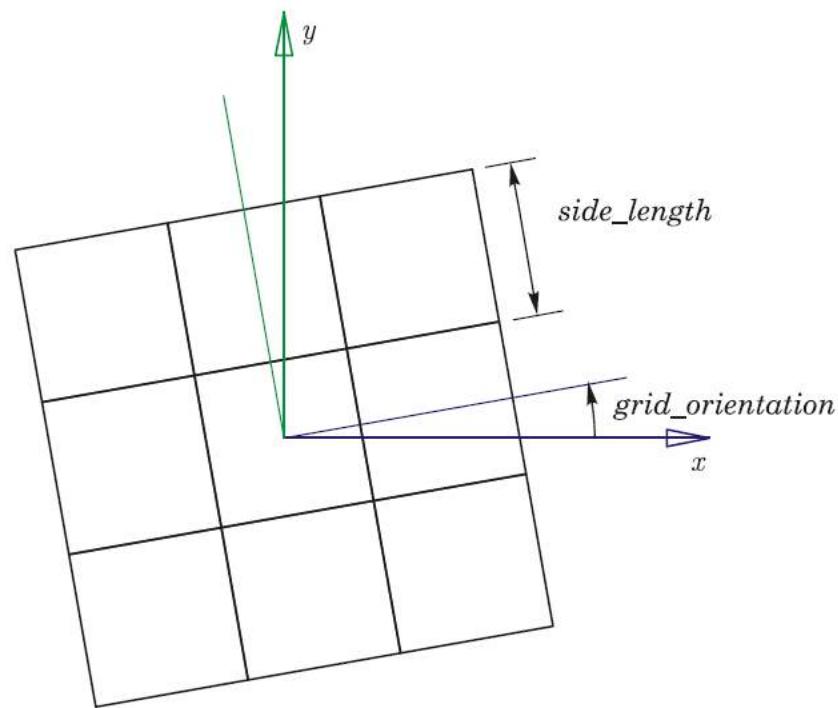


Figure 3 Square back structure shown in the xy -plane of the reflector coordinate system. Nine squares are shown but the structure extends over the complete reflector surface.

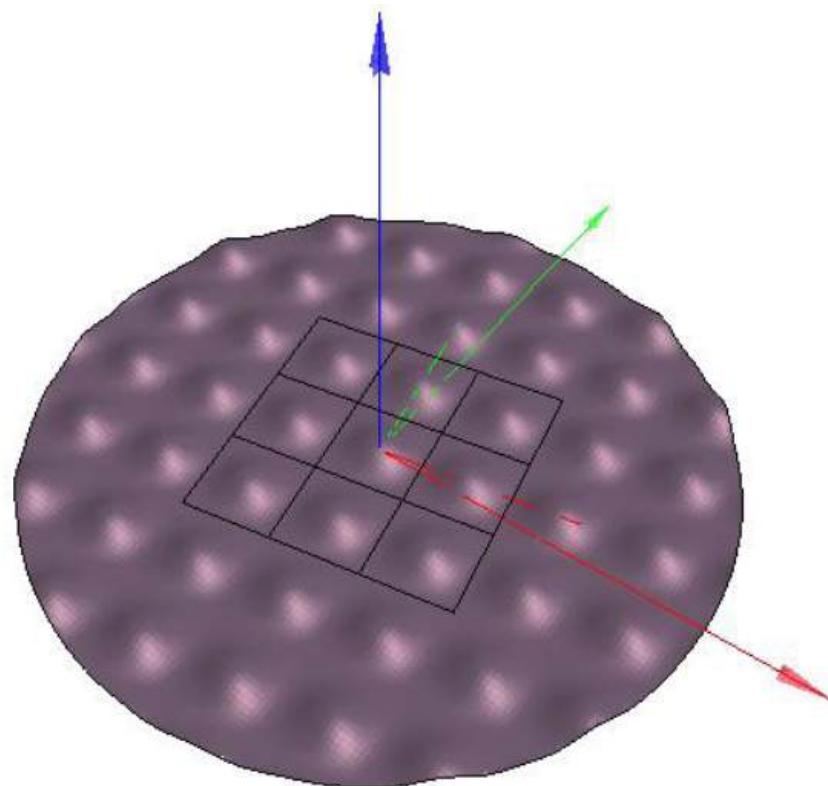


Figure 4 3D view of a plane reflector with bulges based on the square back structure. A part of the square grid is also shown.

a grid line. The value of *side_length* is given by the length of the sides of the triangles.

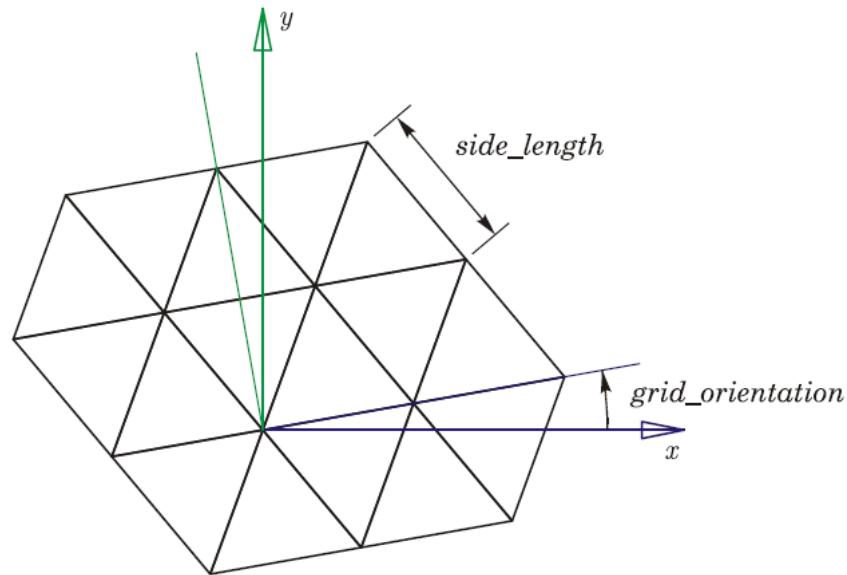


Figure 5

Triangular structure. Some triangles forming the triangular back structure are shown in the xy -plane of the reflector coordinate system.

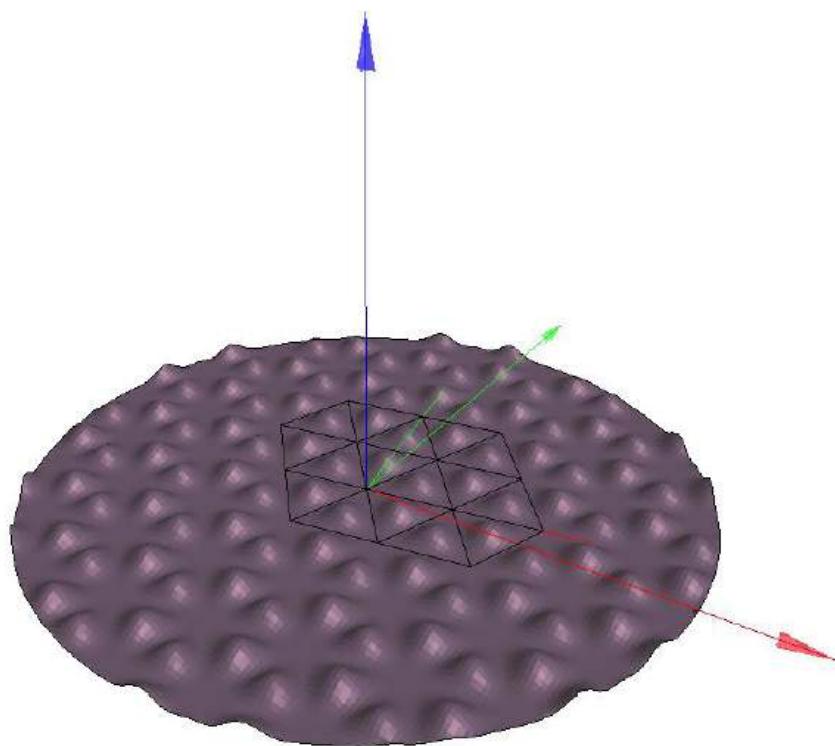


Figure 6

3D view of a plane reflector with bulges based on the triangular back structure. A part of the triangular grid is shown.

UNFURLABLE SURFACE

Purpose

An unfurlable surface may be modelled as a mesh:

Unfurlable Surface, Rib Attached

Unfurlable Surface, Button Attached

Links

Classes→*Geometrical Objects*→*Surface*→*Unfurlable Surface*

UNFURLABLE SURFACE, RIB ATTACHED (unfurlable_surface_rib_attached)

Purpose

This class defines the surface of a tensioned membrane between a number of parabolic ribs. A typical example is an umbrella-type unfurlable reflector.

Links

Classes→*Geometrical Objects*→*Surface*→*Unfurlable Surface*→*Unfurlable Surface, Rib Attached*

Remarks

Syntax

```
<object name> unfurlable_surface_rib_attached
(
    rib_centre                      : struct(x:<rl>, y:<rl>),
    focal_length                     : <rl>,
    ribs                            : <i>,
    rib_orientation                 : <r>,
    pillow                           : <si>
)
where
<i> = integer
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Rib Centre (*rib_centre*) [struct].

Coordinates of the centre of the ribs (the hub) in reflector coordinates.

x (x) [real number with unit of length], default: **0**.

x-coordinate of the centre.

y (y) [real number with unit of length], default: **0**.

y-coordinate of the centre.

Focal Length (*focal_length*) [real number with unit of length].

Focal length of the parent paraboloid of which the ribs are a part.

Ribs (*ribs*) [integer].

Number of ribs; must be greater than 2.

Rib Orientation (*rib_orientation*) [real number], default: **0**.

The ribs are positioned with a constant angular spacing in the *xy*-plane of the reflector coordinate system. The attribute *rib_orientation* indicates the angle in this plane from the *x*-axis to the projection of one of the ribs. The angle shall be given in degrees, measured positive from the *x*-axis towards the *y*-axis.

Pillow (*pillow*) [item from a list of character strings], default: **on**.

Controls the shape of the surface between the ribs.

on

The pillow effect is included.

off

The pillow effect is not included.

Remarks

The surface between two ribs, denoted a gore, can be modelled either with or without the so-called pillow effect.

When the pillow effect is not included, the surface is modelled as follows. Imagine a line, l , through the centre of the ribs and parallel to the axis of their parent paraboloid (see Figure 1 and Figure 2). Points on neighbouring ribs with the same (radial) distance to this line are connected by straight lines, and these straight lines define the reflector surface as shown in the two figures.

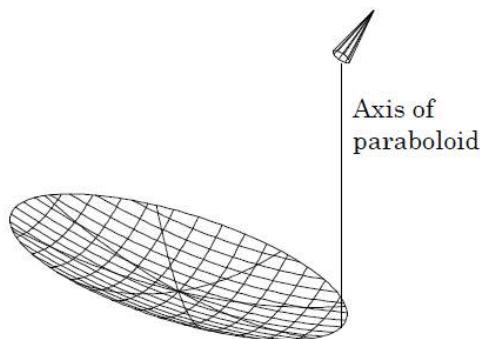


Figure 1 Rib Parent paraboloid of the ribs.

When the pillow effect is included in the surface modelling, the surface between the ribs bulges towards the reflector concave side, see Figure 3. Calling this a 'pillow effect' is misleading, since the shape of a pillow is determined by a surface under pressure from one side. However, the term is retained here since its usage is widespread.

As modelled here the pillow effect describes a stretched mesh without bending stiffness. The tension in the mesh in the area between two ribs contracts the mesh in radial directions. As the ribs are curved, the mesh bulges between the ribs. On the other hand, this results in increased tensions in the mesh in the azimuthal direction. Equilibrium is reached when the curvature of the mesh is the same (but of opposite directions) in radial and azimuthal directions.

Notice that this surface can only be used in a physical optics calculation and not with GTD, since the second derivatives of the surface are undefined along the ribs.

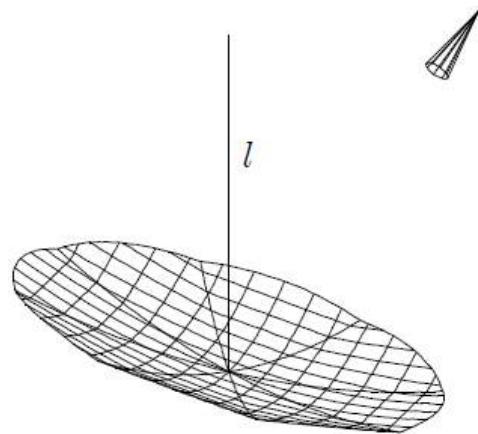


Figure 2

Rib attached mesh without pillow effect (8 ribs). Each segment is here a parabolic cylinder. This is easily seen for the two segments in the plane of symmetry: the lines perpendicular to the plane are straight lines.

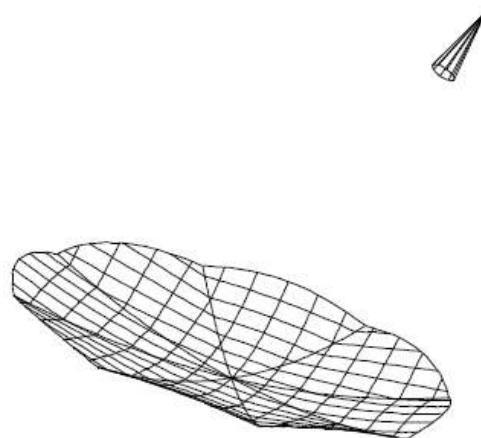


Figure 3

Rib attached mesh with pillow effect.

UNFURLABLE SURFACE, BUTTON ATTACHED (unfurlable_surface_button_attached)

Purpose

This class defines the surface of a tensioned membrane attached to a number of buttons, positioned in an equilateral triangular grid. A typical example is a so-called '3D-scissors' unfurlable reflector.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Unfurlable Surface](#)→[Unfurlable Surface, Button Attached](#)

[Remarks](#)

Syntax

```
<object name> unfurlable_surface_button_attached
(
    button_distance           : <rl>,
    focal_length              : <rl>,
    file_name                 : <f>
)
where
<rl> = real number with unit of length
<f>  = file name
```

Attributes

Button Distance (*button_distance*) [real number with unit of length].

Distance between the buttons, which are distributed in a grid of equilateral triangles (in the *xy*-plane of the reflector coordinate system).

Focal Length (*focal_length*) [real number with unit of length].

Focal length of the parent paraboloid. The buttons are assumed to lie on this paraboloid. The apex is at the origin of the reflector coordinate system.

File Name (*file_name*) [file name].

Name of file containing a user-defined pre-calculated surface of one triangular facet with the buttons at its three corners. The file format shall be in accordance with the format for surface data, cf. [Surface Data in Rectangular Grid](#). See also the remarks below.

Remarks

The surface of one triangular facet has been pre-calculated with a computer program for solving two-dimensional Laplacian problems with general boundary conditions.

A sample surface generated in this way for a button radius equal to 1/20 of the button distance is included in the program and will be applied when File Name is not specified.

Note that this surface definition can only be used in a physical optics calculation and not with GTD, since the second derivatives of the surface are undefined at the buttons.

PYRAMIDAL SURFACE (pyramidal_surface)

Purpose

This class defines a regular *Pyramidal Surface*. The axis of the pyramidal surface is parallel to the z -axis of the coordinate system of the corresponding reflector.

Links

[Classes](#)→[Geometrical Objects](#)→[Surface](#)→[Pyramidal Surface](#)

Remarks

Syntax

```
<object name> pyramidal_surface
(
    apex                      : struct(x:<rl>, y:<rl>, z:<rl>),
    number_of_faces            : <i>,
    rotation                  : <r>,
    angle_to_edges             : <r>,
    rounding_angle             : <r>
)
where
<i> = integer
<r> = real number
<rl> = real number with unit of length
```

Attributes

Apex (*apex*) [struct].

Coordinates of the pyramid apex in the reflector coordinate system.
The axis of the pyramid is directed along positive z .

x (*x*) [real number with unit of length], default: **0**.
x-coordinate of the apex.

y (*y*) [real number with unit of length], default: **0**.
y-coordinate of the apex.

z (*z*) [real number with unit of length], default: **0**.
z-coordinate of the apex.

Number of Faces (*number_of_faces*) [integer].

Number of lateral faces of the pyramid; must be at least 3.

Rotation (*rotation*) [real number], default: **0**.

The angle as a plane through the pyramid axis and parallel to the zx -plane shall be rotated to contain one of the pyramid edges. The angle is positive from the x -axis towards the y -axis.

Angle to Edges (*angle_to_edges*) [real number].

Angle in degrees from the axis of the pyramid (parallel to the positive z -direction) to a pyramid edge.

Rounding Angle (*rounding_angle*) [real number], default: **0**.

The edges of the pyramid may be rounded by this angle. The rounding is performed within a region given by rotating a plane - containing the pyramid axis and a pyramid edge - the angle *rounding_angle* to both sides around the pyramid axis (see also the remarks below).

Remarks

The pyramidal surface has an axis parallel to the z -axis of the coordinate system of the corresponding reflector. The position of the apex is given by the attribute *apex*.

The pyramidal surface is that of a pyramid having n_f lateral faces, where

$$n_f = \text{number_of_faces}$$

The surface is regular in the sense that it will cover itself when it is rotated the angle $\frac{360^\circ}{n_f}$ around the axis of the pyramid.

The opening angle of the pyramid is given as the angle θ_e from the positive z -axis to one of the edges of the pyramid,

$$\theta_e = \text{angle_to_edges}$$

The opening angle may be obtuse ($\theta_e \geq 90^\circ$) resulting in the outer side of the pyramid is seen from the positive z -axis, or it may be acute ($\theta_e < 90^\circ$) in which case the inner side of the pyramid is seen from the positive z -axis, as illustrated in the following figure.

For describing a reflector, the surface is limited by a *Rim*. The reflector appears as a pyramid with a flat base only if this rim is a properly oriented regular polygon with n_f edges (may be defined as an *Tabulated Rim, xy-Input*).

For PO calculations the currents are laid out in a grid which depends on the rim of the reflector and is independent of the edges of the pyramidal surface. PTD currents along the edges are not included (but may be included along the rim of the reflector).

For GO and GTD calculations the edges may cause problems in the ray tracing due to the fold along the edge in the otherwise smooth surface (the surface derivative is discontinuous). It may therefore be desirable to model a more smooth pyramidal surface which may be done by a rounding of the edges. This is done by specifying a *rounding_angle* greater than zero.

The *rounding_angle* is given in the aperture plane (the *xy*-plane of the reflector coordinate system) to either side of each projected edge, see the following figure.

The rounding is given as follows. Within each rounding section the pyramidal surface is replaced by a part of a circular cone defined with the cone tip at the pyramid apex and the cone touching the plane surfaces of the pyramid along the lines defined by the *rounding_angle*. The apex of the surface will

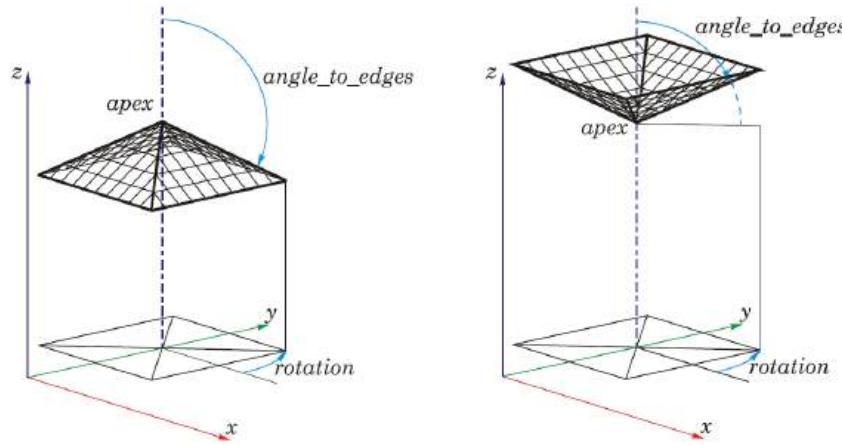


Figure 1

Illustration of the attributes to the *Pyramidal Surface* with *number_of_faces* being 4. The surface is here applied to a *Reflector* with a *Rectangular Rim*. The surface is rotated the angle *rotation* (here 45°) such that the faces are parallel to the *x*- and *y*-axis. In the left figure *angle_to_edges* $> 90^\circ$ and in the right figure *angle_to_edges* $< 90^\circ$.

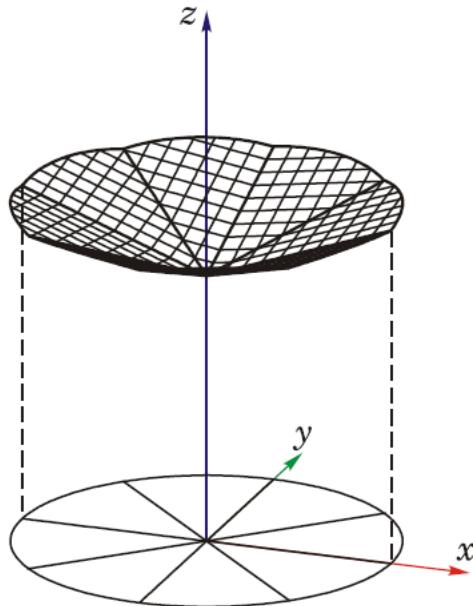


Figure 2

A reflector with pyramidal surface with *number_of_faces* equal to 8 and with circular rim; *apex* is placed on the *z*-axis and the angle *rotation* is 0. Thus, one of the edges falls in the *zx*-plane as shown by the projection of the edges in the *xy*-plane.

still be sharp (discontinuous surface derivatives). The rounding is illustrated in the last figure.

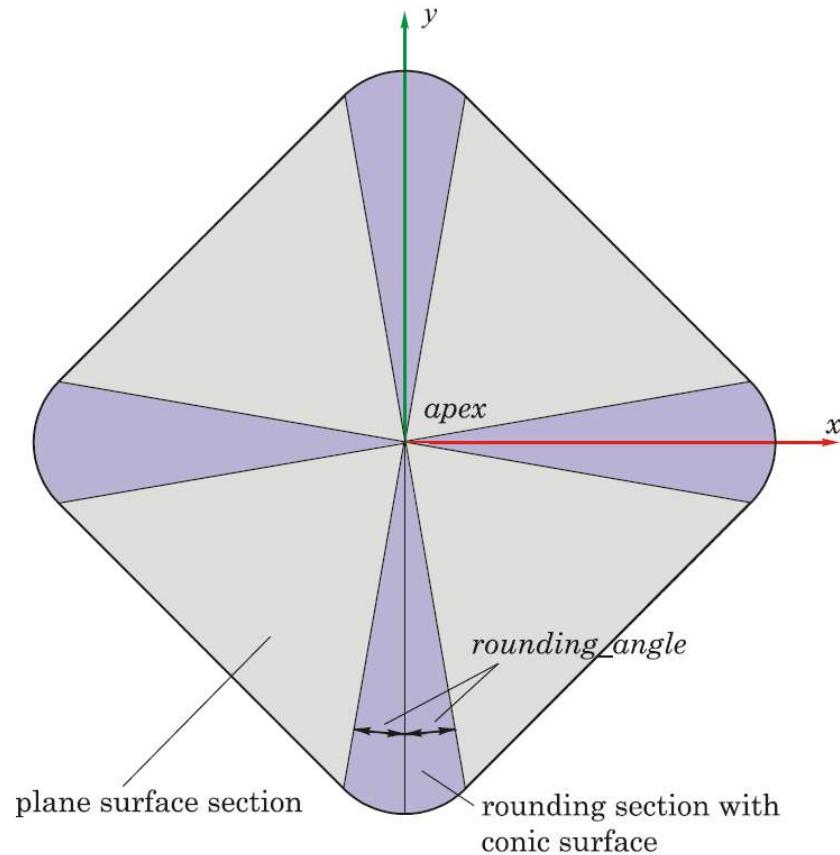


Figure 3

Square pyramid with rounded edges. The pyramid are seen from the positive z -axis and the *rounding_angle* is given in the xy -plane. Within the dark sections the pyramidal surface is rounded and within the light sections the surface is unmodified.

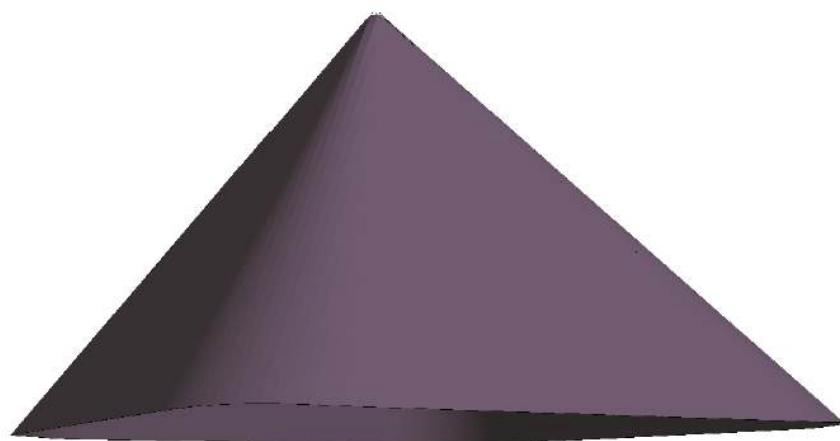


Figure 4

Illustration of the square pyramid with rounded edges.

RIM**Purpose**

This class describes the bounding *Rim* of a *Scatterer*, which in conjunction with a *Surface*, constitutes a *Reflector*. The following types of rims are available.

Elliptical Rim

Rectangular Rim

Triangular Rim

Tabulated Rim

Superelliptical Rim

Links

Classes→*Geometrical Objects*→*Rim*

ELLIPTICAL RIM (elliptical_rim)

Purpose

The class *Elliptical Rim* defines a *Rim* with the shape of an ellipse, typically specified in a *Reflector* object. The rim is specified in a plane perpendicular to the *z*-axis of the coordinate system of its corresponding reflector.

Links

Classes→*Geometrical Objects*→*Rim*→*Elliptical Rim*

Remarks

Syntax

```
<object name> elliptical_rim
(
    centre : struct(x:<rl>, y:<rl>),
    half_axis : struct(x:<rl>, y:<rl>),
    rotation : <r>
)
where
<r> = real number
<rl> = real number with unit of length
```

Attributes

Centre (*centre*) [struct].

Coordinates of the centre of the elliptical rim in the coordinate system of the reflector.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of centre.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of centre.

Half Axis (*half_axis*) [struct].

Half axes of the elliptical rim. When the rim is not rotated, the axes are parallel to the coordinate axes of the reflector coordinate system.

x (*x*) [real number with unit of length].

Half axis in the *x*-direction.

y (*y*) [real number with unit of length].

Half axis in the *y*-direction.

Rotation (*rotation*) [real number], default: **0**.

The rim can be rotated through an angle (in degrees) specified by Rotation, positive from the *x*-axis towards the *y*-axis. The rotation is around the centre of the rim.

Remarks

The non-rotated rim is defined in the reflector coordinate system by the ellipse

$$\left(\frac{x - x_c}{a}\right)^2 + \left(\frac{y - y_c}{b}\right)^2 = 1$$

where (x_c, y_c) is the centre of the elliptical rim, and a and b are the half axes, parallel to the x - and y -axis, respectively. This is illustrated in Figure 1. The rim may be rotated the angle θ around its centre.

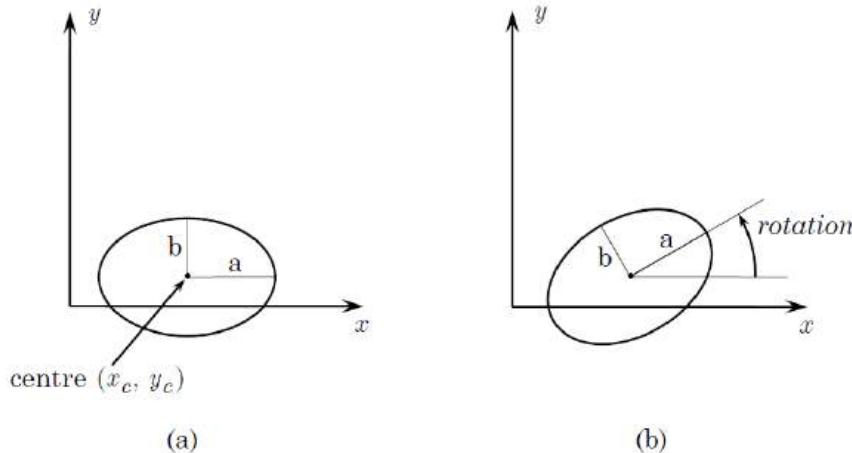


Figure 1

The elliptical rim, (a) unrotated and (b) rotated in positive direction.

RECTANGULAR RIM (rectangular_rim)

Purpose

The class *Rectangular Rim* defines a *Rim* with the shape of a rectangle, typically specified in a *Reflector* object. The rim is specified in a plane perpendicular to the z -axis of the coordinate system of its corresponding reflector.

Links

Classes→*Geometrical Objects*→*Rim*→*Rectangular Rim*

Remarks

Syntax

```
<object name> rectangular_rim
(
    centre                      : struct(x:<rl>, y:<rl>),
    side_lengths                 : struct(x:<rl>, y:<rl>),
    rotation                     : <r>
)
where
<r> = real number
<rl> = real number with unit of length
```

Attributes

Centre (*centre*) [struct].

Coordinates of the centre of the rectangular rim in the coordinate system of the reflector.

x (x) [real number with unit of length], default: **0**.
x-coordinate of centre.

y (y) [real number with unit of length], default: **0**.
y-coordinate of centre.

Side Lengths (*side_lengths*) [struct].

Lengths of the sides of the rectangular rim. When the rim is not rotated, the sides are parallel to the coordinate axes of the reflector coordinate system.

x (x) [real number with unit of length].
Length of the side parallel to the *x*-axis.
y (y) [real number with unit of length].
Length of the side parallel to the *y*-axis.

Rotation (*rotation*) [real number], default: **0**.

The rim can be rotated through an angle (in degrees) specified by Rotation, positive from the *x*-axis towards the *y*-axis. The rotation is around the centre of the rim.

Remarks

Geometry

The geometry of the rectangular rim is defined by the position of its centre as specified by the attribute `Centre(x,y)` and by the lengths of the sides defined by `Side Lengths(x,y)` as illustrated in Figure 1. The rectangle may be rotated the angle `Rotation` around its centre.

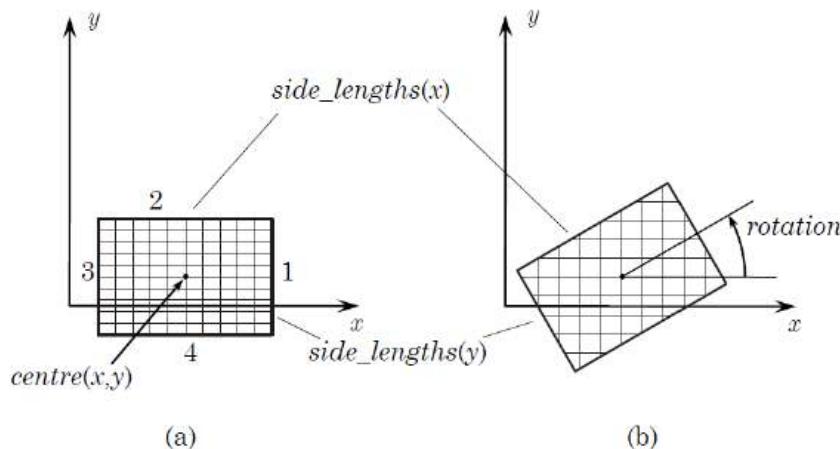


Figure 1 The rectangular rim, (a) unrotated and (b) rotated in positive direction. The edge numbering shown in (a) follows the rim when it is rotated.

Edge numbering

Internally, the sides of the *Rectangular Rim* are numbered from 1 through 4 according to the figure (a) above, which shows the rectangular rim prior to any rotation. Side 1 has an *x*-value larger than side 3, and side 2 has a *y*-value larger than side 4.

This numbering scheme is important when PTD is employed in the field analysis, since it is possible to specify different sampling densities along the individual sides in a rectangular reflector (see the *PO, Single-Face Scatterer* class).

The side numbering is of similar importance when the *Single-Reflector GTD* class is used, since it is possible to trace GTD rays diffracted by selected reflector edges.

TRIANGULAR RIM (triangular_rim)

Purpose

The class *Triangular Rim* defines a *Rim* with the shape of a triangle, typically specified in a *Reflector* object. The rim is specified in a plane perpendicular to the z -axis of the coordinate system of its corresponding reflector.

Links

Classes→*Geometrical Objects*→*Rim*→*Triangular Rim*

Remarks

Syntax

```
<object name> triangular_rim
(
    corner_1 : struct(x:<rl>, y:<rl>),
    corner_2 : struct(x:<rl>, y:<rl>),
    corner_3 : struct(x:<rl>, y:<rl>)
)
where
<rl> = real number with unit of length
```

Attributes

Corner 1 (*corner_1*) [struct].

Coordinates of the first corner of the rim in the xy -plane of the reflector coordinate system.

x (x) [real number with unit of length].

x-coordinate of the first corner.

y (y) [real number with unit of length].

y-coordinate of the first corner.

Corner 2 (*corner_2*) [struct].

Coordinates of the second corner of the rim in the xy -plane of the reflector coordinate system.

x (x) [real number with unit of length].

x-coordinate of the second corner.

y (y) [real number with unit of length].

y-coordinate of the second corner.

Corner 3 (*corner_3*) [struct].

Coordinates of the third corner of the rim in the xy -plane of the reflector coordinate system.

x (x) [real number with unit of length].

x-coordinate of the third corner.

y (y) [real number with unit of length].
 y -coordinate of the third corner.

Remarks

The corner points may be specified in arbitrary order.

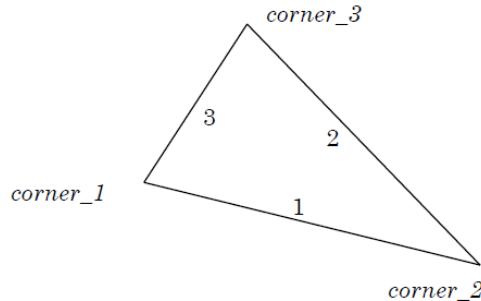


Figure 1 The triangular rim with the numbering of the sides.

The sides of the triangle are numbered as shown in the figure. This numbering scheme is important when PTD is employed in the field analysis, since it is possible to specify different sampling densities along the individual sides (see the class *PO, Single-Face Scatterer*).

The side numbering is of similar importance when the *Single-Reflector GTD* class is used, since it is possible to trace GTD rays diffracted by corresponding reflector edges.

The centre of the triangular rim is defined as the average value of the coordinates of the three corners.

TABULATED RIM

Purpose

The following types of tabulated rims are available:

Tabulated Rim, xy-Input

Tabulated Rim, rho-phi-Input

Tabulated Rim (Obsolete)

Links

Classes→*Geometrical Objects*→*Rim*→*Tabulated Rim*

TABULATED RIM, XY-INPUT (tabulated_rim_xy)

Purpose

The class *Tabulated Rim, xy-Input* defines, by means of a table, a *Rim*, typically specified in a *Reflector* object. The table values define points along the rim, in a plane perpendicular to the z -axis of the coordinate system of the reflector, and the values are tabulated in rectangular (x, y) -coordinates.

Links

[Classes](#)→[Geometrical Objects](#)→[Rim](#)→[Tabulated Rim](#)→[Tabulated Rim, xy-Input](#)

Remarks

Syntax

```

<object name> tabulated_rim_xy
(
    file_name : <f>,
    file_xy_number : struct(in_file:<si>, line:<i>, column:<i>),
    file_xy_values : struct(start_line:<i>, x_column:<i>,
                           y_column:<i>),
    file_corner_points_id : struct(status:<si>, column:<i>,
                                    id_string:<s>),
    unit : <si>,
    file_form : <si>,
    number_of_points : <i>,
    min_distance : <rl>,
    scaling_factor : <r>,
    translation : struct(x:<rl>, y:<rl>),
    polar_origin : struct(status:<si>, x:<rl>, y:<rl>),
    rotation : <r>,
    interpolation : <si>,
    list_rim : <si>,
    plot_points : struct(symbol:<si>, size:<rl>),
    external_command : ref(<n>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<f> = file name
<si> = item from a list of character strings

```

Attributes

File Name (*file_name*) [file name].

Name of the file containing the tabulated rim data.

File xy-Number (*file_xy_number*) [struct].

Specification on how to determine the number of data points in the file. The number of applied data points may be limited by the value of the attribute *number_of_points*.

In File (*in_file*) [item from a list of character strings], default: **yes**.

Specifies if the number of data points is to be read in the file.

yes

The number of points is read in the file as specified by *line* and *column*.

no

The number of points is not read, but the file is scanned to determine the number of points. The values of *line* and *column* are immaterial.

Line (*line*) [integer], default: **2**.

The line in the file where the number of data points shall be found.

Not used for *in_file*: no.

Column (*column*) [integer], default: **1**.

The number of given rim points is read as the *column*'th number/word on the above defined line. Not used for *in_file*: no.

File xy-Values (*file_xy_values*) [struct].

Specification of where the data points will be located in the file.

Start Line (*start_line*) [integer], default: **3**.

Number of the first line in the data file with *x*- and *y*-values.

x-Column (*x_column*) [integer], default: **1**.

The column on the line where the *x*-value will be found.

y-Column (*y_column*) [integer], default: **2**.

The column on the line where the *y*-value will be found.

File Corner Points ID (*file_corner_points_id*) [struct].

Specifies how corner points are located in the file.

Status (*status*) [item from a list of character strings], default: **off**.

Existence of the corner points in the rim data file.

Column (*column*) [integer], default: **3**.

The column on the line where the corner point identification will be found. Not used for *status*: off.

ID-String (*id_string*) [character string], default: **corner_point**.

A string on the line identifying that the *xy*-point is a corner point.

Blanks are not allowed in the string.

Unit (*unit*) [item from a list of character strings], default: **m**.

Defines the unit of length for the *x*- and *y*-values (mm, cm, m, km, in, ft).

File Form (*file_form*) [item from a list of character strings], default: **free_format**.

The format of file may be given in two ways.

free_format

The rim data are given in free format on the file (see an example below).

old_grasp

The rim data are given in the old GRASP rim format described in [Reflector Rim Data](#). The following attributes File xy-Number, File xy-Values and File Corner Points ID shall not be specified and it is not possible to specify corner points.

Number of Points (*number_of_points*) [integer], default: **0**.

Specification of the number of data points to be applied. If zero, all valid data points in the file will be applied. The data are assumed written in the columns specified by the struct members *x_column* and *y_column* of the attribute File xy-Values. As long as the data in the file follows this pattern, the data are assumed valid data points.

Minimum Distance (*min_distance*) [real number with unit of length], default: **0**.

Minimum distance between points in the *xy*-plane. If two points are closer than this distance the one point will be neglected (see the remarks for details).

Scaling Factor (*scaling_factor*) [real number], default: **1**.

All *x*- and *y*-values in the file may be multiplied by this scale factor, before the Translation specified elsewhere.

Translation (*translation*) [struct].

The rim may be translated in the *xy*-plane. The translation is given in the reflector coordinate system and is carried out before the Polar Origin calculation and the Rotation (see these attributes below).

x (*x*) [real number with unit of length], default: **0**.

Translation along *x*.

y (*y*) [real number with unit of length], default: **0**.

Translation along *y*.

Polar Origin (*polar_origin*) [struct].

The rim points in the *xy*-plane are converted to polar form. The origin of this polar coordinate system is specified here. The polar origin also defines the centre of the rim when defining a polar integration grid for PO as well as the central hole in a [Reflector](#).

Status (*status*) [item from a list of character strings], default: **automatic**.

Selection of status.

automatic

The polar origin is calculated as the centre of gravity of the rim points.

user_defined

The polar origin is specified by the members x and y of this struct.

x (x) [real number with unit of length], default: **0**.

x-coordinate of the polar origin in the reflector coordinate system.

Not used for status: automatic.

y (y) [real number with unit of length], default: **0**.

y-coordinate of the polar origin in the reflector coordinate system.

Not used for status: automatic.

Rotation (rotation) [real number], default: **0**.

The rim may be rotated through an angle (in degrees), positive from the *x*-axis towards the *y*-axis. The rotation is around the Polar Origin.

Interpolation (interpolation) [item from a list of character strings], default: **spline**.

When a point on the rim shall be used in a computation, its position is found by interpolation among the tabulated points. The interpolation may be carried out in either of two ways.

spline

The interpolation is carried out by spline functions (in (ρ, ϕ)) resulting in a smooth rim.

linear

The interpolation is linear (in (x, y)) resulting in a polygonal rim.

List Rim (list_rim) [item from a list of character strings], default: **off**.

Specifies whether a list of the generated rim data shall be reproduced in the standard output file.

Plot Points (plot_points) [struct].

The sample points defining the rim may be plotted as crosses (corner points in red) and the polar origin as a circle when the reflector is plotted by *Reflector Plot*.

Symbol (symbol) [item from a list of character strings], default: **off**.

Selection of plot symbol.

off

No plot of sample points.

cross

The sample points are plotted as crosses.

Size (*size*) [real number with unit of length], default: **0**.

Size of the cross lines.

External Command (*external_command*) [name of an object], default: **blank**.

Reference to an object of class *External Command*. If the command specified by this reference is executed, e.g. if a variable is changed, the tabulated rim file will be read again. This can be used to create a plugin that updates a complex reflector geometry seamlessly whenever the input changes.

Remarks

The rim is specified by tabulated (x, y) -values in the reflector coordinate system. The values may be given in arbitrary order. The (x, y) -points are converted to polar form (ρ, ϕ) defined with respect to the user-specified or the program-calculated Polar Origin, which then has $\rho = 0$. It is essential that the polar origin be selected as an interior point, so that ρ becomes a single-valued function of ϕ . After the points have been read (and scaled, translated and rotated if requested) they are sorted in increasing ϕ . ϕ is positive from the x -axis towards the y -axis and the direction parallel to the x -axis defines $\phi = 0$.

Duplicate points are removed automatically. When Minimum Distance is specified positive and when two consecutive points with a distance less than Minimum Distance are found, that point with the largest ϕ will be removed.

A list of tabulated and regenerated values is given if List Rim is specified to 'on'.

File format

When *file_form* is specified to 'free_format' it is possible to read a general table of rim data. For example, the file with the rim data may have the following content:

```
polygonal rim
  x-values  y-values
LINE1  0. ,-.2  corner_point
LINE2  0.15 ,0.
LINE3  0.15 ,0.1  corner_point
LINE4  0.075 ,0.16
LINE5  0. ,0.1  corner_point
LINE6  -.075 ,0.16
LINE7  -.15 ,0.1  corner_point
LINE8  -.15 ,0.
```

The file has two header lines and eight lines of rim points. The number of lines with rim data is not given in the file. Four of the rim points are corners. In order to read this file the following attribute values shall be given:

File Form: *free_format*,

File xy-Number: struct(in_file:no, line:0, column:0),

```
File xy-Values: struct(start_line:3, x_column:2, y_column:3),
File Corner Points ID: struct(status:on, column:4, id_string:corner_point),
Number of Points: 0
```

Corners and edge numbering

Rim points between the tabulated points will in general be determined by interpolation. Thus, if Interpolation is specified to 'spline' the rim will be smooth without sharp corners. If Interpolation is specified to 'linear' the rim will be polygonal.

A point in the data input file may be defined as a corner point by adding a corner point identification string as specified in the attribute File Corner Points ID. The rim interpolation will not be applied across a corner point and the rim will in general not be smooth at a corner.

Note that when linear interpolation is used the rim is polygonal, as mentioned above. However, this is not the same as saying that each point is a corner point. That still needs to be indicated by the File Corner Points ID.

The part of the rim which is between two corner points is called an edge and for PTD and GTD purposes the edges are given consecutive numbers starting from 1 and increasing with increasing ϕ -value. Edge No. 1 is the one which contains the direction $\phi = 0$ from the Polar Origin. (cf. the following figures). The edge numbering is carried out after possible scaling, transformation and rotation of the rim data. If a corner point falls exactly at $\phi = 0$ then edge No. 1 is bounded by this corner and the corner with the next higher positive ϕ -value. If possible, such situation should be avoided since even small numerical inaccuracies may result in a ϕ -value being determined slightly positive, thus changing the edge numbering.

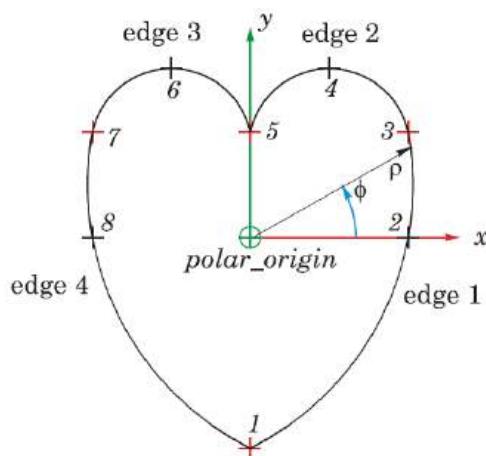


Figure 1

An example on a rim specified by 8 points (each marked by a cross) of which the odd numbered are corner points (marked by red). The Polar Origin is user defined at $(0, 0)$ and marked by a cross in a circle.

The numbering scheme for the edges is important when PTD is employed in the field analysis, since it is possible to specify different integration densities

along the individual edges of the reflector rim (see the class *PO, Single-Face Scatterer* for details).

The edge numbering is of similar importance when using GTD on a reflector, since it is possible to trace GTD rays diffracted by selected reflector edges.

Interpolation

The spline interpolation is only possible if there are four or more input points along an edge. If only three points are available they are connected by a circular arc, while a straight line is used if there are only two points.

Examples

In Figure 2 an example is given in which the tabulated rim is displaced to another position in the reflector coordinate system by the attribute `Translation`. The Polar Origin is specified to have status ‘user_defined’ and is given by its position in the reflector coordinate system. It is from the Polar Origin that the ρ - and ϕ -coordinates are defined. Edge 1 is that at direction $\phi = 0$.

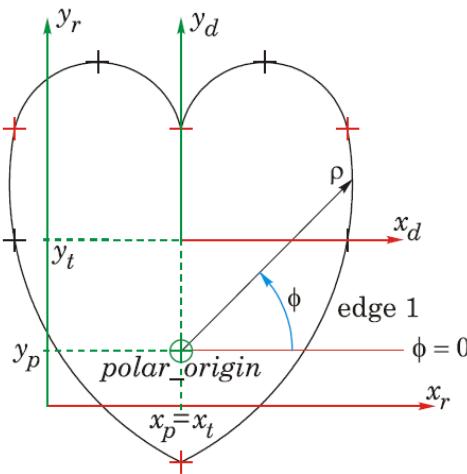


Figure 2

Translated rim. The rim of Figure 1 is here displaced to (x_t, y_t) in the reflector $x_r y_r$ -coordinate system by specifying the values of x_t and y_t in the attribute `Translation`. The $x_d y_d$ -coordinate system is that in which the rim data are given.

The Polar Origin is specified at (x_p, y_p) in the reflector coordinate system, where in this case $x_p = x_t$.

Finally, Figure 3 shows the rim rotated as well. The rim is first translated as in Figure 2, but is now also rotated around the Polar Origin. Again, it is from the Polar Origin that the ρ - and ϕ -coordinates are defined. Edge 1 is that at direction $\phi = 0$. If the rotation angle had been say 180° larger it had been the edge between rim points 7 and 1 (cf. Figure 1) which had been numbered as edge 1.

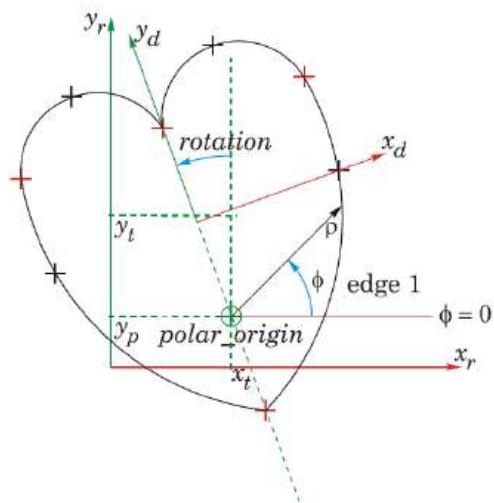


Figure 3

Translated and rotated rim. Again, it is the same tabulated rim as in Figure 1. The rim is, as in Figure 2, displaced to (x_t, y_t) in the reflector $x_r y_r$ -coordinate system by specifying these values in the attribute Translation and the Polar Origin is specified to a point (x_p, y_p) , where in the figure $x_p = x_t$. Finally, the rim is in this case rotated by the attribute Rotation around the Polar Origin. The $x_d y_d$ -coordinate system is that in which the rim data is given.

TABULATED RIM, RHO-PHI-INPUT (tabulated_rim_rho_phi)

Purpose

The class *Tabulated Rim, rho-phi-Input* defines, by means of a table, a *Rim*, typically specified in a *Reflector* object. The table values define points along the rim, in a plane perpendicular to the z -axis of the coordinate system of the reflector, and the values are tabulated in polar (ρ, ϕ) -coordinates.

Links

Classes→*Geometrical Objects*→*Rim*→*Tabulated Rim*→*Tabulated Rim, rho-phi-Input*

Remarks

Syntax

```

<object name> tabulated_rim_rho_phi
(
    file_name : <f>,
    file_number_of_points : struct(in_file:<si>, line:<i>, column:<i>),
    file_rho_phi_values : struct(start_line:<i>, phi_column:<i>,
                                  rho_column:<i>),
    file_corner_points_id : struct(status:<si>, column:<i>,
                                    id_string:<s>),
    unit : <si>,
    polar_origin : struct(x:<rl>, y:<rl>),
    file_form : <si>,
    number_of_points : <i>,
    scaling_factor : <r>,
    min_distance : <rl>,
    rotation : <r>,
    interpolation : <si>,
    list_rim : <si>,
    plot_points : struct(symbol:<si>, size:<rl>),
    external_command : ref(<n>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<f> = file name
<si> = item from a list of character strings

```

Attributes

File Name (*file_name*) [file name].

Name of the file containing the tabulated rim data.

File Number of Points (*file_number_of_points*) [struct].

Specification on how to determine the number of data points in the file. The number of applied data points may be limited by the value of the attribute Number of Points below.

In File (*in_file*) [item from a list of character strings], default: **yes**.

Specifies if the number of data points is to be read in the file.

yes

The number of points is read in the file as specified by line and column.

no

The number of points is not read, but the file is scanned to determine the number of points. The values of line and column are immaterial.

Line (*line*) [integer], default: **2**.

The line in the file where the number of data points shall be found.

Not used for *in_file*: no.

Column (*column*) [integer], default: **1**.

The number of given rim points is read as the *column*'th number/word on the above defined line. Not used for *in_file*: no.

File rho-phi Values (*file_rho_phi_values*) [struct].

Specification of where the data points will be located in the file.

Start Line (*start_line*) [integer], default: **3**.

Number of the first line in the data file with ϕ and ρ -values.

phi-Column (*phi_column*) [integer], default: **1**.

The column on the line where the ϕ -value in degrees will be found.

rho-Column (*rho_column*) [integer], default: **2**.

The column on the line where the ρ -value will be found.

File Corner Points ID (*file_corner_points_id*) [struct].

Specifies how corner points are located in the file.

Status (*status*) [item from a list of character strings], default: **off**.

Status of corner points.

off

No detection of corner points.

on

The corner points will be detected in the rim data file.

Column (*column*) [integer], default: **3**.

The column on the line where the corner point identification has to be found. Not used for *status*: off.

ID-String (*id_string*) [character string], default: **corner_point**.

A string on the line identifying that the point is a corner point. Blanks are not allowed in the string.

Unit (*unit*) [item from a list of character strings], default: **m**.

Defines the unit of length for the ρ -values (mm, cm, m, km, in, ft).
The ϕ values must be in degrees.

Polar Origin (*polar_origin*) [struct].

The (ρ, ϕ) data points are given in the xy -plane and the position of the polar origin ($\rho = 0$) may be specified here. The polar origin also defines the centre of the rim for use in the specification of a central hole in a *Reflector*.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of the polar origin.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of the polar origin.

File Form (*file_form*) [item from a list of character strings], default: **free_format**.

The format of file may be given in two ways:

free_format

The rim data are given in free format on the file (see an example under remarks below).

old_grasp

The rim data are given in the old GRASP rim format described in *Reflector Rim Data*. The following attributes *file_xy_number*, *file_xy_values* and File Corner Points ID shall not be specified and it is not possible to specify corner points.

Number of Points (*number_of_points*) [integer], default: **0**.

Specification of the number of data points to be applied. If zero, all valid data points in the file will be applied. The data are assumed written in the columns specified by the struct members *phi_column* and *rho_column* of the attribute File rho-phi Values. As long as the data in the file follows this pattern, the data are assumed valid data points.

Scaling Factor (*scaling_factor*) [real number], default: **1**.

All ρ -values in the file may be multiplied by this scale factor.

Minimum Distance (*min_distance*) [real number with unit of length], default: **0**.

Minimum distance between points in the xy -plane. If two points are closer than this distance the one point will be neglected (see the remarks for details).

Rotation (*rotation*) [real number], default: **0**.

The rim may be rotated through an angle (in degrees), positive from the x -axis towards the y -axis. The rotation is around the Polar Origin.

Interpolation (*interpolation*) [item from a list of character strings], default: **spline**.

When a point on the rim shall be used in a computation, its position is found by interpolation among the tabulated points. The interpolation may be carried out in either of two ways.

spline

The interpolation is carried out by spline functions (in (ρ, ϕ)) resulting in a smooth rim.

linear

The interpolation is linear (in (x, y)) resulting in a polygonal rim.

List Rim (*list_rim*) [item from a list of character strings], default: **off**.

Specifies whether a list of the generated rim data shall be reproduced in the standard output file.

Plot Points (*plot_points*) [struct].

The sample points defining the rim may be plotted as crosses (corner points in red) and the polar origin as a circle when the reflector is plotted by *Reflector Plot*.

Symbol (*symbol*) [item from a list of character strings], default: **off**.

Selection of plot symbol.

off

No plot of sample points.

cross

The sample points are plotted as crosses.

Size (*size*) [real number with unit of length], default: **0**.

Size of the cross lines.

External Command (*external_command*) [name of an object], default: **blank**.

Reference to an object of class *External Command*. If the command specified by this reference is executed, e.g. if a variable is changed, the tabulated rim file will be read again. This can be used to create a plugin that updates a complex reflector geometry seamlessly whenever the input changes.

Remarks

The rim is specified by tabulated (ρ, ϕ) -values in the reflector coordinate system. It is necessary that the polar origin be selected as an interior point,

so that ρ becomes a single-valued function of ϕ . The values may be in arbitrary order.

The (ρ, ϕ) -values are converted to (x, y) -points by

$$\begin{aligned}x &= x_p + \rho \cos \phi \\y &= y_p + \rho \sin \phi\end{aligned}$$

where (x_p, y_p) is the user-specified Polar Origin.

After the points have been read they are sorted in increasing ϕ . ϕ is positive from the x -axis towards the y -axis and the direction parallel to the x -axis defines $\phi = 0$.

Duplicate points are removed automatically. When Minimum Distance is specified positive and when two consecutive points with a distance less than Minimum Distance are found, that point with the largest ϕ will be removed.

A list of tabulated and regenerated values is given if List Rim is specified to 'on'.

File format

When File Form is specified to 'free_format' it is possible to read a general table of rim data. For example, the file with the rim data may have the following content:

```
polygonal rim
Number of rim points: 8

phi rho

LINE1 270.000 0.200000 meter corner_point
LINE2 0.00000 0.150000 meter
LINE3 33.6901 0.180278 meter corner_point
LINE4 64.8852 0.176706 meter
LINE5 90.0000 0.100000 meter corner_point
LINE6 115.115 0.176706 meter
LINE7 146.310 0.180278 meter corner_point
LINE8 180.000 0.150000 meter
```

The file has three header lines and eight lines of rim points. The number of lines with rim data is given as the 5th column in line 2 of the file. Four of the rim points are corners. In order to read this file the following attribute values shall be given:

File Form: free_format,

File Number of Points: struct(in_file:yes, line:2, column:5),

File rho-phi Values: struct(start_line:4, phi_column:2, rho_column:3),

File Corner Points ID: struct(status:on, column:5, id_string:corner_point),

Number of Points: 0

Corners and edge numbering

Rim points between the tabulated points will in general be determined by interpolation. Thus, if Interpolation is specified to 'spline' the rim will be smooth without sharp corners. If Interpolation is specified to 'linear' the rim will be polygonal.

A point in the data input file may be defined as a corner point by adding a corner point identification string as specified in the attribute File Corner Points ID. The rim interpolation will not be applied across a corner point and the rim will in general not be smooth at a corner.

Note that when linear interpolation is used the rim is polygonal, as mentioned above. However, this is not the same as saying that each point is a corner point. That still needs to be indicated by the File Corner Points ID.

The part of the rim which is between two corner points is called an edge and for PTD and GTD purposes the edges are given consecutive numbers starting from 1 and increasing with increasing ϕ -value. Edge No. 1 is the one which contains the direction $\phi = 0$ from the Polar Origin. (cf. the following figure). The edge numbering is carried out after a possible rotation of the rim data. If a corner point falls exactly at $\phi = 0$ then edge No. 1 is bounded by this corner and the corner with the next higher positive ϕ -value. If possible, such situation should be avoided since even small numerical inaccuracies may result in a ϕ -value being determined slightly positive, thus changing the edge numbering.

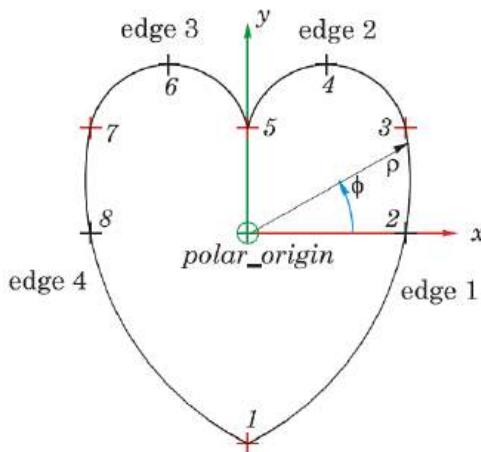


Figure 1

An example on a rim specified by 8 points (each marked by a cross) of which the odd numbered are corner points (marked by red). The Polar Origin is user defined at $(0,0)$ and marked by a cross in a circle.

The numbering scheme for the edges is important when PTD is employed in the field analysis, since it is possible to specify different integration densities along the individual edges of the reflector rim (see the class *PO, Single-Face Scatterer* for details).

The edge numbering is of similar importance when using GTD on a reflector, since it is possible to trace GTD rays diffracted by selected reflector edges.

Interpolation

The spline interpolation is only possible if there are four or more input points along an edge. If only three points are available they are connected by a circular arc, while a straight line is used if there are only two points.

TABULATED RIM (OBSOLETE) (tabulated_rim)

Purpose

The class *Tabulated Rim (Obsolete)* defines, by means of a table, a *Rim* that may be referenced by a *Reflector* object. The table values define points along the rim, in a plane perpendicular to the *z*-axis of the coordinate system of its corresponding reflector. The table values may be specified in either polar or rectangular coordinates. This class is obsolete - use *Tabulated Rim, xy-Input* or *Tabulated Rim, rho-phi-Input* instead.

Links

Classes→*Geometrical Objects*→*Rim*→*Tabulated Rim*→*Tabulated Rim (Obsolete)*

*Remarks***Syntax**

```
<object name> tabulated_rim
(
    file_name          : <f>,
    unit               : <si>,
    centre             : struct(x:<rl>, y:<rl>),
    factor             : <r>,
    n_points           : <i>,
    rotation           : <r>,
    list_rim           : <si>
)
where
<i> = integer
<r> = real number
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

File Name (*file_name*) [file name].

Name of the file containing the tabulated rim data. The contents and format of the file are specified in *Reflector Rim Data*

Unit (*unit*) [item from a list of character strings], default: **m**.

Defines the unit of length for the tabulated rim data (mm, cm, m, km, in, ft).

Centre (*centre*) [struct].

Coordinates of the centre of the rim in the coordinate system of the reflector. The centre is also the centre of the rim for use in the specification of a *PO*, *Single-Face Scatterer* grid or a hole in a *Reflector*.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of centre.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of centre.

Factor (*factor*) [real number], default: **1**.

Scale factor for the rim coordinate data (*x*, *y* or ρ , see the remarks below).

n-Points (*n_points*) [integer], default: **0**.

If larger than 2, the rim is redefined in a polar grid of **n-Points** points, equidistantly spaced in ϕ .

Rotation (*rotation*) [real number], default: **0**.

The rim can be rotated through an angle specified by this value, positive in the direction from the *x*-axis towards the *y*-axis of the reflector coordinate system. The rotation is around the specified centre.

List Rim (*list_rim*) [item from a list of character strings], default: **off**.

Specifies whether a list of the generated rim data shall be reproduced in the standard output file.

Remarks

The rim is specified by tabulated points in a file, given in a plane perpendicular to the *z*-axis of the coordinate system of its corresponding reflector. The table values may be specified in either polar (ρ, ϕ) or Cartesian (*x*, *y*) coordinates, that need not be sorted in any way. The ρ -value is the distance from the Centre to the rim at the corresponding ϕ -angle.

To obtain a fast interpolation, an internal table of rim points is generated. This table stores the (ρ, ϕ)-points equidistantly in ϕ . The ϕ -increment is determined from the specified number of rim points, **n-Points**, or, when **n-Points** = 0, from the smallest increment found in the tabulated values in the file.

The rim interpolation is performed by cubic interpolation in the ρ -values as a function of ϕ . The (*x*, *y*)-points are therefore converted to polar form, (ρ, ϕ), with the rim centre, $\rho = 0$, at (*x*, *y*) as specified by the attribute Centre. It is essential that this centre point is selected as a point interior to the rim, so that ρ becomes a single-valued function of ϕ .

A list of the file-tabulated and the internally generated rim points is produced if **List Rim** is specified to 'on'.

Duplicate points are removed automatically.

SUPERELLIPTICAL RIM (superelliptical_rim)

Purpose

The class *Superelliptical Rim* defines a *Rim* with the shape of a superellipse, typically specified in a *Reflector* object. The rim is specified in a plane perpendicular to the z -axis of the coordinate system of its corresponding reflector.

Links

Classes→*Geometrical Objects*→*Rim*→*Superelliptical Rim*

Remarks

Syntax

```
<object name> superelliptical_rim
(
    centre                  : struct(x:<rl>, y:<rl>),
    half_axis               : struct(x:<rl>, y:<rl>),
    exponent                : <r>,
    rotation                : <r>
)
where
<r> = real number
<rl> = real number with unit of length
```

Attributes

Centre (*centre*) [struct].

Coordinates of the centre of the superelliptical rim in the coordinate system of the reflector.

x (*x*) [real number with unit of length], default: **0**.
x-coordinate of centre.

y (*y*) [real number with unit of length], default: **0**.
y-coordinate of centre.

Half Axis (*half_axis*) [struct].

Half axes of the superelliptical rim. When the rim is not rotated, the axes are parallel to the coordinate axes of the reflector coordinate system.

x (*x*) [real number with unit of length].
Half axis in the *x*-direction.

y (*y*) [real number with unit of length].
Half axis in the *y*-direction.

Exponent (*exponent*) [real number].

Exponent in the equation for a superellipse (see the remarks below).
The exponent must be > 2 .

Rotation (*rotation*) [real number], default: **0**.

The rim can be rotated through an angle (in degrees) specified by Rotation, positive from the *x*-axis towards the *y*-axis. The rotation is around the centre of the rim.

Remarks

The rim is defined by the equation for a superellipse

$$\left| \frac{x - x_c}{a} \right|^m + \left| \frac{y - y_c}{b} \right|^m = 1$$

where (x_c, y_c) is the centre of the superelliptical rim, *a* and *b* are the half axes parallel to the *x*- and *y*-axis, respectively, and *m* is the exponent, which must be larger than 2. With increasing values of the exponent, the elliptical shape approaches a rectangle.

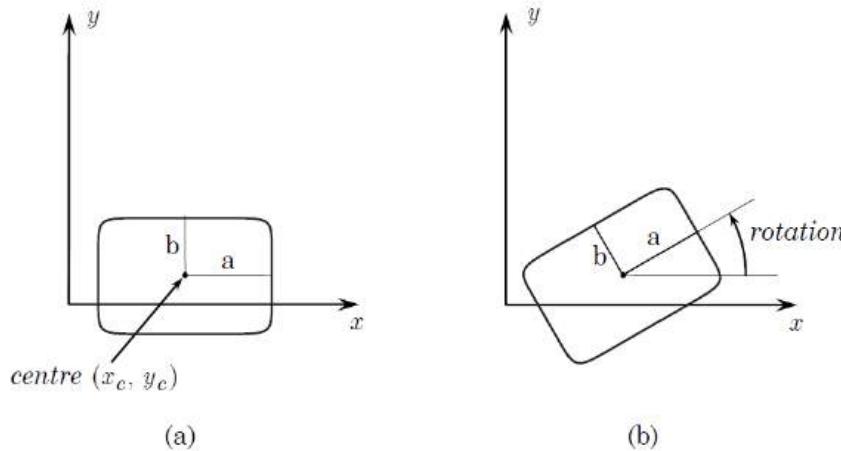


Figure 1

The superelliptical rim, (a) unrotated and (b) rotated in positive direction. The exponent *m* is chosen to 10.

COORDINATE SYSTEMS

Purpose

The following types of coordinate system definitions are available:

Coordinate System

Coordinate System, Euler Angles

Coordinate System, GRASP Angles

Tabulated Coordinate System

Links

Classes→*Geometrical Objects*→*Coordinate Systems*

COORDINATE SYSTEM (coor_sys)

Purpose

The class *Coordinate System* specifies the position and orientation of a coordinate system relative to the global coordinate system or to another coordinate system.

Links

[Classes](#)→[Geometrical Objects](#)→[Coordinate Systems](#)→[Coordinate System](#)

Remarks

Syntax

```
<object name> coor_sys
(
    origin           : struct(x:<rl>, y:<rl>, z:<rl>),
    x_axis          : struct(x:<r>, y:<r>, z:<r>),
    y_axis          : struct(x:<r>, y:<r>, z:<r>),
    base            : ref(<n>)
)
where
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
```

Attributes

Origin (*origin*) [struct].

Coordinates of the origin of the coordinate system.

x (x) [real number with unit of length], default: **0**.

x-coordinate of origin.

y (y) [real number with unit of length], default: **0**.

y-coordinate of origin.

z (z) [real number with unit of length], default: **0**.

z-coordinate of origin.

x-Axis (*x_axis*) [struct].

Coordinates of a point on the positive *x*-axis of the coordinate system.

x (x) [real number], default: **1**.

x-coordinate of *x*-axis.

y (y) [real number], default: **0**.

y-coordinate of *x*-axis.

z (z) [real number], default: **0**.

z-coordinate of *x*-axis.

y-Axis (*y_axis*) [struct].

Coordinates of a point on the positive y -axis of the coordinate system.

x (x) [real number], default: **0**.

x -coordinate of y -axis.

y (y) [real number], default: **1**.

y -coordinate of y -axis.

z (z) [real number], default: **0**.

z -coordinate of y -axis.

Base (*base*) [name of an object], default: **blank**.

Reference to another object of the class *Coordinate System*. If Base is specified, the coordinate system is defined relative to that coordinate system. If Base is omitted, the coordinate system is defined relative to the global coordinate system.

Command Types

The position and orientation of a coordinate system relative to the global coordinate system or to another coordinate system may be determined by the command

Get Coordinate System

The base of a coordinate system may be changed by the command

Change Coordinate System Base.

Remarks

In the GUI, it is possible to orthogonalize one of the directional vectors with respect to the two others. This is done by using the ORTHOGONALIZE functionality in the OBJECT EDITOR of a *Coordinate System*, see Figure 1. When pressing ORTHOGONALIZE on one of the coordinate axes, the corresponding coordinate axis is made orthogonal to the other two.

It is also possible to normalize any of the directional vectors to unit length. This is done by pressing NORMALIZE on one of the coordinate axes.

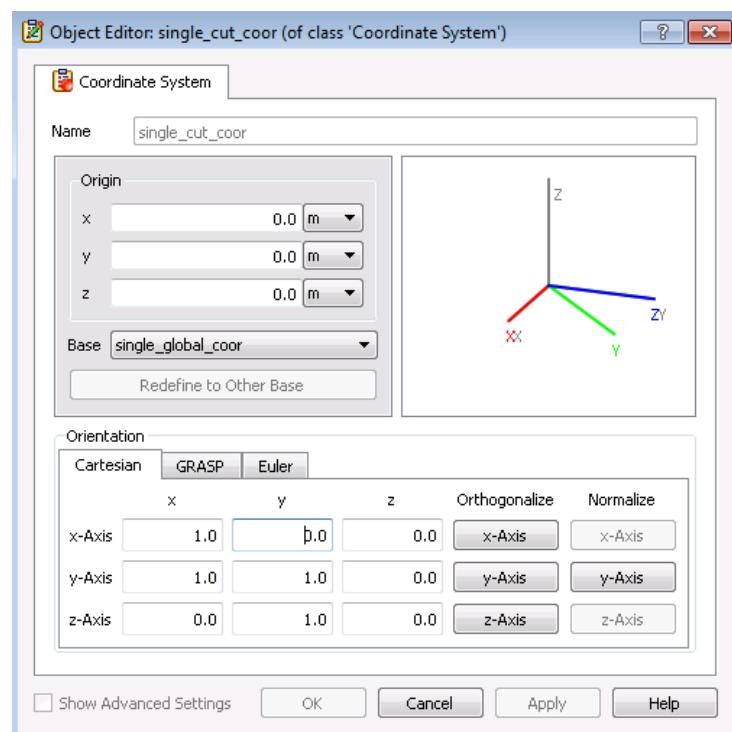


Figure 1

Screen-shot from GRASP showing the OBJECT EDITOR of a *Coordinate System*.

COORDINATE SYSTEM, EULER ANGLES (coor_sys_euler_angles)

Purpose

The class *Coordinate System, Euler Angles* specifies the position and orientation of a coordinate system relative to the global coordinate system, or to another coordinate system base, with the orientation given in terms of the Euler angles α , β , and γ . These Euler angles are described in Section 2.1 of the Technical Description (other definitions of Euler angles exist).

Links

[Classes](#)→[Geometrical Objects](#)→[Coordinate Systems](#)→*Coordinate System, Euler Angles*

Remarks

Syntax

```
<object name> coor_sys_euler_angles
(
    origin                  : struct(x:<rl>, y:<rl>, z:<rl>),
    alpha                   : <r>,
    beta                    : <r>,
    gamma                   : <r>,
    base                    : ref(<n>)
)
where
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
```

Attributes

Origin (*origin*) [struct].

Coordinates of the origin of the coordinate system.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of origin.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of origin.

z (*z*) [real number with unit of length], default: **0**.

z-coordinate of origin.

alpha (*alpha*) [real number], default: **0**.

The first Euler angle, rotation around the *z*-axis, in degrees.

beta (*beta*) [real number], default: **0**.

The second Euler angle, rotation around the new *y*-axis (as resulted from the *alpha*-rotation) in degrees.

gamma (*gamma*) [real number], default: **0**.

The third Euler angle, rotation around the new z -axis (as resulted from the *beta*-rotation) in degrees.

Base (*base*) [name of an object], default: **blank**.

Reference to another object of the class *Coordinate System*. If *Base* is specified, the coordinate system is defined relatively to that coordinate system. If *Base* is omitted, the coordinate system is defined relatively to the global coordinate system.

Command Types

The commands available for *Coordinate System, Euler Angles* are described in *Coordinate System*.

Remarks

The orientation of the axes of a rotated coordinate system, $x_1y_1z_1$, are here defined relatively to the original coordinate system, xyz , by the three Euler angles α , β , and γ . The new $x_1y_1z_1$ -coordinate system may be obtained by three rotations of the original xyz -coordinate system as follows.

First, the xyz -coordinate system is rotated the angle α around its z -axis. This defines an x', y', z' -coordinate system, see the figure below.

Next, the x', y', z' -coordinate system is rotated the angle β around its y' -axis. This defines an x'', y'', z'' -coordinate system.

Finally, the x'', y'', z'' -coordinate system is rotated the angle γ around its z'' -axis. This defines the desired $x_1y_1z_1$ -coordinate system.

Note that the rotations around the axes are with sign. A positive rotation is defined by the right-hand rule: Let the thumb of the right hand be directed along the positive direction of the axis. The other fingers are then pointing in the positive direction of rotation around the axis.

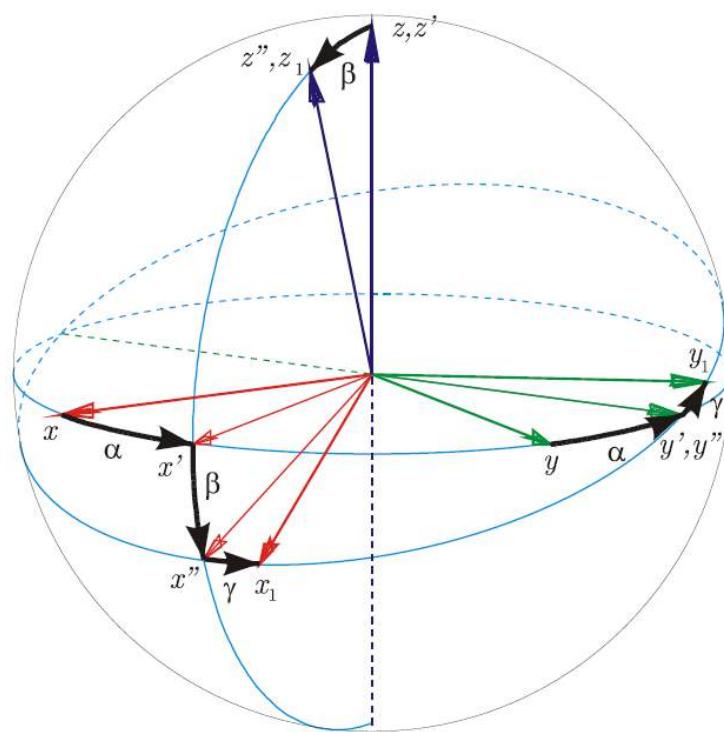


Figure 1

The Euler angles, α , β and γ , rotating the xyz -coordinate system to the $x_1y_1z_1$ -coordinate system.

COORDINATE SYSTEM, GRASP ANGLES (coor_sys_grasp_angles)

Purpose

The class *Coordinate System, GRASP Angles* specifies the position and orientation of a coordinate system relative to the global coordinate system, or to another coordinate system, with the orientation given in terms of the in GRASP applied angles of rotations, θ , φ , and ψ . These GRASP angles are described in Section 2.1 of the Technical Description.

Links

[Classes](#)→[Geometrical Objects](#)→[Coordinate Systems](#)→*Coordinate System, GRASP Angles*

Remarks

Syntax

```
<object name> coor_sys_grasp_angles
(
    origin          : struct(x:<rl>, y:<rl>, z:<rl>),
    theta           : <r>,
    phi             : <r>,
    psi             : <r>,
    base            : ref(<n>)
)
where
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
```

Attributes

Origin (*origin*) [struct].

Coordinates of the origin of the coordinate system.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of origin.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of origin.

z (*z*) [real number with unit of length], default: **0**.

z-coordinate of origin.

theta (*theta*) [real number], default: **0**.

The first GRASP angle, in degrees.

phi (*phi*) [real number], default: **0**.

The second GRASP angle, in degrees. (θ, φ) is the direction of the new *z*-axis.

`psi (psi)` [real number], default: **0**.

The third GRASP angle, in degrees, which defines a rotation around the new z -axis.

`Base (base)` [name of an object], default: **blank**.

Reference to another object of the class *Coordinate System*. If `Base` is specified, the coordinate system is defined relative to that coordinate system. If `Base` is omitted, the coordinate system is defined relatively to the global coordinate system.

Command Types

The commands available for *Coordinate System, GRASP Angles* are described in *Coordinate System*.

Remarks

The three angles of rotation, θ , φ and ψ , are generally applied in GRASP when defining the orientation of one coordinate system with respect to another.

The directions of the axes of the new coordinate system $x_1y_1z_1$ are defined such that the z_1 -axis is pointing in the direction (θ, φ) of the original xyz -coordinate system. The orientations of the x_1 - and y_1 -axes are given by the third angle, ψ .

The orientation of the axes is defined by the following set of rotations:

First, the original xyz -coordinate system is rotated the angle φ around its z -axis, cf. the figure below. The hereby defined coordinate system is denoted by XYZ .

Next, the **original** xyz -coordinate system is rotated the angle θ around the Y -axis. This new coordinate system is denoted by x', y', z' .

Finally, the x', y', z' -coordinate system is rotated the angle ψ around the z' -axis, whereby it reaches the desired position of the new $x_1y_1z_1$ -coordinate system.

Note that the rotations around the axes are with sign. A positive rotation is defined by the right-hand rule: Let the thumb of the right hand be directed along the positive direction of the axis. The other fingers are then pointing in the positive direction of rotation around the axis.

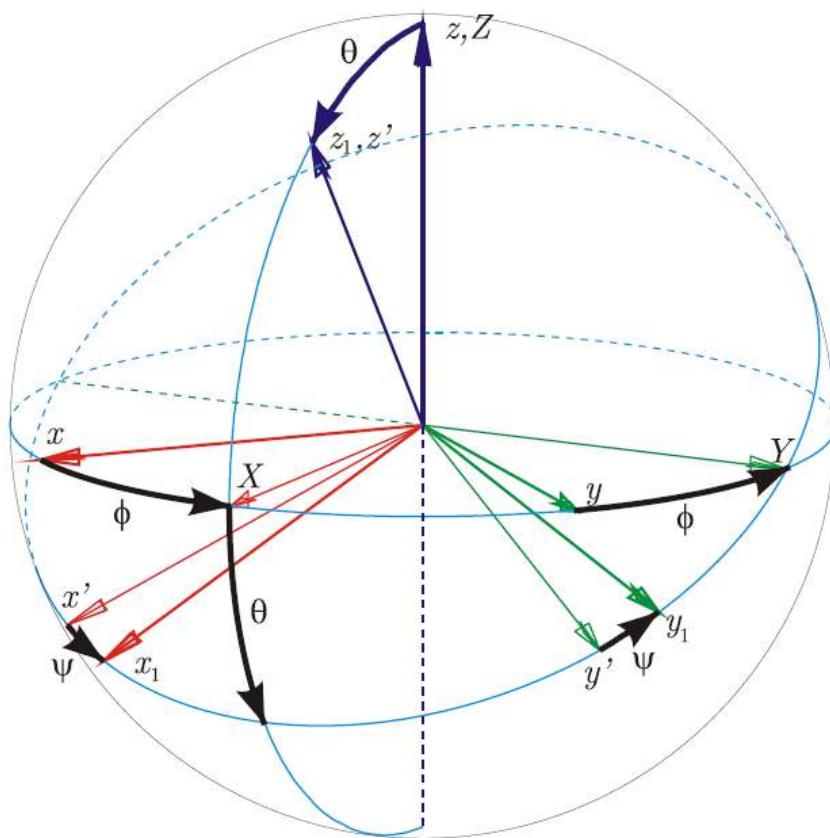


Figure 1

The GRASP angles, θ , φ and ψ , rotating the xyz -coordinate system to the $x_1y_1z_1$ -coordinate system.

TABULATED COORDINATE SYSTEM (tabulated_coor_sys)

Purpose

The class *Tabulated Coordinate System* reads specifications for the position and orientation of a coordinate system relative to the global coordinate system, or to another coordinate system, from a user-specified file.

Links

[Classes](#)→[Geometrical Objects](#)→[Coordinate Systems](#)→*Tabulated Coordinate System*

Syntax

```
<object name> tabulated_coor_sys
(
    file_name : <f>,
    base       : ref(<n>)
)
where
<n> = name of an object
<f> = file name
```

Attributes

File Name (*file_name*) [file name].

Name of the file containing the coordinate system specification. The contents and file format are specified in *Definition of Tabulated Coordinate System*.

Base (*base*) [name of an object], default: **blank**.

Reference to another object of the class *Coordinate System*. If Base is specified, the coordinate system is defined relative to that coordinate system. If Base is omitted, the coordinate system is defined relative to the global coordinate system.

Command Types

The commands available for *Tabulated Coordinate System* are described in *Coordinate System*.

DERIVED GEOMETRY DATA

Purpose

The following classes for analysis and output specifications are available for tabulating reflector data:

To generate tables with one or more of the next following two types of data, use :

Reflector Data Output

to generate tables with reflector surface data only, use :

Surface Data Output

to generate tables with reflector rim data only, use :

Rim Data Output

The classes are only for data management, and have no impact on RF calculations.

The following class is applied in GTD calculations to check if a given ray path is blocked by other scatterers,

*Blockage Check***Remarks**

The last two classes make it possible to create output files with geometrical surface data or geometrical rim data, respectively. These classes may be applied directly.

Alternatively, the first class, *Reflector Data Output*, may be applied. Hereby, one or both of the other classes may be applied for specifying the respective data files.

Links

Classes→*Geometrical Objects*→*Derived Geometry Data*

REFLECTOR DATA OUTPUT (reflector_data_output)

Purpose

This class is used for output of data for reflector structures. One or more attributes for output may be specified. Alternatively, one or more of the derived classes

Surface Data Output

Rim Data Output

Electrical Properties Data Output

may be applied directly.

Links

Classes→*Geometrical Objects*→*Derived Geometry Data*→*Reflector Data Output*

Remarks

Syntax

```
<object name> reflector_data_output
(
    surface_data_output      : ref(<n>),
    rim_data_output          : ref(<n>),
    el_prop_data_output      : ref(<n>)
)
where
<n> = name of an object
```

Attributes

Surface Data Output (*surface_data_output*) [name of an object], default: **blank**.

Reference to an object of the class *Surface Data Output*. The surface *z*-values and/or derivatives will be calculated in a regular *xy*-grid in this object. The *z*-values can be tabulated on a data file for later use and the derivatives can be listed in a table.

Rim Data Output (*rim_data_output*) [name of an object], default: **blank**.

Reference to an object of the class *Rim Data Output*. The rim position as *xy*-values will be calculated in regular ϕ -cuts. The calculated rim values can be written on a data file for later use or listed in a table as $\rho(\phi)$ together with its derivatives.

El Prop Data Output (*el_prop_data_output*) [name of an object], default: **blank**.

Reference to an object of the class *Electrical Properties Data Output*. The electrical properties are calculated in a regular $\theta\phi$ -grid with respect to the surface normal and can be listed in a table or written on data file for later use.

Command Types

The data files given in the specified objects are generated by executing the command *Get Reflector Data*.

Remarks

Generally, only one attribute is required. The attributes are not related and can be specified separately.

SURFACE DATA OUTPUT (surface_data_output)

Purpose

The class *Surface Data Output* is used to produce a file with geometrical data of the surface of a particular reflector. In a regular *xy*-grid the surface values (*z*) and, if requested, its derivatives are determined.

Links

Classes→*Geometrical Objects*→*Derived Geometry Data*→*Surface Data Output*

Remarks

Syntax

```
<object name> surface_data_output
(
    file_name           : <f>,
    file_form          : <si>,
    x_range            : struct(start:<rl>, end:<rl>, np:<i>),
    y_range            : struct(start:<rl>, end:<rl>, np:<i>),
    xy_unit           : <si>,
    z_unit             : <si>,
    values             : <si>,
    coor_sys          : ref(<n>),
    list               : <si>
)
where
<i> = integer
<n> = name of an object
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

File Name (*file_name*) [file name].

Name of the file to which the calculated surface data are written. The file will be written according to the format defined by the attribute *file_form*. The recommended file extension is *.sfc*.

File Form (*file_form*) [item from a list of character strings], default: **xy-grid**.

Format of the file to which the calculated surface data are written. The file may be written in the following formats:

xy-grid

Format for a general surface described by a regular *xy*-grid, cf. *Surface Data in Rectangular Grid*.

rho-z-arc

Format describing a rotational symmetric surface by a $\rho(z)$ -arc. This format may be used in a *Rotationally Symmetric Surface* object. Notice, that (x-range_start, y_range_start) is used as center of the arc, where $\rho = 0$. Each line in the file contains ρ , z , x , y values.

xyz-points

Format for a general surface described by points given by the coordinates (x,y,z), cf. *Surface Data in Irregular Points*.

x-Range (x_range) [struct].

Grid specification in x range.

Start ($start$) [real number with unit of length], default: **0**.

Start value of the x range.

End (end) [real number with unit of length], default: **0**.

End value of the x range.

Np (np) [integer], default: **1**.

The number of values in the x range.

y-Range (y_range) [struct].

Grid specification in y range:

Start ($start$) [real number with unit of length], default: **0**.

Start value of the y range.

End (end) [real number with unit of length], default: **0**.

End value of the y range.

Np (np) [integer], default: **1**.

The number of values in the y range.

xy-Unit (xy_unit) [item from a list of character strings], default: **m**.

Defines the unit of length in which the xy -grid are specified in the file (mm, cm, m, km, in, ft).

z-Unit (z_unit) [item from a list of character strings], default: **m**.

Defines the unit in which the z -values are written in the file (mm, cm, m, km, in, ft). The first order derivatives are expressed without dimension, and the second order derivatives are expressed with the dimension (z -unit)-1.

Values ($values$) [item from a list of character strings], default: **z**.

Determines which data shall be calculated for the surface. The surface is given as a mathematical function $z = z(x,y)$ in the reflector coordinate system.

z

The z -values of the surface are generated.

dz/dx

The first order derivatives of the surface along x are calculated.

dz/dy

The first order derivatives of the surface along y are calculated.

dzz/dxx

The second order derivatives of the surface along x are calculated.

dzz/dxy

The mixed second order derivatives of the surface are calculated.

dzz/dyy

The second order derivatives of the surface along y are calculated.

Coordinate System (coor_sys) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes, defining the coordinate system in which the surface is tabulated.

List (list) [item from a list of character strings], default: **off**.

Generates a list of the read surface grid data in the standard output file.

off

No list is generated.

on

A list is generated.

Command Types

The data files given in the specified objects are generated by executing the command *Get Reflector Data*.

Remarks

The file is only written if a file name is specified. If a file name is not specified, the output may be written in the output file by specifying *list* to 'on'.

The first line in the file is a header, which will contain the name of the object generating the file. The xy -grid formatted file can be used as input for a tabulated surface of the type *Regular xy-Grid*, and the $\rho(z)$ -formatted file can be used as input for a tabulated surface of the type *Rotationally Symmetric*. Note, that in the case of the $\rho(z)$ -formatted file it will be assumed that

the reflector has a tip at $\rho = 0$. This may be changed in the input to the *Rotationally Symmetric* surface.

In order to generate a $\rho(z)$ -arc, only one cut shall be given, i.e. np must be given the value 1 (one) in one of the attributes x_range or y_range . See also the example below.

When the spacing in the grid is chosen, care shall be taken to include the desired surface variations, and only these. For a complex surface, the spacing must be dense to describe the variations but if the surface is based on measurements with some inaccuracies these may cause unrealistic surface derivatives if a too dense spacing is chosen.

The measurement unit of the z -values is *z-unit* as specified. The first order derivatives have no measurement unit. Thus, if *xy-unit* and *z-unit* are different then z is re-calculated in *xy-units* when the derivatives are calculated. The second order derivatives have the unit (*z-unit*)-1.

When a $\rho(z)$ -arc is specified, the ρ -value is determined relatively to (x_s, y_s) , where x_s and y_s are the values of *start* in the attributes x_range and y_range , respectively. The ρ -value is thus defined by

$$\rho = \sqrt{(x - x_s)^2 + (y - y_s)^2}. \quad (1)$$

Example

An example in which this facility may be applied is for the geometrical specification of a ring-focused antenna. An antenna of this type may as well be designed in the Wizard of GRASP but the example illustrates how this facility may be utilized for more special designs.

The ring-focused antenna is rotationally symmetric and may be designed according to Geometrical Optics, cf. the cross section shown in the following figure. The subreflector in the cross section is an arc of an ellipse with one focus at the feed, F , and the other focus at F_r . The latter is further the focus of a parabola which – when rotated – constitutes the main reflector. By the rotation F_r becomes the ring focus of the main reflector.

The apex of the rotated parabola is in the figure vertically below F_r . This is not at the axis of rotation and the main reflector surface is therefore not a simple paraboloid.

The geometry of the reflector surfaces may be specified from the geometry of a Gregorian double reflector system which posses the requested geometry in its symmetry plane as illustrated in the following figure.

When the reflectors of the Gregorian antenna are defined in GRASP the surface contours in the plane of symmetry may – for each reflector – be listed in a file by an object of the present class, *Surface Data Output*. As only the elliptical and the parabolic arcs shall be applied for defining the respective rotationally symmetric surface in a $\rho(z)$ -formatted file, *file_form* must be specified to 'rho-z-arc'. We will assume that the *x*-axis is in the plane of symmetry in the figure above, and the *y*-axis is perpendicular to this plane. The attribute *y_range* shall then be specified as

y_range: struct(*start*:0 m, *end*:0 m, *np*:1),

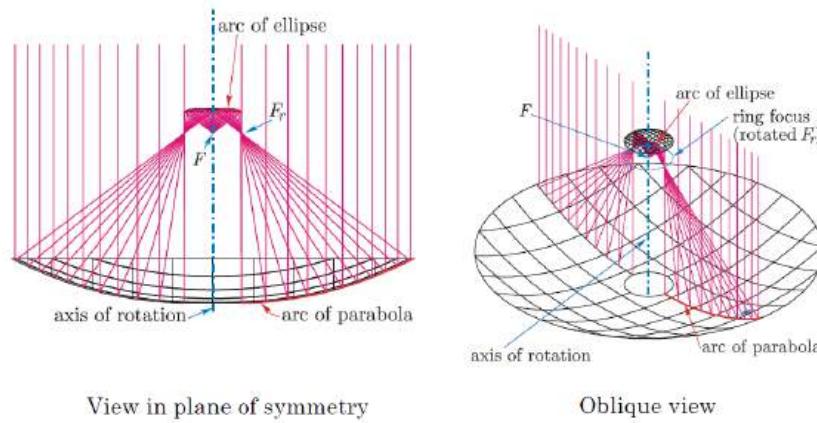


Figure 1 The ring-focused antenna is rotationally symmetric and designed by ray optics in a plane of symmetry. Ray paths are drawn from F .

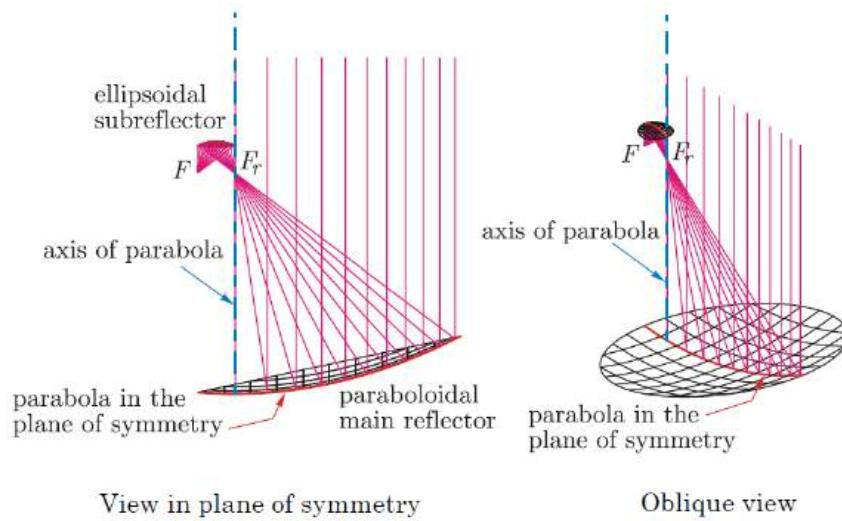


Figure 2 Gregorian reflector antenna. The ellipsoidal subreflector has foci at F and F_r . The feed is placed at F , and F_r is also focus for the paraboloidal main reflector.

while the specification of `x_range` shall be according to the surface dimensions in the plane of symmetry.

An important detail is here that ρ in the file shall start (i.e. have $\rho = 0$) at the axis of rotation for the ring-focused antenna. This corresponds to a negative value of `start` in `x_range`, cf. Eq. (1); in this example it is equal to the negative value of the distance FF_r . For this reason, the main reflector of the Gregorian antenna shall be defined with a **Rim** which includes this start point in the reflector definition.

The hereby generated files (one for each reflector) may next be applied as input in objects of class **Rotationally Symmetric** resulting in the surfaces of the ring-focused antenna as shown in the first figure.

RIM DATA OUTPUT (rim_data_output)

Purpose

The class **Rim Data Output** is used to produce a file with tabulated rim data for a particular reflector. The rim data are written as (x, y) -coordinates as function of the azimuthal angle ϕ . Additionally, the following parameters may be calculated and written in the output file: $d\rho(\phi)/d\phi, d^2\rho(\phi)/d\phi^2$; ρ being the radial distance to the rim point.

Links

[Classes](#)→[Geometrical Objects](#)→[Derived Geometry Data](#)→[Rim Data Output](#)

Remarks

Syntax

```
<object name> rim_data_output
(
    file_name           : <f>,
    phi_start          : <r>,
    phi_end            : <r>,
    n_points           : <i>,
    output_unit        : <si>,
    coor_sys           : ref(<n>),
    list               : <si>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

File Name (*file_name*) [file name].

Name of file to which the calculated rim data are written. For the format of the file, see the remarks below. The recommended file extension is *.cut*.

phi Start (*phi_start*) [real number], default: **0**.

Start value of azimuthal angle ϕ for rim specification.

phi End (*phi_end*) [real number], default: **360**.

End value of azimuthal angle ϕ for rim specification.

N Points (*n_points*) [integer], default: **37**.

Number of ϕ -values tabulated for the rim.

Output Unit (*output_unit*) [item from a list of character strings], default: **m**.

Defines the unit of length for the output data(mm, cm, m, km, in, ft).

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes, defining the coordinate system in which the rim is tabulated.

List (*list*) [item from a list of character strings], default: **on**.

Generates a list of the rim data in the standard output file (see also the remarks below).

on

A list is generated.

off

No list is generated.

Command Types

The *Rim Data Output* class is derived from the class *Reflector Data Output* in which the available commands are described.

Remarks

The rim is projected into the *xy*-plane of *coor_sys*, and ϕ is the azimuthal angle in this plane. The rim is tabulated with a regular ϕ -spacing for the ϕ -values in the specified interval.

The file is only written if a file name is specified. If a file name is not specified, the output may be written in the output file by specifying *list* to 'on'.

The file is written in free format (unless *coor_sys* is unspecified) and it may be used as input for an *Tabulated Rim, xy-Input*.

If *coor_sys* is unspecified, the file is written as formatted in the format described in *Reflector Rim Data* for compatibility with previous versions of the program. In this case the file contains a table with the rim position given as $\rho(\phi)$; ρ being the radial distance to the rim at the angle ϕ . The first line in the written file is a header, which will contain the name of the object generating the file.

When *list* is set to 'on' then also the values of $\rho(\phi)$, $d\rho(\phi)/d\phi$ and $d^2\rho(\phi)/d\phi^2$, the first and second derivative of $\rho(\phi)$, are written in the standard output file.

BLOCKAGE CHECK (blockage_check)

Purpose

The class **Blockage Check** is used to check if blockage is present in a reflector antenna system. The required ray path is traced from a source through the reflectors to the far field. If blockage of this ray path from any scatterer in the total system is found, the information on the blocked ray traces will be written in the log and in the output file.

This class is only available with the Multi GTD add-on.

Links

[Classes](#)→[Geometrical Objects](#)→[Derived Geometry Data](#)→[Blockage Check](#)

Remarks

Syntax

```
<object name> blockage_check
(
    theta_range           : struct(start:<r>, end:<r>, np:<i>),
    phi_range             : struct(start:<r>, end:<r>, np:<i>),
    scatterer_order       : sequence(ref(<n>), ...),
    trace_list            : <si>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<si> = item from a list of character strings
```

Attributes

theta-Range (*theta_range*) [struct].

Defines the range of the polar angle θ in the radiation from the source (see the remarks below).

Start (*start*) [real number], default: **0**.

Start value of the polar coordinate θ .

End (*end*) [real number], default: **0**.

End value of the polar coordinate θ .

Np (*np*) [integer], default: **1**.

Number of θ -values in each polar cut.

phi-Range (*phi_range*) [struct].

Defines the range of the azimuthal angle ϕ in the radiation from the source (see the remarks below).

Start (*start*) [real number], default: **0**.

Start value of the azimuthal coordinate ϕ .

End (*end*) [real number], default: **0**.

End value of the azimuthal coordinate ϕ .

Np (*np*) [integer], default: **1**.

Number of polar cuts ϕ -values.

Scatterer Order (*scatterer_order*) [sequence of names of other objects], default: **blank**.

Sequence of references to objects of the class *Scatterer*. The sequence defines the order of the scatterers between which a ray tracing is performed. If no references are given then the blockage of the direct ray from the source to the field points is investigated.

Trace List (*trace_list*) [item from a list of character strings], default: **off**.

A list of the traced rays and their intersection points with the scatterers may be produced in the standard output file.

off

No list is generated.

on

A list is generated.

Command Types

The blockage check is initiated by the command

Check Blockage

Remarks

Objects of this class are intended to identify blockage in complex reflector antenna systems on basis of a ray optical technique denoted forward ray-tracing. Rays are emitted from the source within a specified angular range and should hit the reflectors in the order given by the attribute *scatterer_order*. If the reflectors are hit in a different order or if other scatterers are hit then a blockage is identified.

Consider, for example, a double reflector system with several feeds. Rays from each feed shall be reflected in the subreflector and next in the main reflector to reach the far field. A blockage may occur if a ray, after reflection in the main reflector, hits the subreflector once more.

The ray directions from the source are defined in usual spherical (θ, ϕ) -coordinates given in the source coordinate system.

$$\hat{r} = \hat{x} \sin \theta \cos \phi + \hat{y} \sin \theta \sin \phi + \hat{z} \cos \theta \quad (1)$$

In the specified angular range θ runs in cuts with fixed ϕ through the values

$$\theta_i = \theta_{start} + \Delta\theta \cdot (i - 1), \quad i = 1, 2, \dots, n_\theta \quad (2)$$

with

$$\Delta\theta = (\theta_{end} - \theta_{start})/(n_\theta - 1)$$

where θ_{start} and θ_{end} are the members *start* and *end*, respectively, of the attribute *theta_range*, and n_θ is the member *np* of the same attribute.

The ϕ -angle is increased equidistantly from one cut to the next through the values

$$\phi_j = \phi_{start} + \Delta\phi \cdot (j - 1), \quad j = 1, 2, \dots, n_\phi \quad (3)$$

with

$$\Delta\phi = (\phi_{end} - \phi_{start})/(n_\phi - 1)$$

Here ϕ_{start} and ϕ_{end} are the members *start* and *end*, respectively, of the attribute *phi_range*, and n_ϕ is the member *np* of the same attribute.

Information on blockage or not is written in the standard output file of GRASP. For details on the blockage the attribute *trace_list* should be specified to 'on'.

MOVEMENT DEFINITION (movement_definition)

Purpose

An object of the class *Movement Definition* defines how a coordinate system and all objects defined herein are moved in rotation(s) and translation(s).

This class is only available with the Coupling add-on.

Links

Classes→*Geometrical Objects*→*Movement Definition*

Remarks

Syntax

```

<object name> movement_definition
(
  coor_sys_for_movements      : ref(<n>),
  moved_coordinate_systems    : sequence(ref(<n>), ...),
  movements                   : sequence(
    struct(
      movement:<si>,
      axis:<si>,
      start:<r>,
      end:<r>,
      number:<i>,
      length_unit:<si>),
    ...),
  list                         : <si>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<si> = item from a list of character strings

```

Attributes

Coor Sys For Movements (*coor_sys_for_movements*) [name of an object].

Reference to an object of one of the *Coordinate Systems* classes. The performed movements are defined in this coordinate system or in coordinate systems derived from this.

Moved Coordinate Systems (*moved_coordinate_systems*) [sequence of names of other objects].

Sequence of references to one or more objects of class *Coordinate System*. These coordinate systems and the objects defined herein will be moved as specified. Also coordinate systems which are derived from the Moved Coordinate Systems and objects defined in these are included. The specified Moved Coordinate Systems must be

coordinate systems derived from the Coor Sys For Movements or the Coor Sys For Movements itself.

Movements (*movements*) [sequence of structs].

Defines the movements to be performed. Each movement is specified in a struct giving the type and the extension of the movement. The extension of the movement is from a Start value to an End value through a Number of values and at each value the specified geometry is moved to that position. Initially, all objects are moved to their respective Start position. Then all steps of the last defined movement are performed before the first step of the second to last specified movement is performed and so on. Thus, for each step of a movement all steps of all subsequently defined movements are carried out. See also the remarks below.

Movement (*movement*) [item from a list of character strings], default: **translation_along**.

Specifies if the movement is a translation or a rotation.

translation_along

The movement is a translation.

rotation_around

The movement is a rotation.

Axis (*axis*) [item from a list of character strings], default: **x_original**.

Alphanumeric identification string of the axis of translation or rotation. It is possible to translate along (or rotate around) 'original' axes as well as 'new' axes: Let CS_o denote the coordinate system specified in the attribute Coor Sys For Movements and let CS_n denote a coordinate system which originally was identical to CS_o but has followed the steps carried out by the previously defined movements. By previously defined movements is here understood the movements which have been defined before this movement in the sequence of movements (the sequence of structs) specified in the present attribute Movement. The 'original' axes are then axes parallel to those of CS_o . If the actual Movement is a rotation then the axis of rotation is through the origin of CS_n . The 'new' axes are the axes of CS_n . The movements may thus be along (around) one of the following axes:

x_original

Translation is along the 'original' *x*-axis; rotation is around the 'original' *x*-axis, positive from the *y*-axis towards the *z*-axis.

y_original

Translation is along the 'original' *y*-axis; rotation is around the 'original' *y*-axis, positive from the *z*-axis towards the *x*-axis.

z_original

Translation is along the 'original' *z*-axis; rotation is around the 'original' *z*-axis, positive from the *x*-axis towards the *y*-axis.

x_new

Translation is along the 'new' *x*-axis; rotation is around the 'new' *x*-axis, positive from the *y*-axis towards the *z*-axis.

y_new

Translation is along the 'new' *y*-axis; rotation is around the 'new' *y*-axis, positive from the *z*-axis towards the *x*-axis.

z_new

Translation is along the 'new' *z*-axis; rotation is around the 'new' *z*-axis, positive from the *x*-axis towards the *y*-axis.

Start (*start*) [real number], default: **0**.

The start value of the movement. For translations the start value shall be given in the unit specified by Length Unit below, for rotations the start value shall be given in degrees.

End (*end*) [real number], default: **0**.

The end value of the movement. For translations the end value shall be given in the unit specified by Length Unit below, for rotations the end value shall be given in degrees.

Number (*number*) [integer], default: **0**.

The number of values in the movement (including the Start value and the End value). If Number is specified to one, then only one value in the movement will be applied namely the start value. If Number is specified to zero, the actual movement will be neglected (see also the remarks below).

Length Unit (*length_unit*) [item from a list of character strings], default: **m**.

The unit for the Start and the End values in a translation; must be a valid unit of length (mm, cm, m, km, in, ft). The item is dummy for rotations (the item may be omitted or a unit of length may be specified).

List (*list*) [item from a list of character strings], default: **off**.

Specifies if the movements shall be listed in the standard output file.

off

No list is generated.

on

All movements are listed.

Command Types

By the attribute *movements* a series of positions of the moved coordinate systems are defined. Moving the coordinate systems to one of these positions may be performed by the command

Activate Movement

Remarks

This section contains discussions on the following topics

- The positions achieved in the movements
- Index of the movements
- Output of values in cuts or grids
- Examples on movements: antenna rotations
 - Azimuth over elevation set-up
 - * Scan in azimuth and step in elevation
 - * Scan in elevation and step in azimuth
 - Elevation over azimuth set-up
 - * Scan in azimuth and step in elevation
 - * Scan in elevation and step in azimuth

The positions achieved in the movements

A translation may be specified to be carried out along, say, the x -axis. In the translation, the moved geometry is translated in such a way that a point originally at $x = 0$ will be moved from $x = x_s$ (given by the attribute *start*) to $x = x_e$ (given by the attribute *end*). The movement is linear from the start position x_s to the end position x_e with totally N positions (given by the attribute *number*):

$$x_i = x_s + (i - 1)\Delta x, \quad i = 1, 2, \dots, N \quad (1)$$

where the step in the translation is

$$\Delta x = \frac{x_e - x_s}{N - 1}$$

An example with $N = 5$ is given in Figure 1.

If *number* is specified to 1 then the movement is to the only position at

$$x_1 = x_s$$

and if *number* is specified to 0 then there is no movement corresponding to

$$x_1 = 0$$

There are no restrictions on x_s and x_e .

A translation along an inclined line is carried out by introducing a coordinate system with an axis parallel to the inclined line. The translation can then be performed along this axis.

The rotations are defined in a similar way.

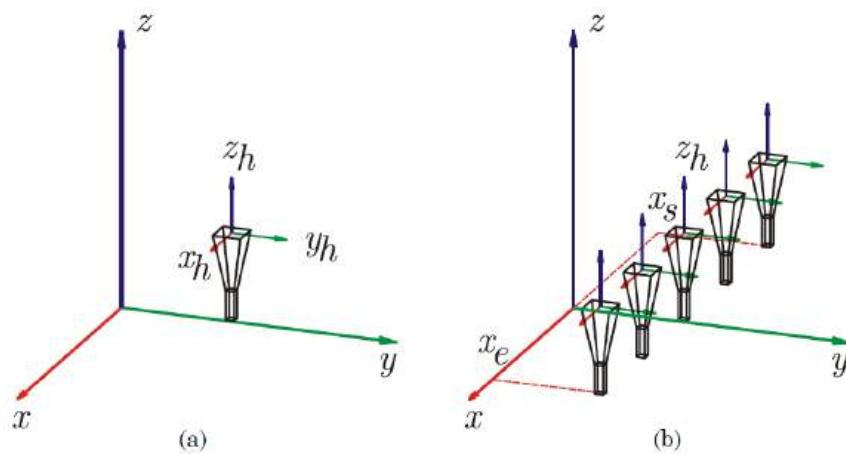


Figure 1

Illustration of translation of a horn antenna along x . (a) shows the geometry. The horn is defined in the horn coordinate system with axes x_h , y_h and z_h . This is again defined in a global coordinate system with axes x , y and z . The *coor_sys_for_movements* is the global xyz -coordinate system; the *moved_coordinate_systems* is the horn coordinate system $x_hy_hz_h$. (b) illustrates then the movement defined by a *start-value*, x_s , an *end-value*, x_e , and a *number* of movements which is 5. During the translation, the horn coordinate system with the horn takes the 5 positions shown.

Index of the movements

The index of the movements follows the order in which the positions of the movements are taken. Thus, for a single movement the index of the movement is given by i in Eq. (1) above.

When more movements are involved, such as a translation followed by a rotation, these are taken in the following order, given by index i , by combining the indices i_1 and i_2 of the movements

$$i = (i_1 - 1)n_2 + i_2$$

where i_1 is the index of the first movement, the translation,

$$i_1 = 1, 2, \dots, n_1$$

and i_2 is the index of the second movement, the rotation,

$$i_2 = 1, 2, \dots, n_2$$

n_m is the number of positions in the m 'th movement. For this case, in which we have two movements,

$$m = 1, 2$$

It is seen that the movements are carried out in such an order that the index i_2 runs through all its values for each value of i_1 .

The same principle applies for more than two movements, the last defined movement is the fastest varying movement and the index of the movement becomes

$$i = (i_1 - 1) \prod_{m=2}^M n_m + (i_2 - 1) \prod_{m=3}^M n_m + \dots + (i_{M-1} - 1) n_M + i_M$$

where M is the number of movements. The order of the movements is the same as the order in which the movements are specified in the attribute *movements*.

If the calculations are specified for a set of frequencies (or wavelengths) in an object of one of the *Frequency* classes, then the frequency is handled as the slowest varying parameter.

Output of values in cuts or grids

When electrical data are determined from a movement definition these may be printed in a file according to specifications in the object defining the electrical data and the file.

If the output to the file is specified as cuts, then each cut corresponds to a full move of the last defined movement (the fastest varying movement), i.e. for i_M running as

$$i_M = 1, 2, \dots, n_M$$

while steps in earlier defined movements (slower movements) cause a new cut for each step.

If the output to the file is specified as grids, then each grid corresponds to full moves of the two last defined movements (the two fastest varying movements), i.e. for i_{M-1} running as

$$i_{M-1} = 1, 2, \dots, n_{M-1}$$

and for each i_{M-1} then i_M shall run as

$$i_M = 1, 2, \dots, n_M$$

Steps in earlier defined movements (slower movements) cause a new grid for each step.

Examples on movements: Antenna rotations

As examples in movement definitions we will consider typical measurements of antenna patterns as carried out in azimuth and elevation. The antenna is then mounted in a two-axes positioner. The one positioner rotates the antenna in azimuth (around a vertical axis) and the other positioner rotates the antenna in elevation (around a horizontal axis).

As there are two positioners the one is mounted upon the other resulting in different schemes for the rotation: In the azimuth over elevation set-up the azimuth positioner is mounted upon the elevation positioner and, vice versa, in the elevation over azimuth set-up the elevation positioner is mounted upon the azimuth positioner.

In both set-ups the measurements may be carried out as scan in azimuth and step in elevation (fastest rotation around the azimuth axis) or as scan in elevation and step in azimuth (fastest rotation around the elevation axis).

All four cases are described in the following.

It shall be noted that in the movement definitions it is assumed that it is the same coordinate system (the *moved_coordinate_systems*) which passes through all the movements. In the following figures we have drawn different coordinate systems, the azimuth coordinate system and the elevation coordinate system in order to reflect the physical designation of the axes of rotation.

Azimuth over elevation set-up

That an antenna is rotated in an azimuth over elevation set-up means that the azimuth positioner is mounted upon the elevation positioner and a rotation in elevation will thus rotate the azimuth axis while a rotation in azimuth has no influence on the elevation axis.

In the following figures a fixed *xyz*-coordinate system is placed, say on the ground at the measurement site. At some height (1.0 m) above the origin of the *xyz*-system the elevation positioner is mounted. The axes which move in elevation is denoted x_{el} , y_{el} and z_{el} . The azimuth positioner is then mounted on the y_{el} -axis, and the axes which are moved in the azimuth rotation is denoted x_{az} , y_{az} and z_{az} . For illustration of the rotations an antenna horn is mounted on the z_{az} -axis at some distance (1.5 m) from the origin.

In order to specify this in GRASP we define a set of coordinate systems, cf. Figure 2. The global coordinate system is fixed at the ground of the measurement site, the *xyz*-system:

```
CS_ground coor_sys ()
```

One meter above this we have the base in which the movements are defined, *CS_base*. By default, the axes are parallel to those of *CS_ground*:

```
CS_base coor_sys
(
    origin : struct(x: 0.0 m, y: 1.0 m, z: 0.0 m),
    base : ref(CS_ground)
)
```

Upon the base (*CS_base*) we define the elevation rotation. The coordinate system (*CS_elev*) is not yet rotated:

```
CS_elev coor_sys
(
    base : ref(CS_base)
)
```

Finally, upon the elevation positioner (in *CS_elev*) the azimuth positioner is placed. For easier illustration in the figures the origin of the azimuth positioner coordinate system (*CS_azim*) is placed 2.0 m above the origin of

the elevation coordinate system (CS_elev). This has no importance as the axis for the azimuth rotation is unchanged.

```
CS_azim coor_sys
(
    origin : struct(x: 0.0 m, y: 2.0 m, z: 0.0 m),
    base : ref(CS_elev)
)
```

The antenna to be rotated is defined in CS_azim.

In the *Movement Definition* object we define CS_base as the system in which the movements are specified:

```
coor_sys_for_movements: ref(CS_base)
```

and as all moving parts are defined in CS_elev we define this to be the coordinate system to move:

```
moved_coordinate_systems: sequence(ref(CS_elev))
```

Scan in azimuth and step in elevation

First we will consider antenna measurement carried out as scans in azimuth and steps in elevation as illustrated in Figure 2.

The slowest movement shall be specified first. This is the elevation rotation in which we step. The elevation rotation is around the original x_{el} -axis. The second rotation is the azimuth rotation which is around the new (rotated) y_{el} -axis.

The *movements* can now be specified. The applied step and scan values are those applied in the figure: step values 0° and 20° (elevation) and scan values -25° , -12.5° , 0° , 12.5° and 25° (in azimuth):

```
movements: sequence(
    struct(
        movement: rotation\around,
        axis: x\_original,
        start: 0., end: 20., number: 2
    ), ...

    struct(
        movement: rotation\around,
        axis: y\_new,
        start: -25., end: 25., number: 5
    ), ...
)
```

Note that there is no reference to CS_azim as the azimuth rotation is carried out as a second rotation of CS_elev. However, CS_azim is applied in order to plot the figures.

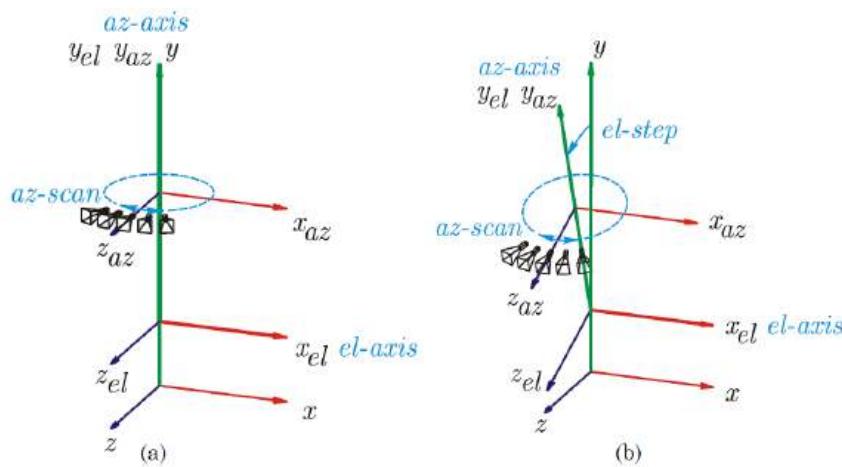


Figure 2

Scan in azimuth and step in elevation in an azimuth over elevation set-up. The scan in azimuth (*az-scan*) is illustrated by five positions of a horn antenna during the scan.(a) azimuth scan at the first step-value in elevation (0°). The antenna rotates around the azimuth axis (*az-axis*). (b) azimuth scan at the next step in elevation (*el-step*= 20°). Here the azimuth axis (*az-axis*) is tilted by the step in elevation.

Scan in elevation and step in azimuth

Next, we will consider scan in elevation followed by step in azimuth as illustrated in Figure 3.

The applied coordinate systems are the same as for the case above but the order of movements are changed (and the second step angle is increased to 40°):

```

movements: sequence(
    struct {
        movement: rotation\around,
        axis: y\_new,
        start: 0., end: 40., number: 2
    ), ...

    struct {
        movement: rotation\around,
        axis: x\_original,
        start: -25., end: 25., number: 5,
    ), ...
)
  
```

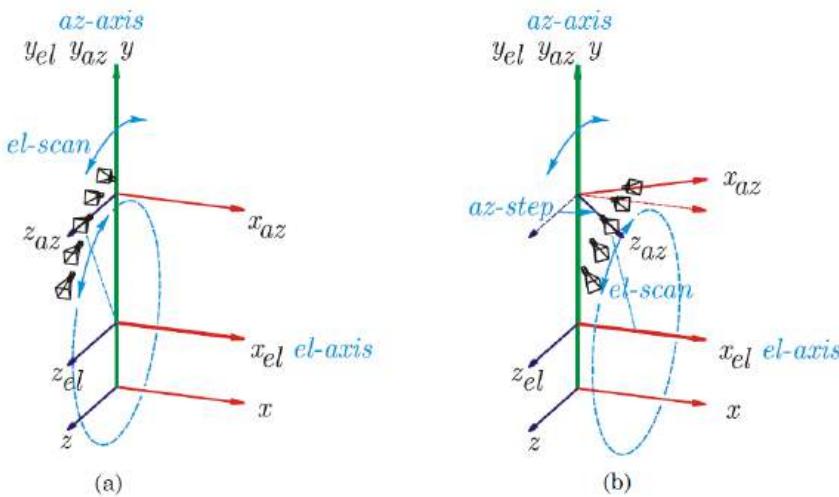


Figure 3

Scan in elevation and step in azimuth in the azimuth over elevation set-up. The scan in elevation (*el-scan*) is illustrated by five positions of a horn antenna during the scan.(a) elevation scan at the first step-value in azimuth (0°). The antenna rotates around the elevation axis (*el-axis*). (b) elevation scan at the next step in azimuth ($\text{az-step}=40^\circ$). Here the elevation axis (*el-axis*) is not influenced by the step in azimuth but the azimuth axis again follows the rotation around the elevation axis.

Elevation over azimuth set-up

Next, we will consider the elevation over azimuth set-up. Here, the elevation positioner is mounted upon the azimuth positioner and a rotation in azimuth will therefore rotate the elevation axis while a rotation in elevation has no influence on the azimuth axis.

We will illustrate this in the following figures. Again, a fixed *xyz*-coordinate system is placed on the ground at the measurement site. For illustration purposes, the azimuth positioner is mounted at some height (1.0 m) above the origin of the *xyz*-system. The axes which move in azimuth is denoted x_{az} , y_{az} and z_{az} . The elevation positioner is mounted with the elevation axis x_{el} identical to the axis x_{az} . The axes which are moved in the elevation rotation are denoted x_{el} , y_{el} and z_{el} . For illustration of the rotations an antenna horn is mounted at $(x_{el}, y_{el}, z_{el}) = (0.0 \text{ m}, 2.0 \text{ m}, 1.5 \text{ m})$.

In order to specify this in GRASP we define a set of coordinate systems. The global coordinate system is fixed at the ground of the measurement site, the *xyz*-system:

```
CS_ground coor_sys ( )
```

One meter above this we have the base in which the movements are defined, CS_base. By default, the axes are parallel to those of CS_ground:

```
CS_base coor_sys
(
    origin : struct(x: 0.0 m, y: 1.0 m, z: 0.0 m),
```

```

    base : ref(CS_ground)
)

```

Upon the base (CS_base) we now define the azimuth rotation. The coordinate system (CS_az) has same origin as the base and is not yet rotated:

```

CS_az coor_sys
(
    base : ref(CS_base)
)

```

Finally, upon the azimuth positioner the elevation positioner is placed. Again, the coordinate system (CS_el) has same origin as the base and is not yet rotated:

```

CS_el coor_sys
(
    base : ref(CS_el)
)

```

The antenna to be rotated is now defined in CS_el.

In the *Movement Definition* object we define CS_base as the system in which the movements are specified:

```
coor_sys_for_movements: ref(CS_base)
```

and as all moving parts are defined in CS_az we define this to be the coordinate system to move:

```
moved_coordinate_systems: sequence (ref(CS_az))
```

Scan in azimuth and step in elevation

Measurements carried out as scan in azimuth and step in elevation will be carried out as scans around the azimuth axes with stepwise rotation around the elevation axis.

The azimuth rotation is around the original y_{az} -axis and the elevation rotation is around the new x_{el} -axis. It is the scan motion which is the fastest motion and it is therefore defined last.

The *movements* can now be specified. The applied step and scan values are those applied in the figure:

```

movements: sequence (
    struct (
        movement: rotation\around,
        axis: x\_new,
        start: 0., end: 20., number: 2
    ), ...
    struct (

```

```

movement: rotation\around,
axis: y\original,
start: -25., end: 25., number: 5
), ...
)

```

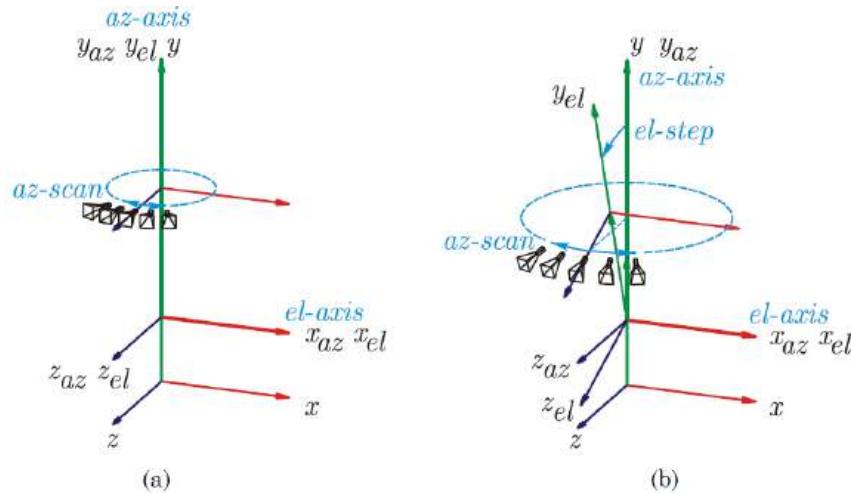


Figure 4

Scan in azimuth and step in elevation in an elevation over azimuth set-up. The scan in azimuth (*az-scan*) is illustrated by five positions of a horn antenna during the scan.(a) azimuth scan at the first step-value in elevation (0°). The antenna rotates around the azimuth axis (*az-axis*). (b) azimuth scan at the next step in elevation ($el-step=20^\circ$). The azimuth axis (*az-axis*) is not influenced by the step in elevation but the elevation axis and the tilted elevation coordinate system follows the rotation around the azimuth axis.

Scan in elevation and step in azimuth

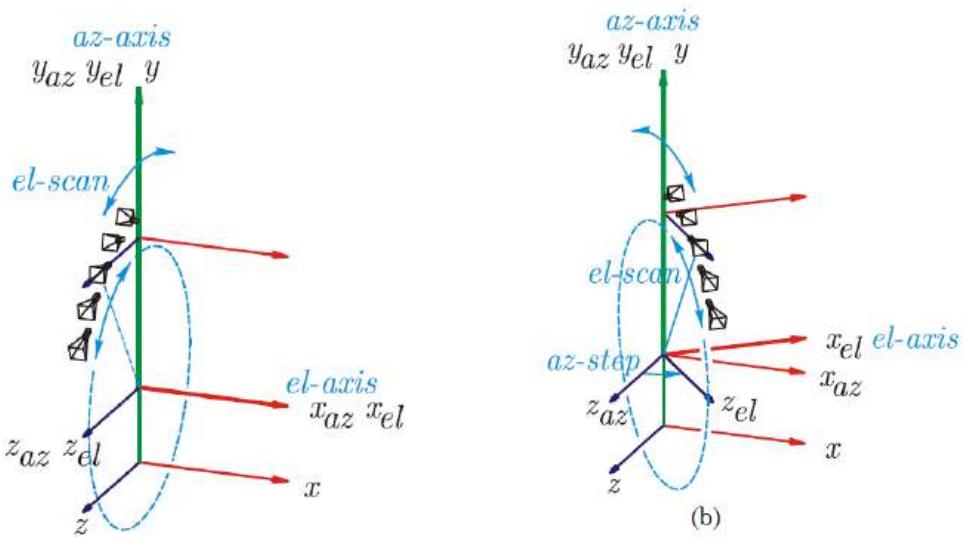
Next, we will consider step in azimuth followed by scan in elevation as illustrated in Figure 5.

The applied coordinate systems are the same as for the case above but the axes for the movements are interchanged:

```

movements: sequence (
struct (
    movement: rotation_around,
    axis: y_original,
    start: 0., end: 40., number: 2
), ...
)
struct (
    movement: rotation_around,
    axis: x_new,
    start: -25., end: 25., number: 5,
)

```



(a) elevation scan at the first step-value in azimuth (0°). The antenna rotates around the elevation axis (*el-axis*).

(b) elevation scan at the next step in azimuth ($\text{az-step}=40^\circ$). Here the elevation axis (*el-axis*) is rotated by the step in azimuth.

Figure 5

Scan in elevation and step in azimuth in the elevation over azimuth set-up. The scan in elevation (*el-scan*) is illustrated by five positions of a horn antenna during the scan.

) , ...
)

ELECTRICAL OBJECTS

Purpose

Classes for creating electrically related objects are collected in the menu *Electrical Objects*. The following items are available:

Frequency(ies) or wavelength(s) are specified in one of the classes in the menu

Frequency.

Simple antennas and pattern descriptions of sources, typically applied as stand-alone antennas or as feed in a reflector antenna, are found in the menu

Feed.

Currents are induced on a scatterer when it is illuminated by a source. These currents are also considered as sources as they radiate the scattered field. For most geometries, the induced currents can be determined by using a Physical Optics calculation which is defined under the menu:

PO Analysis.

Scattering determined as reflections and diffractions by the general Geometrical Theory of Diffractions are defined under

GTD Analysis.

Scattering from struts determined by special analysis routines is defined under

Strut Analysis.

Other sources are defined under

Other Sources.

The field values are determined at points defined by

Field Storage.

The scatterer may possess

Electrical Properties.

Finally, reflector data may be tabulated using

Derived Electrical Data.

MoM add-on: Method of Moments (MoM) calculations are defined under

MoM Analysis.

Links

Classes→*Electrical Objects*

FREQUENCY

Purpose

The frequency (or frequencies) at which the calculations are performed may be specified in the following ways:

One or more arbitrary frequencies may be given as a list in class

Frequency List

Equidistant frequencies within a range may be given in class

Frequency Range

Alternatively, instead of specifying the frequencies, a list of wavelengths may be given applying the class

Wavelength List

or a range with equidistantly spaced wavelengths may be specified using the class

Wavelength Range

Links

Classes→*Electrical Objects*→*Frequency*

FREQUENCY LIST (frequency)

Purpose

The class *Frequency List* defines one or more frequencies at which calculations are carried out.

Links

Classes→*Electrical Objects*→*Frequency*→*Frequency List*

Remarks

Syntax

```
<object name> frequency
(
    frequency_list : sequence(<rf>, ...)
)
where
<rf> = real number with unit of frequency
```

Attributes

Frequency List (*frequency_list*) [sequence of real numbers with unit of frequency].

A list of one or more frequencies each given with a valid unit of frequency (Hz, kHz, MHz, GHz, THz).

Remarks

The same calculations may be carried out by specifying the corresponding wavelengths in a *Wavelength List* object. When a *Frequency List* is to be referenced in other objects, a *Wavelength List* may equally well be referenced.

FREQUENCY RANGE (frequency_range)

Purpose

The class *Frequency Range* defines a set of equally spaced frequencies at which calculations are carried out.

Links

Classes→*Electrical Objects*→*Frequency*→*Frequency Range*

Syntax

```
<object name> frequency_range
(
    frequency_range : struct(start_frequency:<rf>,
                           end_frequency:<rf>,
                           number_of_frequencies:<i>)
)
where
<i> = integer
<rf> = real number with unit of frequency
```

Attributes

Frequency Range (*frequency_range*) [struct].

Definition of a list of equispaced frequencies.

Start Frequency (*start_frequency*) [real number with unit of frequency].

Start value in the list of frequencies with a valid frequency unit (Hz, kHz, MHz, GHz, THz).

End Frequency (*end_frequency*) [real number with unit of frequency].

End value in the list of frequencies with a valid frequency unit (Hz, kHz, MHz, GHz, THz).

Number of Frequencies (*number_of_frequencies*) [integer], default: **0**.

Number of frequency values in the list.

WAVELENGTH LIST (wavelength)

Purpose

The class *Wavelength List* defines one or more wavelengths at which calculations are carried out.

Links

Classes→*Electrical Objects*→*Frequency*→*Wavelength List*

Remarks

Syntax

```
<object name> wavelength
(
    wavelength_list : sequence(<rl>, ...)
)
where
<rl> = real number with unit of length
```

Attributes

Wavelength List (*wavelength_list*) [sequence of real numbers with unit of length].

A list of one or more wavelengths each given with one of the valid units of length (mm, cm, m, km, in, ft).

Remarks

The same calculations may be carried out by specifying the corresponding frequencies in a *Frequency List* object. When a *Wavelength List* is to be referenced in other objects, a *Frequency List* may equally well be referenced.

WAVELENGTH RANGE (*wavelength_range*)

Purpose

The class ***Wavelength Range*** defines a set of equally spaced wavelengths at which calculations are carried out.

Links

Classes→*Electrical Objects*→*Frequency*→*Wavelength Range*

Syntax

```
<object name> wavelength_range
(
    wavelength_range : struct(start_wavelength:<rl>,
                               end_wavelength:<rl>,
                               number_of_wavelengths:<i>)
)
where
<i> = integer
<rl> = real number with unit of length
```

Attributes

Wavelength Range (*wavelength_range*) [struct].

Definition of a list of equispaced wavelengths.

Start Wavelength (*start_wavelength*) [real number with unit of length].

Start value in the list of wavelengths with a valid wavelength unit (Hz, kHz, MHz, GHz, THz).

End Wavelength (*end_wavelength*) [real number with unit of length].

End value in the list of wavelengths with a valid wavelength unit (Hz, kHz, MHz, GHz, THz).

Number of Wavelengths (*number_of_wavelengths*) [integer], default: **0**.

Number of wavelength values in the list.

SOURCE

Purpose

Entities from which a radiated field can be calculated are defined as a *Source*.

General sources, and simple antennas and pattern descriptions, typically applied as feed in a reflector antenna, are found under

Feed

The commonly applied PO method is denoted

PO, Single-Face Scatterer

while all other sources, including arrays, advanced PO methods GTD and MoM, are found under

Other Sources

Links

Classes→*Electrical Objects*→*Source*

FEED

Purpose

Simple and general sources as well as simple pattern descriptions are gathered in the menu *Feed*. The menu name refers to the fact that simple sources often are applied as a feed or as an element of a feed array illuminating a reflector antenna. With the *Tabulated Feed* it is possible to input any pattern (computed or measured) of an antenna and its surrounding structure.

The *Feed* models are collected in the following groups:

Pattern

Horn

Tabulated Feed

Planar Feed

Helix or Dipole

Remarks

To obtain a correct field determination at all distances the program expands the feed models into spherical modes and determines the field on basis of these modes. The exceptions from this rule are the Gaussian beam feeds, *Gaussian Beam*, *Pattern Def.* and *Gaussian Beam, Near-Field Def.* (which includes a precise near-field model), and the spherical wave expansion feed, *Tabulated SWE Coefficients* (which already is expressed in spherical modes).

This has importance, for example, during the design of a reflector antenna system where the user may apply feed elements which are so large that the reflector is in the near field from the feed. This has to be taken into account in order to obtain a correct prediction of the radiation performance. The imbedded spherical wave expansion gives automatically a correct near-field calculation of the illumination from any feed and the user need not to apply special near-field features.

The spherical wave expansion is controlled by the attribute *far_forced* of the relevant feed. This attribute is default 'off' giving an expansion of the far field in the spherical wave *Q*-coefficients.

The power of the feeds is not normalised using the spherical wave coefficients, since the pattern may be scaled in its definition and since the gain may change if the expansion is incomplete. In this way, the power can be used as an indication of a complete expansion with all necessary modes.

The near field can only be calculated outside the sphere of radius r ,

$$r = \frac{N}{k} \quad (1)$$

where k is the wavenumber, $k = \frac{2\pi}{\lambda}$ and N is the maximum number of polar modes (in n).

If the field is required inside this sphere, the modes having n larger than kr are skipped in the calculation (causing field discontinuities at $r = \frac{n}{k}$).

See class *Spherical Wave Expansion (SWE)* for further details.

Links

Classes→*Electrical Objects*→*Feed*

PATTERN

Purpose

This menu groups *Feed*s which are described by their *Pattern*.

The Gaussian feed radiating a Gaussian beam may be defined in two different ways. The Gaussian beam may be defined by its far-field as a

Gaussian Beam, Pattern Def.

The Gaussian beam may be defined by its near-field characteristics as applied in quasi optics as a

Gaussian Beam, Near-Field Def.

Simple patterns given by an E- and H-plane taper may be defined by

Elliptical Pattern

Simple Tapered Pattern (m=1)

and a rotational symmetric pattern with a cross polar back radiation may be defined by

Cardioid Pattern

Links

Classes→*Electrical Objects*→*Feed*→*Pattern*

GAUSSIAN BEAM, PATTERN DEF. (gaussian_beam_pattern)

Purpose

The class *Gaussian Beam, Pattern Def.* defines a feed, which radiates a Gaussian beam. A Gaussian beam provides a field with a Gaussian taper, and it satisfies Maxwell's equations in both near and far field. The feed model is particularly useful as a simple model for the radiation from a corrugated horn.

The class defines the Gaussian beam by its far-field pattern. The Gaussian beam may also be modelled from its near-field characteristics, cf. class *Gaussian Beam, Near-Field Def.*.

Links

Classes→*Electrical Objects*→*Feed*→*Pattern*→*Gaussian Beam, Pattern Def.*

Remarks

Syntax

```
<object name> gaussian_beam_pattern
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    taper_angle         : <r>,
    taper               : <r>,
    polarisation        : <si>,
    far_forced          : <si>,
    factor              : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Taper Angle (*taper_angle*) [real number].

Angle from boresight (in degrees) at which the field Taper is specified in the far field.

Taper (*taper*) [real number].

Field level in dB relative to boresight at the Taper Angle defined above. Usually, the value of Taper will be negative, in accordance with the intended use of *Gaussian Beam, Pattern Def.* as a simple model for the radiation from a corrugated horn. However, if Taper is positive a pattern with an inverse taper is realized.

Polarisation (*polarisation*) [item from a list of character strings], default: **lin-ear_x**.

Definition of the polarisation of the beam.

linear_x

Linear x_f -polarisation according to Ludwig's 3rd definition.

linear_y

Linear y_f -polarisation according to Ludwig's 3rd definition.

rhc

Right-hand circular polarisation.

lhc

Left-hand circular polarisation.

Far-Field Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (see also the Remarks section):

off

As the near field inherent in the model.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor:

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index for Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute Frequency above a sequence of frequencies (wavelengths) is listed. The Frequency Index for Plot points to the frequency (wavelength) number in this sequence.

Remarks

The *Gaussian Beam, Pattern Def.* provides a beam with a Gaussian taper. The beam is radiating along $\theta = 0^\circ$ in the feed coordinate system specified by Coordinate System (the x_f, y_f, z_f -coordinate system).

The beam is broader - measured in degrees - in the near field than in the far field. It is a closed form model satisfying Maxwell's equations in both near and far field. Because of the taper, the behaviour of the near field is more complicated than that of a point source, so that the phase fronts are not simply spherical, but rather ellipsoidal. The phase centre for the far field is at the waist of the Gaussian beam which is at the origin of the feed coordinate system.

The feed pattern is normalised to dBi, i.e. to radiate 4π watts. The total radiated power will be 4π times the value of Factor (converted from dB to power).

The pattern is not expanded into spherical waves as the model includes a precise near-field description.

GAUSSIAN BEAM, NEAR-FIELD DEF. (gaussian_beam)

Purpose

The class *Gaussian Beam, Near-Field Def.* defines a feed, which radiates a Gaussian beam, by the near-field parameters of the Gaussian beam. The class is similar to *Gaussian Beam, Pattern Def.*, except that the input is more oriented towards users familiar with the terminology used in quasi-optics.

The Gaussian beam is modelled from its far-field pattern in class *Gaussian Beam, Pattern Def.*

A Gaussian beam provides a field with a Gaussian taper, and it satisfies Maxwell's equations in both the near and the far field. The feed model is particularly useful as a simple model for the radiation from a corrugated horn.

Links

Classes→*Electrical Objects*→*Feed*→*Pattern*→*Gaussian Beam, Near-Field Def.*

Remarks

Syntax

```
<object name> gaussian_beam
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    beam_radius         : <rl>,
    phase_front_radius : <rl>,
    polarisation        : <si>,
    factor              : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Beam Radius (*beam_radius*) [real number with unit of length].

Distance from beam axis at which the field is decreased to $1/e$ of the on-axis field.

Phase Front Radius (*phase_front_radius*) [real number with unit of length].

Radius of curvature of the beam phase-front surface on the axis. If the source is located in the waist of the Gaussian beam the phase-front radius of curvature becomes infinite. This special case is specified by setting *phase_front_radius* equal to zero. A point source (Huygens source) is obtained if also *beam_radius* is equal to zero.

Polarisation (*polarisation*) [item from a list of character strings], default: **linear_x**.

Definition of the polarisation of the beam.

linear_x

Linear x_f -polarisation according to Ludwig's 3rd definition.

linear_y

Linear y_f -polarisation according to Ludwig's 3rd definition.

rhc

Right-hand circular polarisation.

lhc

Left-hand circular polarisation.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor:

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index For Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute *frequency* above a sequence of frequencies (wavelengths) is listed. The *frequency_index_for_plot* points to the frequency (wavelength) number in this sequence.

Remarks

The *Gaussian Beam, Near-Field Def.* provides a feed radiating a beam with a Gaussian taper. The beam is radiating along $\theta = 0^\circ$ in the feed coordinate system specified by *coor_sys* (the x_f, y_f, z_f -coordinate system). It is a closed form model satisfying Maxwell's equations in both near and far fields. Because of the taper, the behaviour of the near field is more complicated than

that of a point source, so that the phase fronts are not simply spherical, but rather ellipsoidal. The phase centre for the far field is at the waist of the Gaussian beam.

The *beam_radius* w and the *phase_front_radius* R_c are related to the beam radius w_0 at the waist and the position of the waist at $z_f = -z_0$ by

$$w = w_0 \sqrt{1 + z_0^2/b^2} \quad (1)$$

$$R_c = z_0(1 + b^2/z_0^2) \quad (2)$$

where $b = w_0^2 k / 2$ is the confocal distance and $k = 2\pi/\lambda$ is the wavenumber.

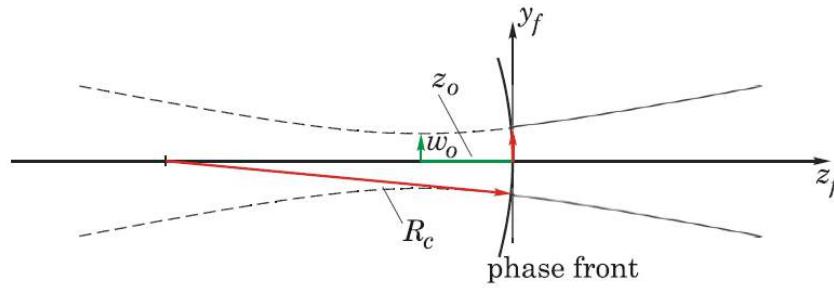


Figure 1

Definition of a Gaussian beam by its *beam_radius* w and *phase_front_radius* R_c given at the origin of the feed coordinate system. The Gaussian beam field has decreased to $1/e$ of the on-axis field along the drawn hyperbolas. The field radiates in the direction of positive z_f .

z_0 is the distance from the waist to the origin of the feed coordinate system and has the same sign as R_c .

The above equations can be solved for w_0 and z_0

$$w_0 = \frac{w}{\sqrt{1 + \left(\frac{kw^2}{2R_c}\right)^2}} \quad (3)$$

$$z_0 = \frac{R_c}{1 + \left(\frac{2R_c}{kw^2}\right)^2} \quad (4)$$

The feed pattern is normalised to dBi, i.e. to radiate 4π watts. The total radiated power will be 4π times the value of *factor* (converted from dB to power).

The pattern is not expanded into spherical waves as the model includes a precise near-field description.

Examples

In the following, two examples on Gaussian beam feeds are given.

By *Feed Plot* the feed is symbolically drawn as a horn with apex at the phase centre for the specified phase front. When the *phase_front_radius*, R_c , is positive this phase centre is behind the aperture as shown in Figure 2 and Figure 3. The drawn horn is truncated to a length of 20λ if $R_c > 20\lambda$.

The horn is further given an aperture radius of 1.6 times the *beam_radius* w . The aperture of the horn is drawn in the $x_f y_f$ -plane of the feed coordinate system.

In the aperture of the horn a spherical phase front of curvature R_c and diameter w is drawn. Rays from the feed may be drawn by *Rays from Point Sources*. Such rays will be directed from the waist position of the Gaussian beam which is the phase centre for the far field.

When the *phase_front_radius* R_c is specified negative, the phase centre for the specified phase front is in front of the aperture as in the example of Figure 4 and Figure 5. The drawn feed ‘horn’ then appears as a ring extending 20λ behind the aperture. Continuations of the lines of the ring intersect on the feed axis at the centre of curvature for the phase front drawn in the aperture.

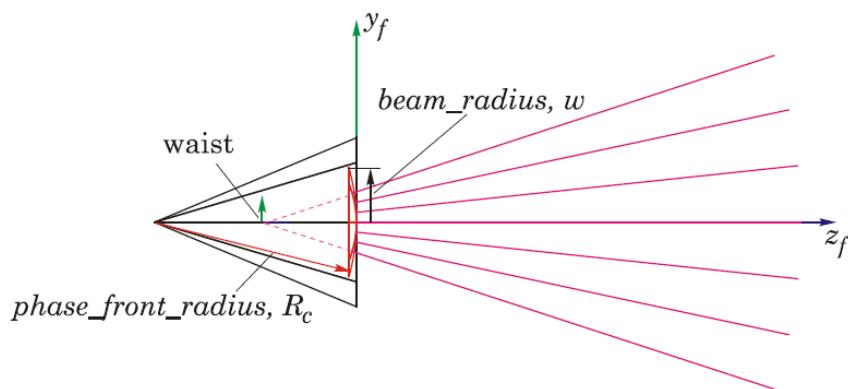


Figure 2

Side view

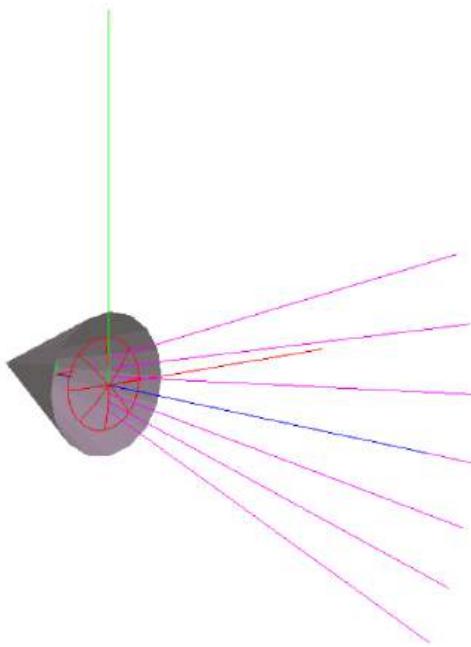


Figure 3

OpenGL 3D view - as in Figure 2, Gaussian beam feed with positive *phase_front_radius*. In the example the *phase_front_radius* is specified to $R_c = 4.17\lambda$ and the *beam_radius* is specified to $w = 1.11\lambda$. This results in the waist is at $z_f = -z_0 = -1.93\lambda$. The waist is here marked by a coordinate system with short axes. When the *phase_front_radius* is positive the horn is drawn in grey.

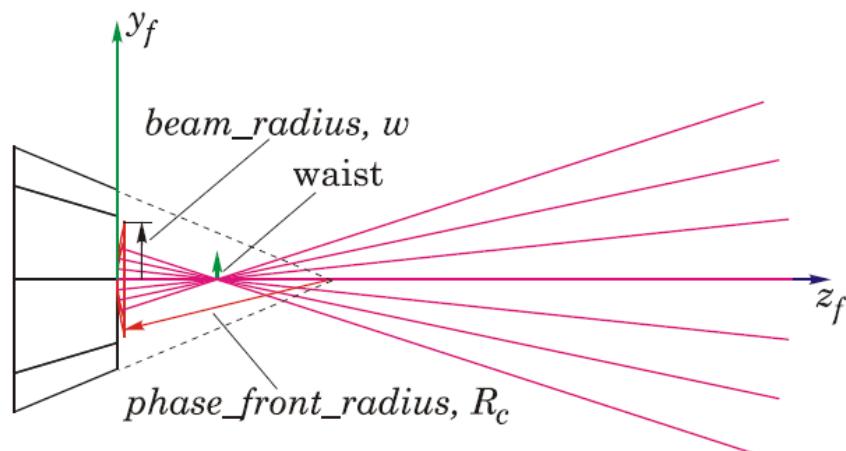


Figure 4

Side view

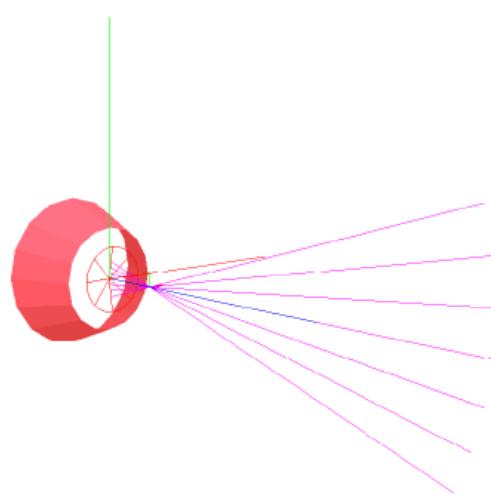


Figure 5

OpenGL 3D view. As in Figure 4, Gaussian beam feed with negative *phase_front_radius*. In the example the *phase_front_radius* is now specified to $R_c = -4.17\lambda$ and the *beam_radius* is specified to $w = 1.11\lambda$ as in Figure 1. This results in the waist is $z_0 = 1.93\lambda$ in front of the aperture. The waist is here marked by a coordinate system with short axes. When the *phase_front_radius* is negative the horn is drawn in red.

ELLIPTICAL PATTERN (elliptical_pattern)

Purpose

The class *Elliptical Pattern* defines a feed with a radiation pattern with an elliptical main beam by means of simple exponential functions. The pattern has a zero in the back direction.

Links

Classes→*Electrical Objects*→*Feed*→*Pattern*→*Elliptical Pattern*

Remarks

Syntax

```
<object name> elliptical_pattern
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    taper               : <r>,
    taper_angles        : struct(zx:<r>, zy:<r>),
    polarisation        : <si>,
    polarisation_angle : <r>,
    far_forced          : <si>,
    factor              : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Taper (*taper*) [real number].

Field level in dB relative to boresight at the *taper_angles* defined below. For a normal beam the value of *taper* is negative.

Taper Angles (*taper_angles*) [struct].

Angles from boresight at which the field *taper* defined above is specified. The *taper_angles* are given in the two major planes, which are the z_f, x_f and the z_f, y_f -plane of the feed coordinate system.

Zx (zx) [real number].

Taper angle, θ_{zx} , in degrees, in the z_f, x_f -plane (i.e. $\phi = 0^\circ$)

Zy (zy) [real number].

Taper angle, θ_{zy} , in degrees, in the z_f, y_f -plane (i.e. $\phi = 0^\circ$)

Polarisation (polarisation) [item from a list of character strings], default: **linear**.

Definition of the polarisation of the beam, which can be one of several options.

linear

Linear polarisation according to Ludwig's 3rd definition.

rhc

Right-hand circular polarisation.

lhc

Left-hand circular polarisation.

Polarisation Angle (polarisation_angle) [real number], default: **0**.

Angle of the on-axis linear polarisation measured from the x_f -axis towards the y_f -axis (see also the remarks below).

Far Forced (far_forced) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on *Near-Field Calculations*).

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (factor) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (db) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (deg) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index For Plot (frequency_index_for_plot) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute *frequency* above a sequence of frequencies (wavelengths) is listed. The *frequency_index_for_plot* points to the number to be applied of the frequency (wavelength) in this sequence.

Remarks

The feed model is based on the following mathematical expression:

$$A(\theta, \phi) = A_0(1 + \cos \theta)e^{-(1-\cos \theta)(\alpha \cos^2 \phi + \beta \sin^2 \phi)}$$

where the constants α and β are determined to fulfil the beam width specifications as given by *taper* and *taper_angles* (see the GRASP Technical Description for the details). A_0 normalises the pattern to dBi, i.e. to radiate 4π watts. The total radiated power will be 4π times the value of *factor* (converted from dB to power).

An example of a highly elliptical beam is shown in Figure 1.

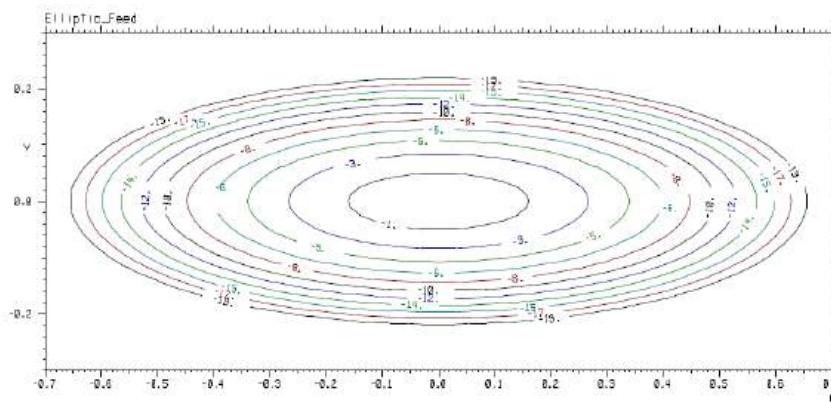


Figure 1

An elliptical beam defined by *Elliptical Pattern*. Contours are relative to peak, drawn in *uv-grid*. The feed is defined by a *taper* of -3 dB at *taper_angles* 16° in the $z_f x_f$ -plane and 5° in the $z_f y_f$ -plane.

The directivity of the pattern is normalised by pattern integration as follows:

For $taper < 0$:

The integration is carried out to a pattern level 100 dB below peak. This integrated power is equated to 4π , and the pattern is then with high precision normalised to dBi.

For $taper > 0$:

An inverse taper ($taper > 0$) or a uniform illumination ($taper = 0$) may be useful in certain special cases. Then the integration is carried out up to the average taper angle, i.e. the power radiated within $|\theta| \leq (\theta_{zx} + \theta_{zy})/2$ is equal to 4π , which means that the pattern will not be normalised to dBi, since the feed radiation continues outside *taper angles*.

Since the pattern will always be zero at $\theta = 180^\circ$, a uniform radiation cannot be obtained. If $taper = 0$, the field at $\theta = 0^\circ$ will be equal to the field at the two *taper_angles* in the respective planes. The maximum of the pattern will in this case not be located at $\theta = 0^\circ$ but somewhere inside the

range given by the *taper_angles*. This is only important for large values of the *taper_angles* ($> 90^\circ$). Drawing a few cuts of the feed pattern is recommended for such extreme cases.

For the *polarisation_angle* zero, the field is linearly *x*-polarised according to Ludwig's 3rd definition. When the *polarisation_angle* is non-zero, the polarisation is defined correspondingly with respect to an *x*-axis, which is rotated this *polarisation_angle* from the x_f -axis towards the y_f -axis around the common *z*- and z_f -axis. The x_f y_f z_f -axes are those of the feed coordinate system.

When *far-forced* is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

The number of m-modes depends on the ellipticity of the beam, ε_θ , and is found empirically to

$$M = 5 + 7\varepsilon_\theta$$

where ε_θ is given as

$$\varepsilon_\theta = \max\left(\frac{\theta_{zx}}{\theta_{zy}}, \frac{\theta_{zy}}{\theta_{zx}}\right)$$

The number of ϕ -cuts over 360° is then taken as

$$N_\phi = 2(M + 1)$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by the size of the feed

$$N_\theta - 1 = N = kr_0 + \max(10, 3.6\sqrt[3]{kr_0})$$

where k is the wavenumber and r_0 is the radius of the equivalent aperture. For a normal beam with negative taper, A_{dB} in dB, the maximum aperture radius is determined from

$$\frac{r_0}{\lambda} = 0.184 \frac{\sqrt{-A_{dB}}}{\sin \theta_t}$$

where θ_t is the minimum defined taper angle (smallest of θ_{zx} and θ_{zy}).

When A_{dB} is specified positive N is defined as

$$N = 13$$

For a given N the field may be determined for distances larger than r_{swe}

$$r_{swe} = \frac{N}{k} \tag{1}$$

Example

An example on the mode expansion is given below for an *Elliptical Pattern* with a taper of -10dB at an angle of 10° in $z_f x_f$ -plane and 30° in $z_f y_f$ -plane. The feed has linear *polarisation* with *polarisation_angle* 0, i.e. polarised

along the x_f -axis. Both the far field and the near field at a radius of 10λ will be considered.

For this case (rounded up)

$$M = 27$$

and

$$N_\theta - 1 = N = 33 \quad (2)$$

It shall be checked that this maximum mode number does not represent sources outside the near-field distance $r_{nf} = 10\lambda$:

$$r_{swe} = 33/k \cong 5.3\lambda \leq r_{nf} \quad (3)$$

In Figure 2 the near field is shown, as the modelled far field scaled to the near-field distance (*far-forced* is 'on') and as expressed at the near-field distance by a spherical wave expansion of the far field (*far-forced* is 'off').

The feed model does not fulfil Maxwell's equations as it is based on a simple mathematical model. The spherical wave expansion can therefore not fit completely to the pattern and the expansion in spherical wave coefficients does not have vanishing mode coefficients for $N = 33$. We therefore find ripples in the patterns with a period of

$$\Delta\theta = \frac{180^\circ}{33} = 5.5^\circ \quad (4)$$

corresponding to the non-vanishing modes with $N = 33$.

When the spherical expansion can not reproduce the near field as seen above neither the far-field pattern can be reproduced. This is illustrated Figure 3.

Consider first the mathematical model which is obtained when *far-forced* is 'on' (dotted lines). The co-polar field is again shown for $\phi = 0^\circ, 45^\circ$ and 90° while the cross-polar field component vanishes.

When *far-forced* is 'off' a similar pattern is obtained in the main beam. But the mode truncation results in lobes as described for the near field with a period of 5.5° . Inspection of the tables with the mode power contents in the standard output file tells that the power content of modes with $n = 33$ is about 10^{-7} relative to the total power. 4π watts evenly radiated corresponds to 0 dBi and the maxima of the lobes therefore occur up to $4\pi \cdot 10^{-7}$ watts or approximately -60 dBi.

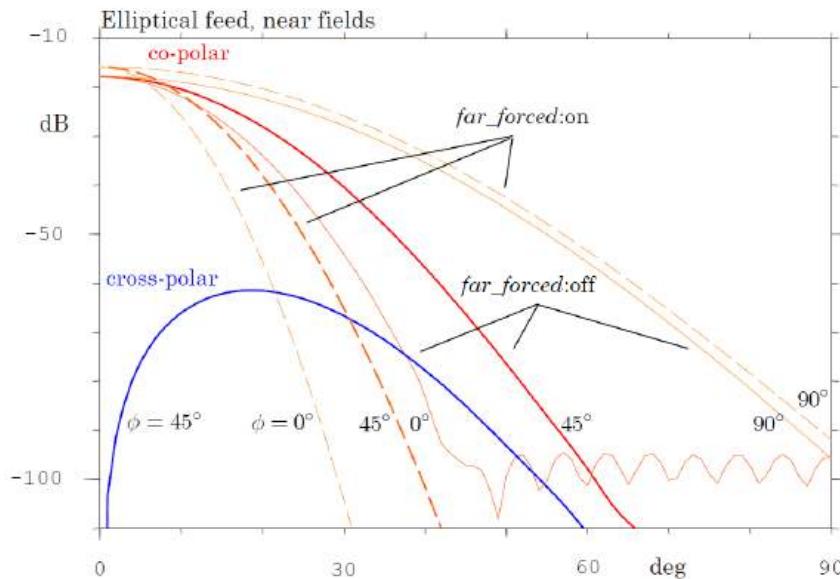


Figure 2

Near-field patterns for an *Elliptical Pattern* for $\phi = 0^\circ$ (narrow), 45° and 90° (wide). For patterns in full lines *far_forced* is specified to 'off' and for dashed patterns *far_forced* is specified to 'on'. The latter are proportional to the far-field patterns and the near-field effects are seen to result in a wider beam in the narrow $\phi = 0^\circ$ plane. Cross polarisation only appears for *far_forced* 'off' in the $\phi = 45^\circ$ plane.

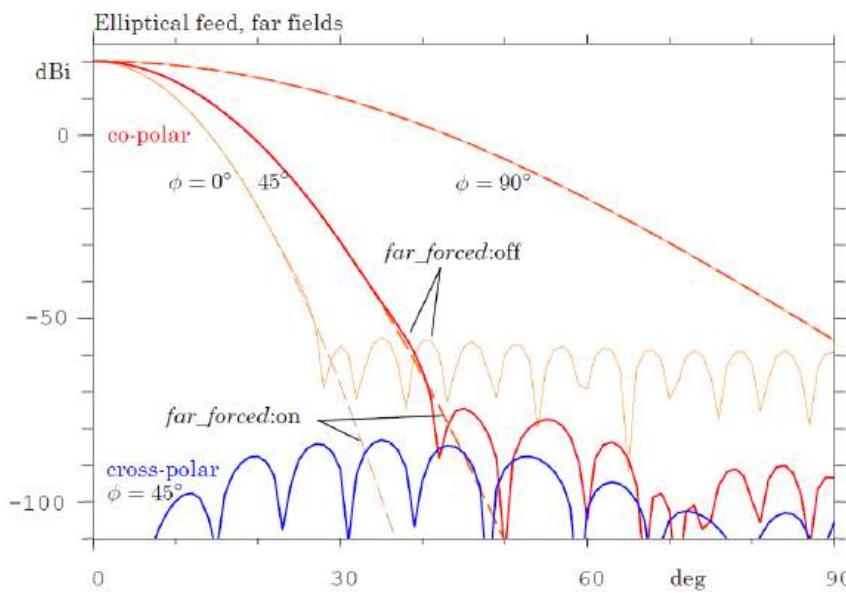


Figure 3

Far-field patterns for the *Elliptical Pattern* for $\phi = 0^\circ$ (narrow), 45° and 90° (wide). For patterns in full lines *far_forced* is specified to 'off' and for dashed patterns *far_forced* is specified to 'on'. The latter are the far-field patterns as given by its mathematical expression. Cross polarisation only appears for *far_forced* 'off' in the $\phi = 45^\circ$ plane.

SIMPLE TAPERED PATTERN (M=1) (simple_tapered_pattern)

Purpose

The class *Simple Tapered Pattern (m=1)* defines a feed with a radiation pattern by means of a simple exponential function, which provides a fairly good approximation to the radiation from a corrugated horn.

Links

Classes→*Electrical Objects*→*Feed*→*Pattern*→*Simple Tapered Pattern (m=1)*

Remarks

Syntax

```
<object name> simple_tapered_pattern
(
    frequency : ref(<n>),
    coor_sys   : ref(<n>),
    taper_angle : <r>,
    taper      : struct(zxdb:<r>, zydb:<r>),
    polarisation : <si>,
    cross_polar  : struct(db:<r>, deg:<r>),
    far_forced   : <si>,
    factor       : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Taper Angle (*taper_angle*) [real number].

Angle (in degrees) from boresight at which the field has decreased by the *taper* specified. The *taper_angle* is fixed, while the *taper* can be specified independently in the z_f, x_f - and z_f, y_f -planes in the feed coordinate system.

Taper (*taper*) [struct].

Field level in dB relative to boresight at the *taper_angle* defined above.
For a normal beam, the values of the *taper* are negative.

zx-dB (*zxdb*) [real number].

Taper (in dB) in the z_f, x_f -plane (i.e. $\phi = 0^\circ/180^\circ$)

zy-dB (*zydb*) [real number].

Taper (in dB) in the z_f, y_f -plane (i.e. $\phi = 90^\circ/270^\circ$)

Polarisation (*polarisation*) [item from a list of character strings], default: **lin-ear_x**.

Definition of the polarisation of the beam:

linear_x

Linear *x*-polarisation according to Ludwig's 3rd definition.

linear_y

Linear *y*-polarisation according to Ludwig's 3rd definition.

rhc

Right-hand circular polarisation.

lhc

Left-hand circular polarisation.

Cross Polar (*cross_polar*) [struct].

A cross-polar pattern identical to the co-polar pattern apart from a complex factor may be included in the field.

Amplitude in dB (*db*) [real number], default: **-600**.

The amplitude in dB of the cross-polar pattern relative to the co-polar pattern.

Phase in degrees (*deg*) [real number], default: **0**.

The phase (in degrees) of the cross-polar pattern relative to the co-polar pattern.

Far Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on [Near-Field Calculations](#)):

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (db) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (deg) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index For Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute *frequency* above a sequence of frequencies (wavelengths) is listed. The *frequency_index_for_plot* points to the number to be applied of the frequency (wavelength) in this sequence.

Remarks

The *Simple Tapered Pattern ($m=1$)* is generated from the specified *taper*-values in the principal planes. The pattern is a so-called $m = 1$ pattern, i.e. the pattern in the azimuthal (ϕ) direction is determined by a $\sin(m\phi)$ and $\cos(m\phi)$ variation with $m = 1$, ϕ being the usual azimuthal angle.

When a beam is specified to be narrow in one plane and wide in the perpendicular plane, the azimuthal variation causes the beam to be oval rather than elliptical, see the figure below. A beam with an elliptical cross-section should be modelled by the *Elliptical Pattern*, cf. the figure in that section. In the principal planes, the two definitions have similar patterns.

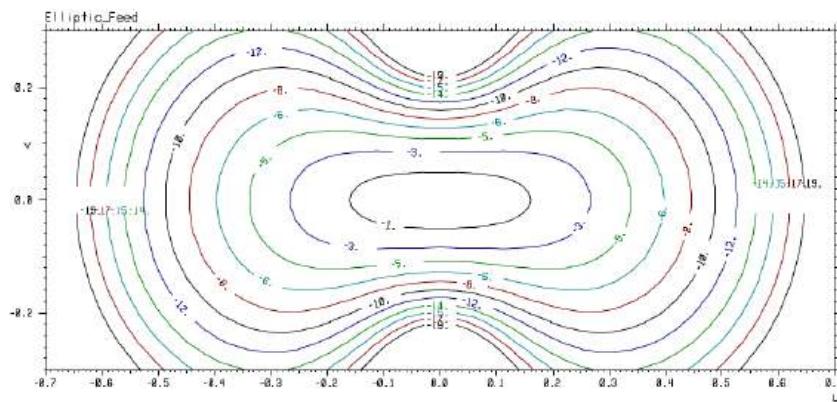


Figure 1

An ‘elliptical’ beam defined by the *Simple Tapered Pattern ($m=1$)*, contours are relative to peak in a uv-grid:
taper_angle: 16, *taper*: struct(*zxdB*: -3, *zydB*: -30.5) i.e. -3 dB resp. -30.5 dB at 16° in the principal planes.

The amplitude is finite in the back direction, $\theta = 180^\circ$, resulting in a discontinuity in the pattern polarisation. Furthermore, when different taper values (*zxdB*, *zydB*) are specified in the two principal planes, also the amplitude is discontinuous.

Depending on the specified *taper*, the directivity is normalised by pattern integration as follows:

When both *zxdB* and *zydB* < 0 The integration is carried out to the -40 dB level, the remaining power being ignored. This integrated power is equated

to 4π , i.e. the pattern is normalised to dBi.

When either $zxdB$ or $zydB \geq 0$

An inverse taper ($taper > 0$) or a uniform illumination ($taper = 0$) may be simulated. The integration is then carried out to the specified *taper_angle*; however, the feed radiation continues outside the *taper_angle*. The pattern is therefore not expressed in dBi.

The feed pattern is normalised to dBi, i.e. to radiate 4π watts. The total radiated power will be 4π times the value of *factor* (converted from dB to power).

When *far-forced* is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

As the feed is described by 4 ϕ -cuts over 360°

$$N_\phi = 4$$

the number of azimuthal modes is limited to

$$M = 1$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by

$$N_\theta - 1 = N = kr_o + \max(10, 3.6\sqrt[3]{kr_o})$$

where k is the wavenumber and r_o is the radius of an aperture radiating a similar beam, empirically determined as

$$r_o = 0.184 \frac{\sqrt{-A}}{\sin \theta_t} \lambda$$

A being the lowest (with sign) of $zxdB$ and $zydB$, and θ_t being the *taper_angle*. For

If $A > 0$ or $\theta_t \leq 0$ then

$$r_o = 0.5\lambda$$

is applied.

For a given N the field may be determined for distances larger than r_{swe}

$$r_{swe} = \frac{N}{k}$$

The applied values of N_ϕ , N_θ , M and N can be found in the standard output file.

CARDIOID PATTERN (cardioid_pattern)

Purpose

The class *Cardioid Pattern* defines a feed with a cardioid-shaped, circularly polarised low-gain antenna pattern. Such a pattern may be a good approximation to a satellite telemetry and telecommand antenna.

Links

Classes→*Electrical Objects*→*Feed*→*Pattern*→*Cardioid Pattern*

Remarks

Syntax

```
<object name> cardioid_pattern
(
    frequency                  : ref(<n>),
    coor_sys                   : ref(<n>),
    beamwidth_co               : <r>,
    beamwidth_cx               : <r>,
    front_to_back_ratio        : <r>,
    polarisation               : <si>,
    far_forced                 : <si>,
    factor                     : struct(db:<r>, deg:<r>),
    frequency_index_for_plot   : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Beamwidth Co (*beamwidth_co*) [real number].

Half-power beamwidth in degrees of the co-polar pattern (in forward direction).

Beamwidth Cx (*beamwidth_cx*) [real number].

Half-power beamwidth in degrees of the cross-polar pattern (in backward direction, a back lobe).

Front to Back Ratio (*front_to_back_ratio*) [real number].

Level of front radiation relative to back radiation, in dB.

Polarisation (*polarisation*) [item from a list of character strings], default: **rhc**.

Polarisation of the co-polar pattern.

rhc

Right-hand circular polarisation.

lhc

Left-hand circular polarisation.

Far Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on *Near-Field Calculations*).

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index For Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute *frequency* above a sequence of frequencies (wavelengths) is listed. The *frequency_index_for_plot* points to the number to be applied of the frequency (wavelength) in this sequence.

Remarks

The *Cardioid Pattern* approximates the radiation from a satellite telemetry and telecommand (TT&C) antenna. The pattern is constructed as a superposition of two patterns: a wide co-polar beam in the front hemisphere with a null in the backward direction, and a wide cross-polar beam in the rear hemisphere with a null in the forward direction.

Figure 1 shows a typical cardioid pattern. The radiation in the rear hemisphere around 180° has the opposite circular polarisation than the radiation in the front hemisphere around boresight, and its level is lower by the *front_to_back_ratio*. Also the half-beamwidths at half power (3 dB below the peaks) are illustrated.

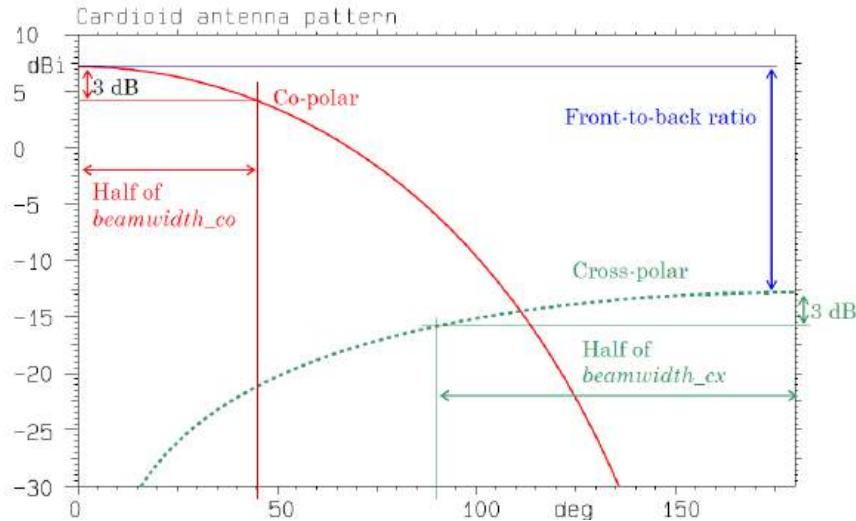


Figure 1 Example of a *Cardioid Pattern*.

The shape of the co- and cross-polar patterns are modelled by the functions

$$f_{front}(\theta) = K [(1 + \cos \theta)/2]^{n_{front}} \quad (1)$$

$$f_{back}(\theta) = K [(1 - \cos \theta)/2]^{n_{back}} \quad (2)$$

respectively. The exponents n_{front} and n_{back} are determined from the specified beamwidths, while K is a normalisation constant, which normalises the field to dBi.

The feed pattern is normalised to dBi, i.e. to radiate 4π watts. The total radiated power will be 4π times the value of *factor* (converted from dB to power).

When *far-forced* is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

Due to the rotational symmetry of the feed the number of azimuthal modes is limited to

$$M = 1$$

and the necessary number of ϕ -cuts over 360° is

$$N_\phi = 4$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by the size of the feed

$$N_\theta - 1 = N = kr_o + \max(10, 3.6 \sqrt[3]{kr_o})$$

where k is the wavenumber and r_o is the radius of an equivalent aperture given by

$$r_o = \frac{1}{4} \lambda / \sin(\theta_{3dB})$$

θ_{3dB} is the half of the smallest of *beamwidth_co* and *beamwidth_cx*.

The spherical wave expansion will be different from the pattern as given by Eqs. (1) and (2) above because these mathematical expressions do not satisfy Maxwell's equations. The mathematical expressions are applied when *far_forced* is specified to 'on' while the spherical wave expansion is applied when *far_forced* is specified to 'off'.

A note on the polarisation

The two beams are circularly polarised according to a polarisation definition which is continuous across the peak of the respective beam. This has the following consequences for the phases of the two beams.

Let the cardioid pattern be calculated in right and left hand circular components in the feed coordinate system which has the z -axis in the direction of the co-polar main beam of the cardioid. Then the co-polar beam will have an amplitude which varies according to Eq. (1) above and a phase which is constant in all directions. The cross-polar beam will have an amplitude which varies according to Eq. (2) above but the phase will vary 4π linearly in ϕ when ϕ runs from 0 to 2π .

On the other hand, if the cardioid pattern is calculated in circular components in a coordinate system which has the z -axis in the direction of the cross-polar back lobe then the cross-polar beam will have a phase which is constant in all directions and the co-polar beam will have a phase which varies 4π in ϕ when ϕ runs from 0 to 2π .

This is a consequence of the definition of circular polarisation (cf. the Technical Description).

HORN

Purpose

This menu groups *Feed*s which are given by a *Horn* model.

Conical Horn

Fundamental Mode Circular Waveguide

Potter Horn

Hybrid Mode Conical Horn

Rectangular Horn

Hexagonal Horn

Links

Classes→*Electrical Objects*→*Feed*→*Horn*

CONICAL HORN (conical_horn)

Purpose

In the class *Conical Horn* the radiation from a circular conical horn, or a circular open-ended waveguide is modelled. A quadratic phase variation in the aperture caused by a finite horn flare is included. The excitation may be a combination of circular waveguide modes.

Links

Classes→*Electrical Objects*→*Feed*→*Horn*→*Conical Horn*

Remarks

Syntax

```
<object name> conical_horn
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    aperture_radius     : <rl>,
    flare_length        : <rl>,
    phase_displacement  : <rl>,
    ground_plane        : <si>,
    modes               : sequence(
                                struct(type:<s>,
                                       amp:<r>,
                                       phase:<r>,
                                       rot:<r>),
                                ...),
    far_forced          : <si>,
    factor              : struct(db:<r>, deg:<r>),
    mode_power_normalization : <si>,
    frequency_index_for_plot : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located at the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Aperture Radius (*aperture_radius*) [real number with unit of length].

Radius of the circular aperture.

Flare Length (*flare_length*) [real number with unit of length], default: **0**.

The slant length of the flare, measured from the edge of the aperture to the horn apex, see the remarks below. If the flare length is less than the aperture radius, the flare is ignored.

Phase Displacement (*phase_displacement*) [real number with unit of length], default: **0**.

Displacement of the phase reference point along the horn zf -axis. A positive value of Phase Displacement moves the reference point along the positive zf -axis, and vice versa for negative displacements. For details of the phase reference, see the Section *Phase of a Radiating Horn*.

Ground Plane (*ground_plane*) [item from a list of character strings], default: **off**.

The horn aperture may either be placed in an infinite ground plane or radiate in free space.

off

The radiation is calculated as if the aperture is located in free space.

on

The radiation is calculated as if the aperture is located in an infinite ground plane (this is the best approximation if the feed is in a larger array).

Modes (*modes*) [sequence of structs].

Several circular waveguide modes may be present in the aperture.

Type (*type*) [character string], default: **TE11**.

Mode type selector. The *type* is a string of four characters, the first two of which must be either 'TE' or 'TM', while the third (m) and the fourth (n) are single-digit numbers within the ranges : $0 \leq m \leq 5, 1 \leq n \leq 5$.

Amplitude (*amp*) [real number], default: **1**.

Amplitude of the mode excitation, must be positive (amp^2 gives the power).

Phase (*phase*) [real number], default: **0**.

Phase of the mode excitation, in degrees.

Rotation (*rot*) [real number], default: **0**.

Rotation (in degrees) of the mode from the x_f -axis towards the y_f -axis. When *rot* = 0, TE_{11} is polarised along y_f and TM_{11} along x_f .

Far-Field Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on [Near-Field Calculations](#)):

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Mode Power Normalization (*mode_power_normalization*) [item from a list of character strings], default: **off**.

If mode power normalization is on, each mode is exactly normalized to 4pi by far field power integration. After normalization the individual excitations are used. The total power from the horn is still normalized to 4pi.

Frequency Index for Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute Frequency above a sequence of frequencies (wavelengths) is listed. The Frequency Index for Plot points to the number to be applied of the frequency (wavelength) in this sequence.

Remarks

A horn flare, defined through the attribute Flare Length, results in a spherical wave front centred at the apex. Accordingly, the phase variation in the horn aperture is quadratic. For details of the phase reference, see the Section [Phase of a Radiating Horn](#).

The feed is defined in the feed coordinate system specified by Coordinate System (the x_f, y_f, z_f -coordinate system).

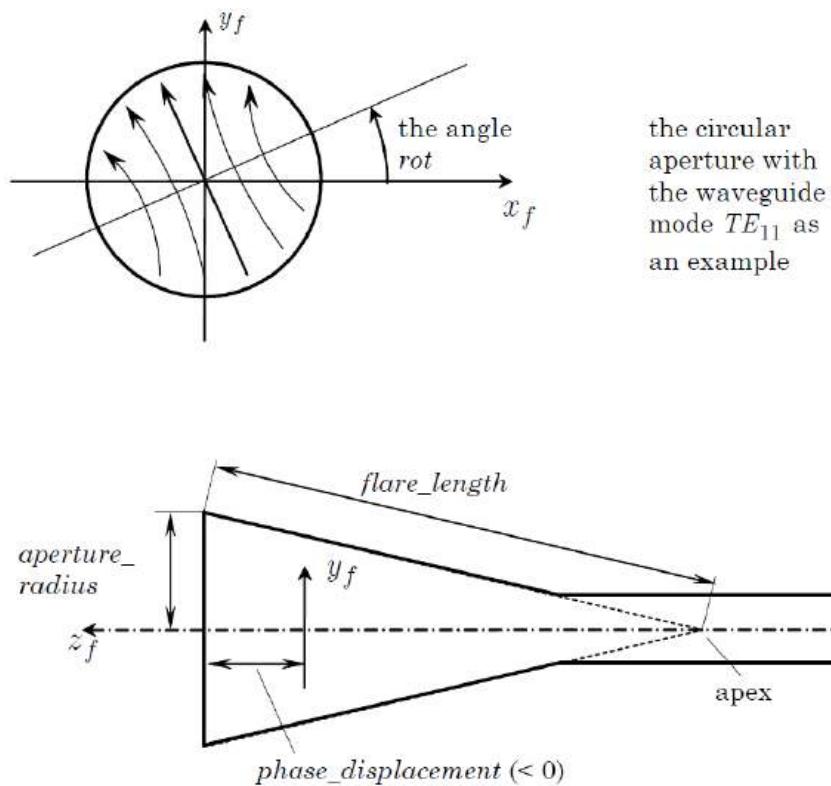


Figure 1 Conical horn with illustration of the Flare Length and the rotation rot (in attribute `Modes`) of a TE_{11} mode.

Modes below cut-off are not permitted. The total mode power is normalised to 4π watts so that the pattern is normalised to dBi. The total radiated power will be 4π times the value of `Factor` (converted from dB to power).

In the case the aperture is placed in a ground plane the field only exists for $\theta \leq 90^\circ$. As the spherical expansion requires the field on a full sphere, a mirrored source radiating within $90^\circ \leq \theta \leq 180^\circ$ is added resulting in a continuous field across the ground plane. The source itself is normalised to 4π watts and the spherical expansion including the mirrored source contains coefficients representing 8π watts. The field determined for $\theta > 90^\circ$ is zero resulting in 4π watts radiated over $\theta \leq 90^\circ$.

When `Far-Field Forced` is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

The number of azimuthal modes of the field is the same as for the modes in the waveguide i.e. given by the maximum index m_{\max} for m of the excitation modes, TE_{mn} and TM_{mn}

$$M = m_{\max}$$

The number of ϕ -cuts over 360° , N_ϕ , is then (see the remarks under *Spherical Wave Expansion (SWE)*)

$$N_\phi = \max((2M + 1), 4) = \max((2m_{\max} + 1), 4)$$

The minimum number of samples in θ is N_θ from 0° to 180° inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given

by the size of the horn aperture

$$N_\theta - 1 = N = kr_o + \max(10, 3.6\sqrt[3]{kr_o})$$

where k is the wavenumber and r_o is the radius of the smallest sphere centred at the origin of the feed coordinate system and enclosing the aperture.

For a given N the field may be determined for distances larger than r_{swe}

$$r_{swe} = \frac{N}{k}$$

Example

An example of the pattern for a conical horn is shown in the following figure. The aperture radius is 2λ , the flare length is 5λ and the aperture is excited with a TE_{11} mode. The near field is determined at a sphere with radius 5λ .

The correct near field is determined when Far-Field Forced is specified to 'off' and is shown as full-line patterns. When Far-Field Forced is specified to 'on' the near field is determined as the far field multiplied with the distance factor; these patterns are shown as dotted lines. For not mentioned attributes default values are applied.

In the spherical expansion we find

$$N_\phi = 4$$

and

$$N_\theta = 23$$

have been applied. This maximum mode number allows the field to be determined at distances

$$r > r_{swe} = N_\theta/k = 3.7\lambda$$

which is fulfilled here.

The figure shows an important difference when applying the correct near field (Far-Field Forced: off) compared to the far-field expression (Far-Field Forced: on). The near-field effect is significant as the distance at which the field is determined, 5λ , is considerable less than the traditional far-field distance

$$r_{ff} = 2D^2/\lambda = 32\lambda$$

between the near and the far field.

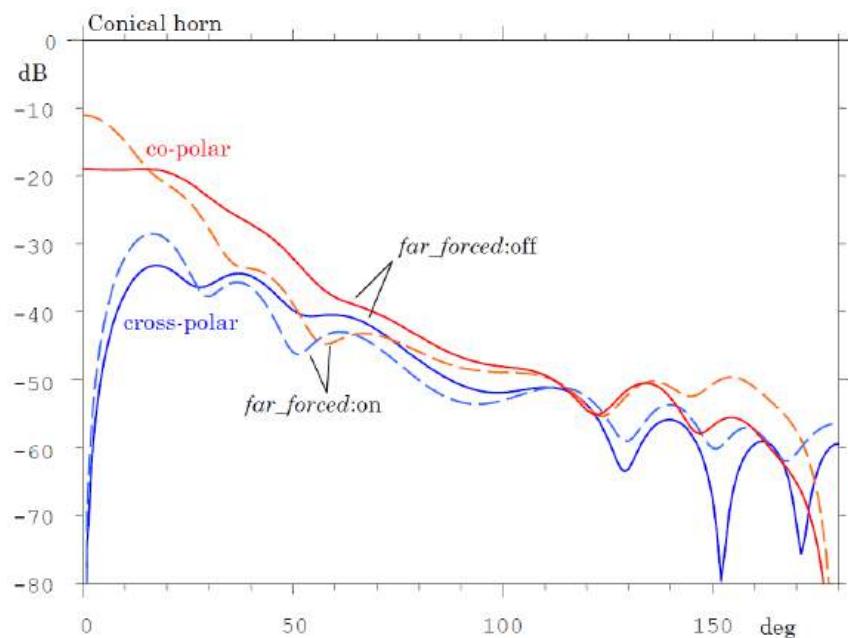


Figure 2

Near-field patterns for a conical horn, co- and cross-polar, at $\phi = 45^\circ$. For patterns in full lines Far-Field Forced is specified to 'off' and for dashed patterns Far-Field Forced is specified to 'on'. The latter are proportional to the far-field patterns (not shown) and the near-field effects are clearly seen.

FUNDAMENTAL MODE CIRCULAR WAVEGUIDE (fundamental_mode_circular_waveguide)

Purpose

In the class *Fundamental Mode Circular Waveguide* the radiation from a circular open-ended waveguide excited with the fundamental TE_{11} waveguide mode is modelled.

Links

Classes→*Electrical Objects*→*Feed*→*Horn*→*Fundamental Mode Circular Waveguide*

Remarks

Syntax

```
<object name> fundamental_mode_circular_waveguide
(
    frequency                  : ref(<n>),
    coor_sys                   : ref(<n>),
    aperture_radius            : <rl>,
    phase_displacement         : <rl>,
    ground_plane               : <si>,
    polarisation               : <si>,
    far_forced                 : <si>,
    factor                     : struct(db:<r>, deg:<r>),
    frequency_index_for_plot  : <i>
)
```

where

<i> = integer

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

<si> = item from a list of character strings

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Aperture Radius (*aperture_radius*) [real number with unit of length].

Radius of the circular waveguide aperture.

Phase Displacement (*phase_displacement*) [real number with unit of length], default: **0**.

Displacement of the phase reference point along the horn z_f -axis. A positive value of Phase Displacement moves the reference point along the positive z_f -axis, and vice versa for negative displacements. For details of the phase reference, see the Section [Phase of a Radiating Horn](#).

Ground Plane (*ground_plane*) [item from a list of character strings], default: **off**.

The horn aperture may either be placed in an infinite ground plane or radiate in free space.

off

The radiation is calculated as if the aperture is located in free space.

on

The radiation is calculated as if the aperture is located in an infinite ground plane (this is the best approximation if the feed is in a larger array).

Polarisation (*polarisation*) [item from a list of character strings], default: **linear_x**.

Definition of the polarisation of the beam.

linear_x

Linear polarisation along x_f , according to Ludwig's 3rd definition.

linear_y

Linear polarisation along x_f , according to Ludwig's 3rd definition.

rhc

Right-hand circular polarisation.

lhc

Left-hand circular polarisation.

Far Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on [Near-Field Calculations](#)).

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (factor) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index for Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute `Frequency` above a sequence of frequencies (wavelengths) is listed. The Frequency Index for Plot points to the number to be applied of the frequency (wavelength) in this sequence.

Remarks

The feed boresight is along the z_f -axis, $\theta = 0^\circ$, and the waveguide aperture is in the $x_f y_f$ -plane, $x_f y_f z_f$ being the feed coordinate system.

An Aperture Radius causing the TE_{11} mode to be cut-off is not permitted.

The pattern of the feed is calculated as a special case of the *Conical Horn* without flare. The pattern is normalised to dBi.

If the aperture is placed in a ground plane the field only exists for $\theta \leq 90^\circ$. As the spherical expansion requires the field on a full sphere, a mirrored source radiating within $90^\circ \leq \theta \leq 180^\circ$ is added resulting in a continuous field across the ground plane. The source itself is normalised to 4π watts and the spherical expansion including the mirrored source contains coefficients representing 8π watts. The field determined for $\theta > 90^\circ$ is zero resulting in 4π watts radiated over $\theta \leq 90^\circ$. The power will further be multiplied by the value of Factor (converted from dB to power).

When `far-forced` is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

Due to the rotational symmetry of the feed the number of azimuthal modes is limited to

$$M = 1$$

and the necessary number of ϕ -cuts over 360° is

$$N_\phi = 4$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by the size of the aperture

$$N_\theta - 1 = N = kr_o + \max(10, 3.6\sqrt[3]{kr_o})$$

where k is the wavenumber and r_o is the radius of the smallest sphere centred at the origin of the feed coordinate system and enclosing the aperture. For a given N the field may be determined for distances larger than r_{swe}

$$r_{swe} = N/k$$

Example

The following figure illustrates the effects of the near field for a fundamental mode circular waveguide with Aperture Radius 2λ and with default values for the other parameters. For this case

$$N_\phi = 4$$

and

$$N_\theta - 1 = N = 23$$

Thus $r_{swe} = 3.7\lambda$. The near field will be determined at a distance of 5λ .

When the near field is determined as the scaled far field (Far Forced is 'on') it has the same lobes as the far field. However, the correct near field based on the spherical wave expansion (Far Forced is 'off') shows the characteristics of a near field, a more smooth pattern with filled nulls. The effect is significant here as the near-field distance is only a fraction of the traditional far-field distance given by

$$r_{ff} = 2D^2/\lambda = 32\lambda$$

D being the diameter of the aperture, 4λ .

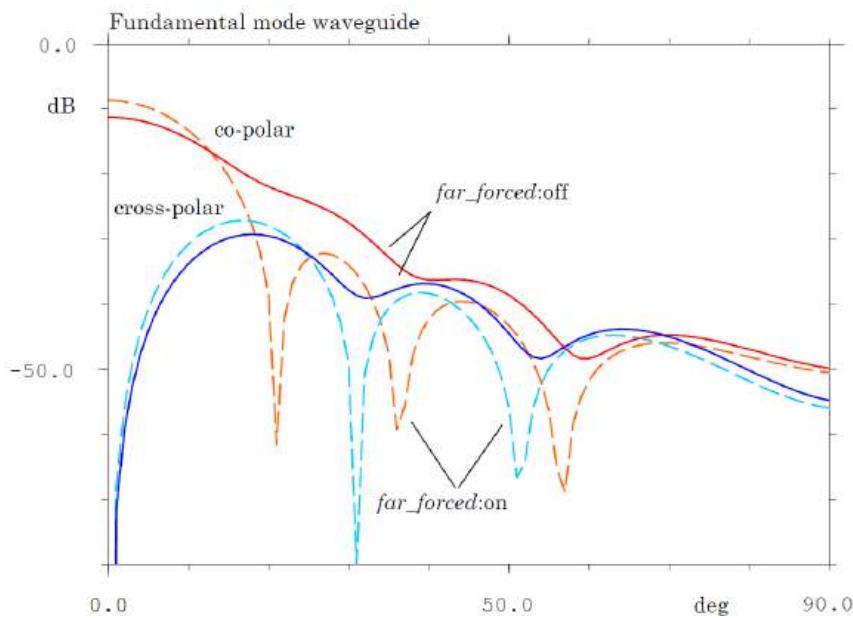


Figure 1

Near-field patterns, co and cx at $\phi = 45^\circ$, for a waveguide of diameter 4λ excited by a fundamental mode. For patterns in full lines Far Forced is specified to 'off'. For dashed patterns Far Forced is specified to 'on' and these patterns are proportional to the far-field patterns (not shown). The near-field effects are clearly seen.

POTTER HORN (potter_horn)

Purpose

A feed of the class *Potter Horn* approximates the radiation from a Potter horn. A quadratic phase variation in the aperture caused by a finite horn flare is included.

Links

Classes→*Electrical Objects*→*Feed*→*Horn*→*Potter Horn*

Remarks

Syntax

```
<object name> potter_horn
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    aperture_radius    : <rl>,
    flare_length        : <rl>,
    phase_displacement : <rl>,
    ground_plane        : <si>,
    polarisation        : <si>,
    far_forced          : <si>,
    factor              : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Aperture Radius (*aperture_radius*) [real number with unit of length].

Radius of the circular aperture.

Flare Length (*flare_length*) [real number with unit of length], default: **0**.

The slant length of the flare, measured from the edge of the aperture to the horn apex, see the figure below. If the flare length is less than the aperture radius, the flare is ignored.

Phase Displacement (*phase_displacement*) [real number with unit of length], default: **0**.

Displacement of the phase reference point along the horn z_f -axis. A positive value of Phase Displacement moves the reference point along the positive z_f -axis, and vice versa for negative displacements. For details of the phase reference, see [Phase of a Radiating Horn](#).

Ground Plane (*ground_plane*) [item from a list of character strings], default: **off**.

The horn aperture may either be placed in an infinite ground plane or radiate in free space.

off

The radiation is calculated as if the aperture is located in free space.

on

The radiation is calculated as if the aperture is located in an infinite ground plane (this is the best approximation if the feed is in a larger array).

Polarisation (*polarisation*) [item from a list of character strings], default: **linear_x**.

Definition of the polarisation of the field radiated by the Potter horn.

linear_x

Linear polarisation along x , according to Ludwig's 3rd definition.

linear_y

Linear polarisation along y , according to Ludwig's 3rd definition.

rhc

Right-hand circular polarisation.

lhc

Left-hand circular polarisation.

Far-Field Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on [Near-Field Calculations](#)).

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (factor) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (db) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (deg) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index for Plot (frequency_index_for_plot) [integer], default: **1.**

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute Frequency above a sequence of frequencies (wavelengths) is listed. The Frequency Index for Plot points to the number to be applied of the frequency (wavelength) in this sequence.

Remarks

A Potter horn is essentially a dual mode conical horn, excited with a prescribed combination of the circular waveguide modes TE_{11} and TM_{11} . Their relative amplitudes and phases are adjusted to cancel the electric field at the aperture boundary. The ratio of the mode amplitudes are $TE_{11}/TM_{11} = 1./0.4$ and the TM_{11} mode is rotated 90° relative to the TE_{11} mode.

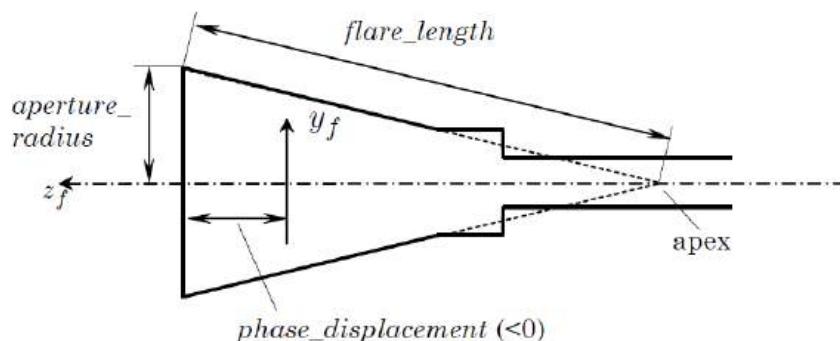


Figure 1 Potter horn with illustration of Aperture Radius and Flare Length in the feed coordinate system.

A horn flare, defined through the attribute Flare Length, results in a spherical wave front centred at the apex. Accordingly, the phase variation in the horn aperture is quadratic. For details of the phase reference, see [Phase of a Radiating Horn](#).

Modes below cut-off are not permitted.

The radiated power is normalised to 4π watts, so that the pattern is normalised to dBi. The total radiated power will further be multiplied by the value of Factor (converted from dB to power).

If the aperture is placed in a ground plane the field only exists for $\theta \leq 90^\circ$. As the spherical expansion requires the field on a full sphere, a mirrored source radiating within $90^\circ \leq \theta \leq 180^\circ$ is added resulting in a continuous field across the ground plane. The source itself is normalised to 4π watts and the spherical expansion including the mirrored source contains coefficients representing 8π watts. The field determined for $\theta > 90^\circ$ is zero resulting in 4π watts radiated over $\theta \leq 90^\circ$.

Due to the rotational symmetry of the feed and the excitation by the modes TE_{11} and TM_{11} , the number of azimuthal modes is limited to

$$M = 1$$

and the necessary number of ϕ -cuts over 360° is

$$N_\phi = 4$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by the size of the aperture

$$N_\theta - 1 = N = kr_o + \max(10, 3.6\sqrt[3]{kr_o})$$

where k is the wavenumber and r_o is the radius of the smallest sphere centred at the origin of the feed coordinate system and enclosing the aperture.

For a given N the field may be determined for distances larger than r_{swe}

$$r_{swe} = \frac{N}{k}$$

HYBRID MODE CONICAL HORN (hybrid_mode_conical_horn)

Purpose

In the class *Hybrid Mode Conical Horn* the radiation from a narrow flare angle, corrugated conical horn operating under the balanced-hybrid condition is modelled.

Links

Classes→*Electrical Objects*→*Feed*→*Horn*→*Hybrid Mode Conical Horn*

Remarks

Syntax

```
<object name> hybrid_mode_conical_horn
(
    frequency : ref(<n>),
    coor_sys : ref(<n>),
    waveguide_radius : <rl>,
    aperture_radius : <rl>,
    semi_flare_angle : <r>,
    phase_displacement : <rl>,
    ground_plane : <si>,
    modes : sequence(
        struct(type:<s>,
               amp:<r>,
               phase:<r>,
               rot:<r>),
        ...),
    list_mode : <si>,
    far_forced : <si>,
    factor : struct(db:<r>, deg:<r>),
    mode_power_normalization : <si>,
    frequency_index_for_plot : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Waveguide Radius (*waveguide_radius*) [real number with unit of length].

Radius of the circular waveguide leading to the horn (must be larger than 0). The value is only applied in plots of the feed, cf. the Figure in the remarks below, and has no effect on the field calculations.

Aperture Radius (*aperture_radius*) [real number with unit of length].

Radius of circular horn aperture (must be larger than the waveguide radius).

Semi Flare Angle (*semi_flare_angle*) [real number].

The flare angle in degrees, as measured from the axis of the horn. The Semi Flare Angle must be non-negative and must not exceed 20°. If the Semi Flare Angle is zero, a waveguide is considered. The Aperture Radius must then be equal to the Waveguide Radius.

Phase Displacement (*phase_displacement*) [real number with unit of length], default: **0**.

Displacement of the phase reference point along the horn z_f -axis. A positive value of Phase Displacement moves the reference point along the positive z_f -axis, and vice versa for negative displacements. For details of the phase reference, see *Phase of a Radiating Horn*.

Ground Plane (*ground_plane*) [item from a list of character strings], default: **off**.

The horn aperture may either be placed in an infinite ground plane or radiate in free space.

off

The radiation is calculated as if the aperture is located in free space.

on

The radiation is calculated as if the aperture is located in an infinite ground plane (this is the best approximation if the feed is in a larger array).

Modes (*modes*) [sequence of structs].

Several modes may be present in the aperture.

Type (*type*) [character string], default: **HE11**.

Mode type selector; see the remarks below for allowed values of the integers m and n .

Amplitude (*amp*) [real number], default: **1**.

Amplitude of the mode excitation (amp^2 gives the power).

Phase (*phase*) [real number], default: **0**.

Phase of the mode excitation, in degrees.

Rotation (*rot*) [real number], default: **0**.

Rotation (in degrees) of the mode from the x_f -axis towards the y_f -axis. For *rot*= 0, the HE_{11} mode is x_f -polarised.

List Mode (*list_mode*) [item from a list of character strings], default: **off**.

The modes which can propagate in the horn may be listed.

off

No list is given.

on

A list of the modes that can propagate in the horn will be written in the output file.

Far-Field Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on *Near-Field Calculations*):

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Mode Power Normalization (*mode_power_normalization*) [item from a list of character strings], default: **off**.

If mode power normalization is on, each mode is exactly normalized to 4pi by far field power integration. After normalization the individual excitations are used. The total power from the horn is still normalized to 4pi.

Frequency Index for Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute Frequency above a sequence of frequencies (wavelengths) is listed. The Frequency Index for Plot points to the number to be applied of the frequency (wavelength) in this sequence.

Remarks

In the present model, only the hybrid modes HE_{mn} ($1 \leq m \leq 8$ and $1 \leq n \leq 8$), EH_{mn} ($1 \leq m \leq 8$ and $2 \leq n \leq 8$) and the cylindrical modes TE_{0n} ($1 \leq n \leq 8$) and TM_{0n} ($2 \leq n \leq 8$) are supported (m is the azimuthal mode index and n is the radial mode index). As an example, mode HE_{23} is specified by type: HE23 in the attribute Modes. The EH_{m1} modes as well as TM_{01} are not present in the model since they most frequently will be high-frequency cut-off. Modes below cut-off should not be specified as this is not tested in the module.

By default the phase reference point is placed at the centre of the horn aperture. For details of the phase reference, see [Phase of a Radiating Horn](#).

The modal power is normalised so that Σamp^2 equals unity. The radiated power is normalised to 4π watts, so that the pattern is normalised to dBi. The total radiated power will further be multiplied by the value of Factor (converted from dB to power).

In the case the aperture is placed in a ground plane the field only exists for $\theta \leq 90^\circ$. As the spherical expansion requires the field on a full sphere, a mirrored source radiating within $90^\circ \leq \theta \leq 180^\circ$ is added resulting in a continuous field across the ground plane. The source itself is normalised to 4π watts and the spherical expansion including the mirrored source contains coefficients representing 8π watts. The field determined for $\theta > 90^\circ$ is zero resulting in 4π watts radiated over $\theta \leq 90^\circ$.

After a hybrid mode conical horn object has been defined, the output file may be inspected to check if the specified modes are cut-off.

The output file also contains information about the axial length of the horn as well as its flare length. See the Technical Description for a detailed documentation of this feed type.

When Far-Field Forced is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

The number of azimuthal modes of the field is the same as for the modes in the waveguide i.e. given by the maximum index m_{\max} for m of the excitation waveguide modes, TE_{mn} and TM_{mn}

$$M = m_{\max}$$

The number of ϕ -cuts over 360° , N_ϕ , is then (see the remarks under [Spherical Wave Expansion \(SWE\)](#))

$$N_\phi = \max((2M + 1), 4) = \max((2m_{\max} + 1), 4)$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by the size of the horn aperture

$$N_\theta - 1 = N = kr_o + \max(10, 3.6\sqrt[3]{kr_o})$$

where k is the wavenumber and r_o is the radius of the smallest sphere centred at the origin of the feed coordinate system and enclosing the aperture.

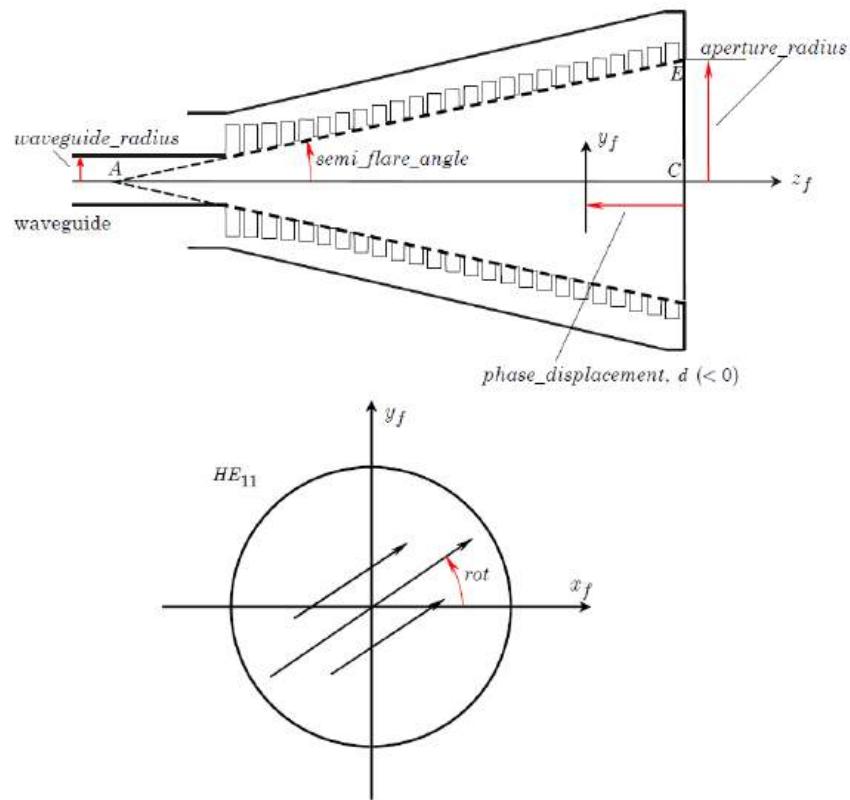


Figure 1

Illustration of a typical corrugated horn geometry, which may be modelled as a **Hybrid Mode Conical Horn**. Each mode is rotated an angle rot (in the attribute Modes), here illustrated for the HE_{11} mode. The axial length of the horn is measured from the apex A to the centre of the aperture C . The flare length is measured from A to the edge of the aperture E . These measures are calculated and printed in the output file when the object is defined.

For a given N the field may be determined for distances larger than r_{swe}

$$r_{swe} = \frac{N}{k}$$

Example

An example of the pattern for a hybrid mode conical horn is shown in the following figure. The `waveguide_radius` is 0.45λ and the Aperture Radius is 2λ . The Semi Flare Angle is 6° and the feed is excited with a HE_{11} mode. Default values are applied for the remaining attributes.

According to the expressions above we have

$$M = 1$$

and with $r_o = 2\lambda$

$$N_\theta - 1 = N = 23$$

resulting in

$$r_{swe} = 3.7\lambda$$

The distance at which the near field shall be determined is 5λ .

The correct near field is found when Far-Field Forced is specified to 'off' and is shown as full-line patterns. When Far-Field Forced is specified to 'on' the near field is determined as the far field multiplied with the distance factor; that pattern is shown with dotted line.

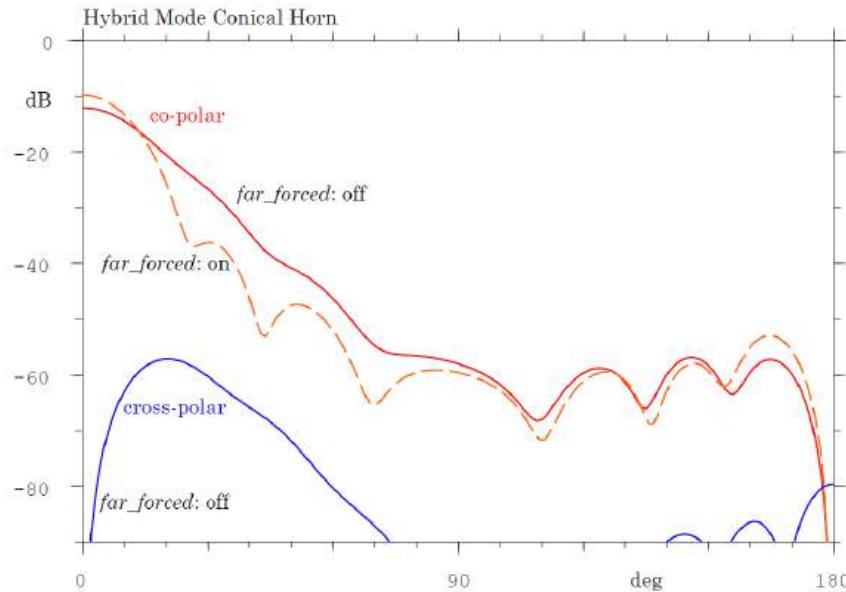


Figure 2

Near-field patterns for a hybrid mode conical horn, linear co-polar pattern at $\phi = 45^\circ$. For the pattern in full lines Far-Field Forced is specified to 'off' and for the dashed pattern Far-Field Forced is specified to 'on'. The latter are proportional to the far-field patterns (not shown). The pattern has a very high rotational symmetry and a low cross-polar field in the $\phi = 45^\circ$ plane.

RECTANGULAR HORN (rectangular_horn)

Purpose

The class *Rectangular Horn* approximates the radiation from a rectangular horn, or a rectangular open-ended waveguide. A quadratic phase variation in the aperture caused by a finite horn flare is included. The excitation may be a combination of rectangular waveguide modes.

Links

Classes→*Electrical Objects*→*Feed*→*Horn*→*Rectangular Horn*

Remarks

Syntax

```
<object name> rectangular_horn
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    aperture_width     : <rl>,
    aperture_height    : <rl>,
    flare_length_xz    : <rl>,
    flare_length_yz    : <rl>,
    phase_displacement : <rl>,
    ground_plane       : <si>,
    modes               : sequence(
                            struct(type:<s>,
                                   amp:<r>,
                                   phase:<r>),
                            ...),
    far_forced          : <si>,
    factor              : struct(db:<r>, deg:<r>),
    mode_power_normalization : <si>,
    frequency_index_for_plot : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Aperture Width (*aperture_width*) [real number with unit of length].

Width of the rectangular aperture along x_f .

Aperture Height (*aperture_height*) [real number with unit of length].

Height of the rectangular aperture along y_f .

Flare Length Xz (*flare_length_xz*) [real number with unit of length], default: **0**.

The slant length of the flare in the x_f, z_f -plane, measured from the edge of the aperture to the apex, see the figure under remarks. If the flare length is less than half the *aperture_width*, no flare is considered.

Flare Length Yz (*flare_length_yz*) [real number with unit of length], default: **0**.

The slant length of the flare in the y_f, z_f -plane, measured from the edge of the aperture to the apex, see the figure under remarks. If the flare length is less than half the *aperture_width*, no flare is considered.

Phase Displacement (*phase_displacement*) [real number with unit of length], default: **0**.

Displacement of the phase reference point along the horn z_f -axis. A positive value of *phase_displacement* moves the reference point along the positive z_f -axis, and vice versa for negative displacements. For details of the phase reference, see the Section *Phase of a Radiating Horn*.

Ground Plane (*ground_plane*) [item from a list of character strings], default: **off**.

The horn aperture may either be placed in an infinite ground plane or radiate in free space.

off

The radiation is calculated as if the aperture is located in free space.

on

The radiation is calculated as if the aperture is located in an infinite ground plane (this is the best approximation if the feed is in a larger array).

Modes (*modes*) [sequence of structs].

Several rectangular waveguide modes may be present in the aperture. When the mode type is given as TE_{mn} or TM_{mn} , then the first index refers to the variation along x_f , the second along y_f . Thus, the TE_{10} mode is y_f -polarised.

Type (*type*) [character string], default: **TE10**.

Mode type selector. The *type* is a string of four characters, the first two of which must be 'TE' or 'TM', while the third (*m*) and the fourth (*n*) are single-digit numbers within the range of physical modes: For TE_{mn} modes: $0 \leq m \leq 9$, $0 \leq n \leq 9$, excluding $m = n = 0$ For TM_{mn} modes: $1 \leq m \leq 9$, $1 \leq n \leq 9$.

Amp (*amp*) [real number], default: **1**.

Amplitude of the mode excitation (*amp*² gives the power).

Phase (*phase*) [real number], default: **0**.

Phase of the mode excitation, in degrees.

Far Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on [Near-Field Calculations](#)).

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Mode power normalization (*mode_power_normalization*) [item from a list of character strings], default: **off**.

If mode power normalization is on, each mode is exactly normalized to 4π by far field power integration. After normalization the individual excitations are used. The total power from the horn is still normalized to 4π .

Frequency Index For Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute *frequency* above a sequence of frequencies (wavelengths) is listed. The *frequency_index_for_plot* points to the number to be applied of the frequency (wavelength) in this sequence.

Remarks

The flares of the horn, given by the attributes `flare_length_xz`, and `flare_length_yz`, define the phase in the aperture to follow an astigmatic wave front centred at the corresponding two apices. Accordingly, the phase variation in the horn aperture is quadratic. For details of the phase reference, see the Section [Phase of a Radiating Horn](#).

Modes below cut-off are not permitted. The radiated power is normalised to 4π watts, so that the pattern is normalised to dBi. The total radiated power will further be multiplied by the value of `factor` (converted from dB to power).

If the aperture is placed in a ground plane the field only exists for $\theta \leq 90^\circ$. As the spherical expansion requires the field on a full sphere, a mirrored source radiating within $90^\circ \leq \theta \leq 180^\circ$ is added resulting in a continuous field across the ground plane. The source itself is normalised to 4π watts and the spherical expansion including the mirrored source contains coefficients representing 8π watts. The field determined for $\theta > 90^\circ$ is zero resulting in 4π watts radiated over $\theta \leq 90^\circ$.

See the Technical Description for a detailed documentation of this feed type.

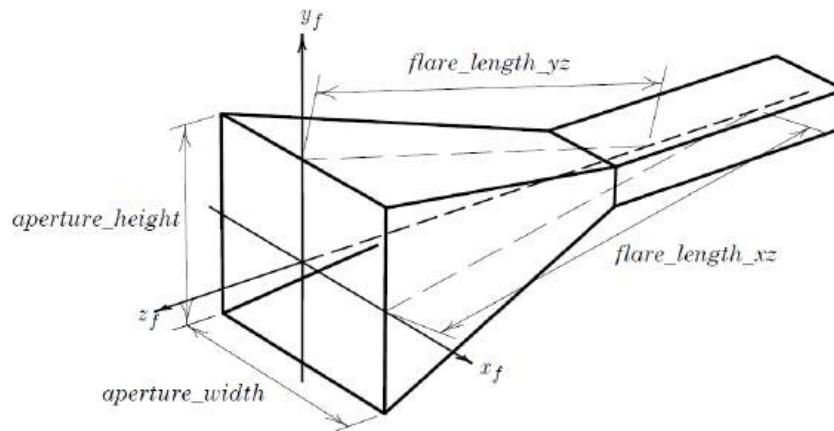


Figure 1

The rectangular horn with the definitions of the aperture (`aperture_width` and `aperture_height`) and of the flares (`flare_length_xz` and `flare_length_yz`). The figure is drawn for a `phase_displacement` of zero.

When `far-forced` is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

The number of azimuthal modes of the field is given by

$$M = kr_a + 10$$

where k is the wavenumber and r_a is the radius of the smallest sphere surrounding the aperture (half of the aperture diagonal).

The number of ϕ -cuts over 360° , N_ϕ , is then

$$N_\phi = 2M$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by the radius, r_o , of the smallest sphere centred at the origin of the feed coordinate system and enclosing the aperture

$$N_\theta - 1 = N = kr_o + \max(10, 3.6\sqrt[3]{kr_o})$$

For a given N the field may be determined for distances larger than r_{swe}

$$r_{swe} = \frac{N}{k}$$

HEXAGONAL HORN (hexagonal_horn)

Purpose

The class *Hexagonal Horn* approximates the radiation from a hexagonal waveguide or horn. A quadratic phase variation in the aperture caused by a finite horn flare is included. The excitation may be a combination of hexagonal waveguide modes.

Links

Classes→*Electrical Objects*→*Feed*→*Horn*→*Hexagonal Horn*

Remarks

Syntax

```
<object name> hexagonal_horn
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    diameter             : <rl>,
    flare_length         : <rl>,
    phase_displacement   : <rl>,
    ground_plane          : <si>,
    modes                : sequence(
                                struct(type:<s>,
                                       amp:<r>,
                                       phase:<r>,
                                       rot:<r>),
                                ...),
    far_forced           : <si>,
    factor               : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>,
    file_name             : <f>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Diameter (*diameter*) [real number with unit of length].

The diameter of the hexagonal aperture measured from one corner of the aperture to the opposite corner (see the Figure in the remarks below).

Flare Length (*flare_length*) [real number with unit of length], default: **0**.

The slant length of the flare measured from the apex to a corner of the hexagonal aperture (see the Figure below). If the flare length is less than half the aperture *diameter*, the flare is ignored.

Phase Displacement (*phase_displacement*) [real number with unit of length], default: **0**.

Displacement of the phase reference point along the horn z_f -axis. A positive value of Phase Displacement moves the reference point along the positive z_f -axis, and vice versa for negative displacements. For details of the phase reference, see the Section *Phase of a Radiating Horn*.

Ground Plane (*ground_plane*) [item from a list of character strings], default: **off**.

The horn aperture may either be placed in an infinite ground plane or radiate in free space.

off

The radiation is calculated as if the aperture is located in free space.

on

The radiation is calculated as if the aperture is located in an infinite ground plane (this is the best approximation if the feed is in a larger array).

Modes (*modes*) [sequence of structs].

Several hexagonal modes may be present in the aperture.

Type (*type*) [character string], default: **TE01**.

Mode type selector. The *type* shall be one of the following strings presenting the built-in modes in GRASP: 'TE01', 'TE11A', 'TE11B', 'TE21A', 'TE21B', 'TE31A', 'TE31B', 'TM01', 'TM11A', 'TM11B', 'TM21A', 'TM21B'. If other modes are needed, these must be defined on the file specified under *file_name*.

Amp (*amp*) [real number], default: **1**.

Amplitude of the mode excitation (amp^2 gives the power).

Phase (*phase*) [real number], default: **0**.

Phase of the mode excitation, in degrees.

Rot (rot) [real number], default: **0**.

Rotation in degrees of the mode from the x_f -axis towards the y_f -axis. When $rot = 0$, TE_{11a} is polarised along y_f and TE_{11b} along x_f . Only multiples of 60° should be selected.

Far Forced (far_forced) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on [Near-Field Calculations](#)).

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (factor) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (db) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (deg) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index For Plot (frequency_index_for_plot) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute *frequency* above a sequence of frequencies (wavelengths) is listed. The *frequency_index_for_plot* points to the number to be applied of the frequency (wavelength) in this sequence.

File Name (Obsolete) (file_name) [file name].

Name of file containing the hexagonal waveguide modes if other modes than those mentioned under *Modes* below are needed. The file format is described in Section [Hexagonal Wave Guide Modal Aperture Field](#). If no name is given, the waveguide modes defined in the code will be applied.

Remarks

The data for the available modes of the hexagonal horn imbedded in GRASP are generated by the finite element program IFEM from TICRA. Contact TICRA for further information if data are needed for other modes than those listed under the attribute *modes*.

A horn flare, defined through the attribute *flare_length*, results in a spherical wave front centred at the apex. Accordingly, the phase variation in the horn aperture is quadratic. For details of the phase reference, see the Section [Phase of a Radiating Horn](#). See also Figure 1 and Figure 2.

Modes below cut-off are not permitted. The radiated power is normalised to 4π watts, so that the pattern is normalised to dBi. The total radiated

power will further be multiplied by the value of factor (converted from dB to power).

If the aperture is placed in a ground plane the field only exists for $\theta \leq 90^\circ$. As the spherical expansion requires the field on a full sphere, a mirrored source radiating within $90^\circ \leq \theta \leq 180^\circ$ is added resulting in a continuous field across the ground plane. The source itself is normalised to 4π watts and the spherical expansion including the mirrored source contains coefficients representing 8π watts. The field determined for $\theta > 90^\circ$ is zero resulting in 4π watts radiated over $\theta \leq 90^\circ$.

See the Technical Description for a detailed documentation of this feed type.

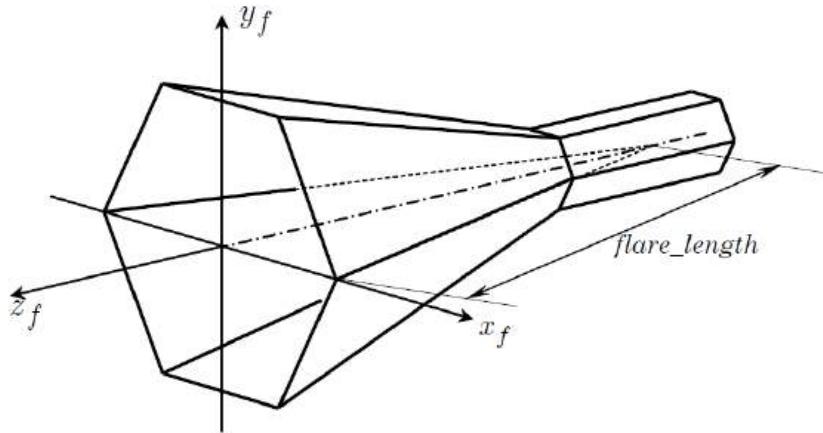


Figure 1 Hexagonal horn with a TE_{11a} mode, illustrating some of the key parameters: Illustration of the *flare_length*.

When *far-forced* is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

The number of azimuthal modes of the field is found empirically to be given by the minimum and maximum indices, m_{\min} and m_{\max} , for m of the excitation modes, TE_{mn} and TM_{mn}

$$M = \max(6 - m_{\min}, m_{\max})$$

The number of ϕ -cuts over 360° , N_ϕ , is then (see the remarks under *Spherical Wave Expansion (SWE)*)

$$N_\phi = \max((2M + 1), 4) = \max((2m_{\max} + 1), 4)$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by the size of the horn aperture

$$N_\theta - 1 = N = kr_o + \max(10, 3.6\sqrt[3]{kr_o})$$

where k is the wavenumber and r_o is the radius of the smallest sphere centred at the origin of the feed coordinate system and enclosing the aperture.

For a given N the field may be determined for distances larger than r_{swe}

$$r_{swe} = \frac{N}{k}$$

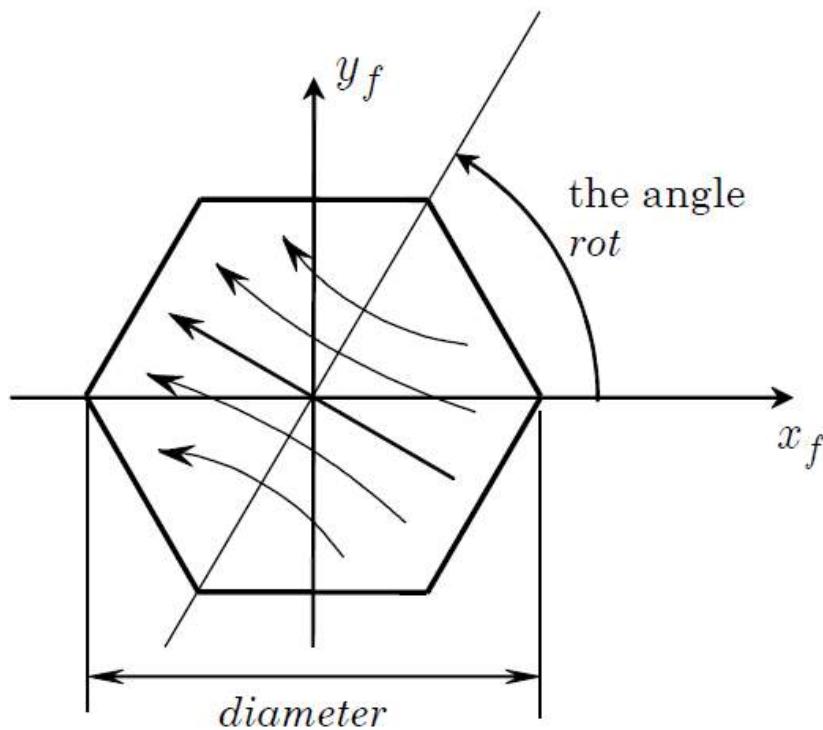


Figure 2

Hexagonal horn with a TE_{11a} mode, illustrating some of the key parameters: The *diameter* and the rotation *rot* (in attribute *modes*). The figure is drawn for a *phase_displacement* of zero.

Example

An example of the pattern for a hexagonal horn is shown in Figure 3. The aperture *diameter* is 2λ , the *flare_length* is 5λ and a *phase_displacement* of 0.1λ is applied. The feed is excited with a TE_{11a} mode (*y*-polarised on the axis). Otherwise the default values are applied.

Thus,

$$M = 5$$

and

$$N_\theta - 1 = N = 17$$

resulting in

$$r_{swe} = 2.7\lambda$$

The distance at which the near field is determined is 5λ .

The correct near field is determined when *far_forced* is specified to 'off' and is shown as full-line patterns. When *far_forced* is specified to 'on' the near field is determined as the far field multiplied with the distance factor; these patterns are shown as dotted lines.

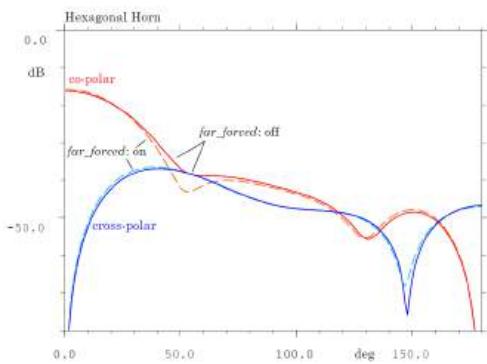


Figure 3

Near-field patterns for a hexagonal horn, linear co- and cross-polar, at $\phi = 45^\circ$. For patterns in full lines *far_forced* is specified to 'off' and for dashed patterns *far_forced* is specified to 'on'. The latter are proportional to the far-field patterns (not shown). For this feed the near-field effects are small.

TABULATED FEED

Purpose

This menu groups *Feed*s which are given in a tabular form. In spite of the menu name, *Tabulated Feed*, the patterns are not restricted to be patterns of a proper feed but may be representing any radiating source.

The table may either contain pattern data (e.g. a measured pattern)

Tabulated Pattern

or the table may contain mode coefficients for an mode expansion of the pattern.

Tabulated SWE Coefficients

Links

Classes→*Electrical Objects*→*Feed*→*Tabulated Feed*

TABULATED PATTERN (tabulated_pattern)

Purpose

The *Tabulated Pattern* class defines a source by means of field data tabulated on a spherical surface. The tabulated data shall be provided on a file in polar cuts, i.e. cuts for constant ϕ -values and varying θ , with θ and ϕ being usual spherical coordinates. The cuts shall be equidistant in ϕ , and must be either symmetric cuts, i.e. from $-\theta_{\max}$ to $+\theta_{\max}$, or asymmetric cuts from 0° to θ_{\max} . The number of cuts must be chosen to adequately sample the field from 0° to 360° in ϕ . An efficient interpolation is used to obtain the field at intermediate points.

The input field may be a far field or a near field.

Links

Classes → *Electrical Objects* → *Feed* → *Tabulated Feed* → *Tabulated Pattern*

Remarks

Syntax

```
<object name> tabulated_pattern
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    file_name           : <f>,
    file_format         : <si>,
    number_of_cuts      : <i>,
    power_norm          : <si>,
    phase_reference     : struct(x:<rl>, y:<rl>, z:<rl>),
    on_axis_adjust      : <si>,
    ij_notation         : <si>,
    near_far            : <si>,
    near_dist           : <rl>,
    max_m_mode_index   : <i>,
    max_n_mode_index   : <i>,
    far_forced          : <si>,
    factor              : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>,
    obsolete_cut_def    : <si>
)
```

where

<i> = integer

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

<f> = file name

<si> = item from a list of character strings

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the source operates. See the remarks below.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes to which the tabulated pattern refers. The phase reference may be changed to a different point, see the remarks below.

File Name (*file_name*) [file name].

Name of file containing the tabulated field data, usually a far-field pattern, cf. Near Far below. The file format shall be in accordance with TICRA .cut-files (not .grd files) as described in Section *Tabulated Pattern Data* for far fields, and Section *Field Data in Cuts*, spherical cuts for near fields. A far field is specified by two orthogonal field components. The near field must also contain the r-component.

File Format (*file_format*) [item from a list of character strings], default: **TICRA**.

Format of file.

TICRA

ASCII file in standard TICRA format. See Section *Tabulated Pattern Data* for far fields and *Field Data in Cuts* for near fields.

EDI

Obsolete file format name. The same as EDX.

EDX

Format according to the Electromagnetic Data Exchange (EDX) standard.

Number of Cuts (*number_of_cuts*) [integer], default: **0**.

Number of equidistant polar cuts per frequency in the file; at least one cut for each 90 degrees in phi. If the file contains cuts for only one frequency the value 0 may be specified. The number of cuts will then be determined from the file (see also the remarks below).

Power Normalisation (*power_norm*) [item from a list of character strings], default: **off**.

If on, the total radiated power is equal to 4π Watts, which then expresses the pattern in dBi.

off

No normalisation.

on

The field is power normalised to 4π .

Phase Reference (*phase_reference*) [struct].

When the input field is a far field (Near Far is 'far') the phase reference of the cuts may be altered. Assume the tabulated field data originate in a measurement. The phase of the input field then refers to the centre of rotation in the measurement. If the phase shall refer to another point (x, y, z) in the input coordinate system (given by attribute Coordinate System above) then the Phase Reference shall be specified to this point. See also the remarks below.

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of new the phase-reference point.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of new the phase-reference point.

z (*z*) [real number with unit of length], default: **0**.

z-coordinate of new the phase-reference point.

On-Axis Adjustment (*on_axis_adjust*) [item from a list of character strings], default: **off**.

Determines a possible on-axis adjustment of the field level in the input cuts.

off

No adjustments made.

on

All cuts are scaled by a complex factor so that the amplitude, phase and polarization at $\theta = 0^\circ$ are the same as in the first read cut.

ij-Notation (*ij_notation*) [item from a list of character strings], default: **j**.

Specifies the time-factor for the field data.

j

The time factor is $e^{+j\omega t}$.

i

The time factor is $e^{-i\omega t}$.

Near Far (*near_far*) [item from a list of character strings], default: **automatic**.

Specifies whether the tabulated field is a near or a far field. A far field requires two orthogonal field components, a near field contains an additional *r*-component, although this component is not used.

far

The tabulated field is a far field. Two field components will be read.

near

The tabulated field is a near field. Two field components will be read. Although near-field data contain an additional r-component this is not used.

automatic

It is automatically detected if the tabulated field is a near- or far-field.

Near-Field Distance (*near_dist*) [real number with unit of length], default: **0**.

Radius of the sphere on which the near-field data is specified. Not used when Near Far is specified to 'far'.

Max m-Mode Index (*max_m_mode_index*) [integer], default: **-1**.

Maximum number of azimuthal (*m*) modes to be used in the azimuthal ϕ -interpolation of the tabulated data and to be included in the spherical-wave expansion of the field. If negative, all azimuthal modes are retained. Max m-Mode Index shall not exceed the following Max n-Mode Index.

Max n-Mode Index (*max_n_mode_index*) [integer], default: **-1**.

Maximum polar mode index (*n*) to be included in the spherical-wave expansion of the field. If negative, all *n*-modes are included. Note: It is recommended to limit the number of modes according to the extent of the source. The field may not be determined correctly at distances less than Max n-Mode Index/k, k being the wavenumber. See the Remarks for details.

Far-Field Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on *Near-Field Calculations*).

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor (after a possible power normalisation).

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index for Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the source shall be plotted. In the attribute *Frequency* above a sequence of frequencies (wavelengths) is listed. The Frequency Index for Plot points to the number to be applied of the frequency (wavelength) in this sequence.

Cut Definition (Obsolete) (*obsolete_cut_def*) [item from a list of character strings], default: **automatic**.

Defines the type of the polar cuts (see the remarks below for restrictions and polarisation).

asymmetric

The cuts are from 0° to θ_n ($\theta_n > 0^\circ$).

symmetric

The cuts are from $-\theta_n$ to $+\theta_n$ ($\theta_n > 0^\circ$). The number of samples in each cut must be odd, so that the boresight direction is included.

automatic

Detect automatically if cuts are symmetric or asymmetric.

Remarks

This section contains discussions on the following topics

- Power normalisation
- Handling more than one frequency
- Changing the phase reference
- Polarisation conventions
- Restrictions on θ and ϕ in the tabulated grid
- Method of field determination
- Maximum spacing for the tabulated field data
- Maximum number of modes to be included
- Illustration of too coarse azimuthal sampling (too few m -modes)

Power normalisation

When Power Normalisation is specified to 'on', the feed pattern is normalised to dB_i, i.e. to radiate 4π watts. The total radiated power will be 4π times the value of Factor (converted from dB to power).

More than one frequency

The file with input patterns must contain a pattern for each frequency specified in the *Frequency* object and the patterns must be stored in the same order as the frequencies. For each frequency the number of pattern cuts is given by Number of Cuts.

If the *Frequency* attribute is specified, the file with input patterns must contain a pattern for each frequency. The number of patterns must agree with the number of frequencies and must be stored in the same order as the frequencies. The patterns for each of the frequencies must contain a number of pattern cuts as given by *Number of Cuts*.

If only one frequency is given then *Number of Cuts* may be specified to zero and the number of cuts is determined by scanning through the file.

If the *Tabulated Pattern* object is specified in an object of one of the classes in *Feed* then it is required that the *Frequency* object specified in the *Feed* object is the same as the *Frequency* object in the *Tabulated Pattern* object. However, it is permitted not to specify a *Frequency* object in the *Tabulated Pattern* object. The file must then only contain one pattern, and this pattern will then be used for all frequencies.

Phase reference

The input pattern refers to a coordinate system, the input coordinate system, specified by the attribute *Coordinate System*. The origin of the input coordinate system is the phase reference point for the specified tabulated field.

Typically, the pattern originates in a measurement in which a source is rotated around two perpendicular axes. The point of intersection of the two axis is denoted the centre point for rotation and the input coordinate system has origin at this point.

If the tabulated pattern originates in a calculation the input coordinate system is equivalent to the coordinate system in which the calculated field is specified.

Consider for example a horn which is to be applied as feed for a reflector antenna. The horn pattern is measured (or calculated) in cuts of which a single is shown in Figure 1. The centre of the cut (the centre point for rotation) is denoted M , and the horn is aligned so this centre is in the middle of the aperture. This is normally not the phase centre for the pattern as the phase centre is positioned slightly behind the aperture, at O (shown exaggerated in Figure 1).

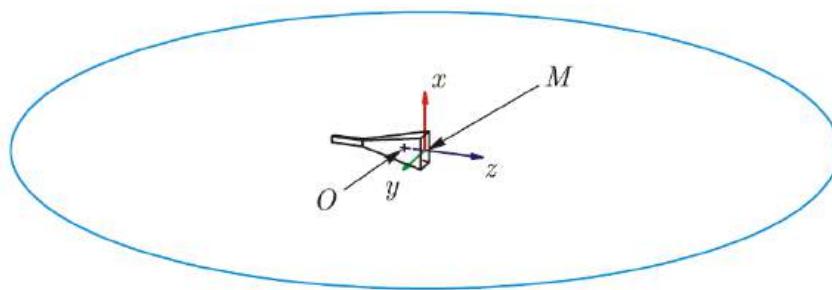


Figure 1

Measuring the field of a horn in cuts centred at the centre of the aperture, M (only a horizontal cut is shown). M is then the phase reference point. The phase centre of the horn is at O .

When this horn is applied as feed for a paraboloidal reflector antenna, Figure 3, it shall ideally be positioned with its phase centre, O , at the focal point of

the paraboloidal reflector. There is, however, no geometrical information in a measured pattern apart from the measurement coordinate system given by its origin and the directions of its axes as illustrated in Figure 1.

The phase reference point may be changed for a far-field pattern. This is the same as in a measurement to change the position of the horn relative to the centre point for rotation. By specifying a phase centre as the position of O relative to M , the tabulated pattern will be modified to have reference to the point O , cf. Figure 2, by multiplying the field by the factor

$$e^{-jk\bar{V}\cdot\hat{r}}$$

where $k = 2\pi/\lambda$ is the wavenumber, \bar{V} is the vector \overrightarrow{MO} and \hat{r} is the far-field direction for the actual data point,

$$\hat{r} = \hat{x} \sin \theta \cos \phi + \hat{y} \sin \theta \sin \phi + \hat{z} \cos \theta$$

This phase change is performed on the input field automatically when the attribute Phase Reference is specified to the values of x , y and z where $\bar{V} = (x, y, z)$ is the vector \overrightarrow{MO} .

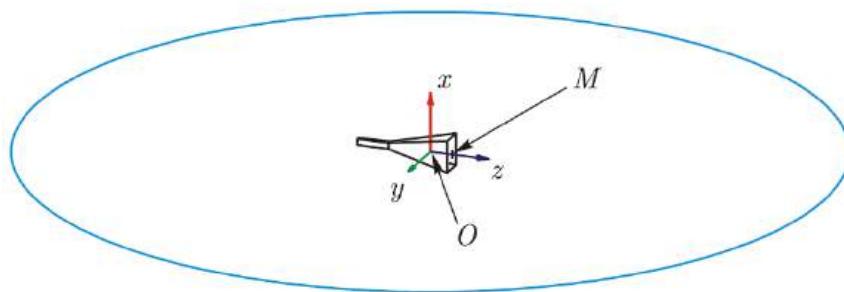


Figure 2 Redefining the phase reference point to O . Far-field cuts are then centred at O . The previous phase reference point is at the centre of the aperture, M .

When the tabulated pattern is applied as source it will now be positioned with its xyz -coordinate system (as shown in Figure 2) congruent to the input coordinate system specified in the attribute Coordinate System. Shall, for example, the horn be applied as feed for a paraboloidal reflector antenna, Figure 3, it shall be positioned with its xyz -coordinate system identical to the feed coordinate system, $x_fy_fz_f$, which is defined at F , the focal point of the paraboloidal reflector.

If the tabulated pattern is a near field, the change of the phase reference can not be carried out as described above. Instead, the far field of the tabulated pattern can be calculated. This far field may then be printed on a file and this file may be applied as input to a new **Tabulated Pattern** for which the phase reference may be changed as described.

Polarisation conventions

For asymmetric cuts, the θ -polarisation vector must point away from the pole $\theta = 0^\circ$ and the ϕ -polarisation vector must be counter-clockwise, see Figure 9-4(a).

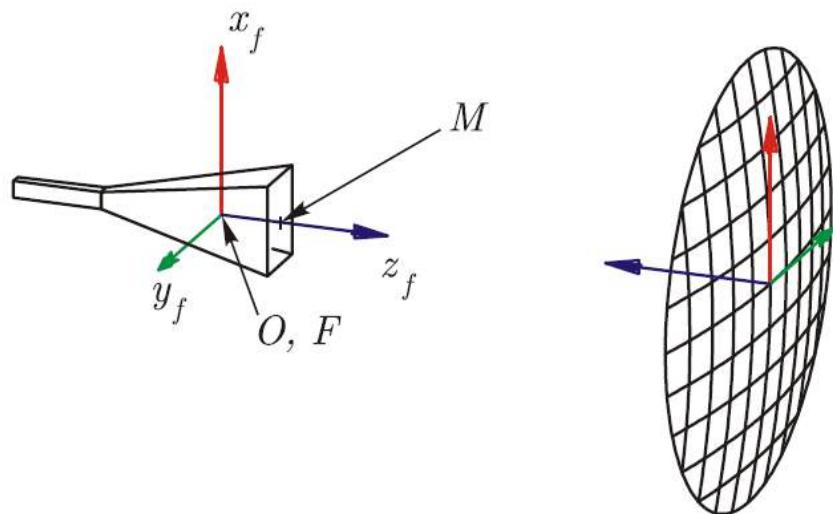


Figure 3

The horn mounted as feed for a paraboloidal reflector. The phase reference point O is positioned at the focal point F of the paraboloid.

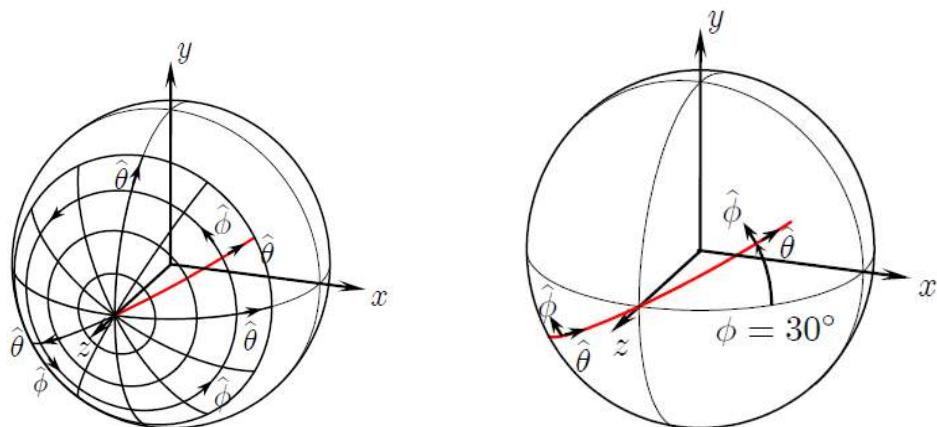
(a) $\theta\phi$ -polarisation for asymmetric cuts (for $0^\circ \leq \theta \leq 60^\circ, 0^\circ \leq \phi < 360^\circ$)(b) $\theta\phi$ -polarisation for symmetric cuts (for $-60^\circ \leq \theta \leq 60^\circ, \phi = 30^\circ$)

Figure 4

Orientation of the θ - and ϕ -polarisation vectors for asymmetric cuts (a) and for symmetric cuts (b). One cut is shown in red for both cases; for the latter case only this single cut is shown.

For symmetric cuts, the θ - and ϕ -polarisations shall follow the conventional definition as for the asymmetric cuts when θ is positive. But for negative θ -values, the θ - and ϕ -polarisation vectors must point opposite the conventional definition in order to ensure a continuous polarisation definition across the pole.

See the Technical Description for the other polarisation definitions, and the appendix *Ludwig's 3rd Definition of Polarization* for important details of the definition at the poles $\theta = 0^\circ$ and $\theta = 180^\circ$.

Restrictions on θ and ϕ in the tabulated grid

The spacing in θ , as well as the spacing in ϕ , must be equidistant. The ϕ -samples shall cover full azimuthal circles, the θ -samples may be truncated, i.e. θ may run from 0° to θ_{\max} with $\theta_{\max} \leq 180^\circ$. The cuts must appear as scans in θ .

Each input cut must then be defined in either the interval from 0° to θ_{\max} (asymmetric cuts) or in the interval from $-\theta_{\max}$ to θ_{\max} (symmetric cuts). With, say, $N_{\phi,A}$ asymmetric cuts over 360° , the ϕ -values of the cuts must be given by

$$\phi_i = \phi_0 + \frac{360^\circ}{N_{\phi,A}}(i - 1), \quad i = 1, \dots, N_{\phi,A} \quad (1)$$

where ϕ_0 is the ϕ -value of the first cut. $N_{\phi,A}$ must be an even number, larger than or equal to 4.

Similarly, with $N_{\phi,S}$ symmetric cuts over 180° , the ϕ -values of the cuts must be at

$$\phi_i = \phi_0 + \frac{180^\circ}{N_{\phi,S}}(i - 1), \quad i = 1, \dots, N_{\phi,S} \quad (2)$$

where $N_{\phi,S}$ must now be larger than or equal to 2.

The spacing of the field values in these cuts shall also be constant, i.e. for asymmetric cuts with $N_{\theta,A}$ field points

$$\theta_i = \frac{\theta_{\max}}{(N_{\theta,A} - 1)}(i - 1), \quad i = 1, \dots, N_{\theta,A} \quad (3)$$

Irrespective of the type of cuts, symmetric or asymmetric, the pole at $\theta = 0^\circ$ must always be included. As a consequence, the number of θ -values from $-\theta_n$ to θ_n , $N_{\theta,S}$, in a symmetric cut must be odd and the θ -values are

$$\theta_i = \frac{2\theta_{\max}}{(N_{\theta,S} - 1)}\left(i - \frac{N_{\theta,S} + 1}{2}\right), \quad i = 1, \dots, N_{\theta,S}. \quad (4)$$

If the last θ -value, θ_{\max} , is not 180° then it must be assured that the field has a low value at θ_{\max} for all values of ϕ . The interpolation inaccuracies which will occur around the truncation at θ_{\max} will be proportional to the input field values at θ_{\max} .

Method of field determination

If Far-Field Forced is specified to 'off' which is recommended, then the input field will be expanded in spherical modes from which the field is then determined.

The spherical wave expansion requires field information for a full sphere and field values zero will be inserted at the grid points with $\theta > \theta_{\max}$. It is further required that $\theta = 180^\circ$ is a point in this extended grid. If the given θ -spacing is not divisor in 180° the spacing will be changed (decreased) to include both $\theta = 0^\circ$ and $\theta = 180^\circ$. The field values will be interpolated by third order polynomial (cubic) interpolation to the new sample points within $|\theta| < \theta_{\max}$ (note: the cubic interpolation requests a much denser sampling, see the next section).

If Far-Field Forced is specified to ‘on’ the field will be calculated based on the input field scaled to the requested distance. Field values at points different from the tabulated input points will be interpolated. In the azimuthal angle ϕ the interpolation will be based on a Fourier expansion (similarly to the spherical modes). In the polar angle, θ , a cubic interpolation will be applied. It is noted that this requests a much denser sampling of the tabulated field as compared to the spherical wave expansion normally used when Far-Field Forced is specified to ‘off’.

For both cases field values determined for $\theta > \theta_{\max}$ will be set to zero.

Maximum spacing for the tabulated field data

In order to be able to regenerate a radiation pattern from tabulated data, it is necessary that the data spacing fulfils the Nyquist sampling criterion. This results in requirements to a maximum spacing for the tabulated data.

Theoretically, if the radiating or scattering structure is confined within a spherical volume with radius r_0 , measured from the reference centre for the tabulated field data (defined by the attribute Phase Reference), then the sample spacing shall be $\lambda/2$ or less over this spherical surface. The minimum number of samples for $0^\circ \leq \theta \leq 180^\circ$ is denoted N_θ so we have $N_\theta - 1$ intervals. This gives a minimum value for N_θ

$$N_\theta - 1 = \frac{\pi r_0}{\lambda/2} = kr_0 \quad (5)$$

The theoretical number of polar modes is hereby $N = N_\theta - 1$. Modes with a higher polar index will drop in amplitude for increasing index value and in order to ensure a sufficient accuracy, the number of polar modes, and hereby the number of samples, shall be increased so that at least

$$N_\theta - 1 = N = kr_0 + \max(10, 3.6\sqrt[3]{kr_0}) \quad (6)$$

shall be applied (rounded up to an integer). Polar modes may then be determined with mode indices n

$$n = 1, 2, 3, \dots, N \quad (7)$$

The maximum allowed angular spacing, $\Delta\theta$, in the polar cut occurs with the minimum value of N :

$$\Delta\theta = \frac{\pi}{N} = \frac{180^\circ}{N} \quad (8)$$

with N given by Eq. (6).

However, when a cubic interpolation procedure in the θ -coordinate is applied (see ‘Method of field determination’ above), a spacing at least four times denser is required, i.e.

$$\Delta\theta \leq \frac{180^\circ}{4N}. \quad (9)$$

The sampling may be truncated in θ when the field level at the truncation angle, θ_{\max} , is low. For $\theta_{\max} < 90^\circ$, the circumference of the largest circle in

ϕ is $2\pi r_0 \sin \theta_{\max} < 2\pi r_0$. On this circle there shall be at least N_ϕ samples spaced by $\lambda/2$, thus N_ϕ is given by

$$N_\phi = \frac{2\pi r_0 \sin \theta_{\max}}{\lambda/2} = 2N \sin \theta_{\max} \quad (10)$$

where N_ϕ shall be rounded up to an even integer. The required maximum spacing in ϕ is therefore

$$\Delta\phi = \frac{2\pi}{N_\phi} = \frac{180^\circ}{N \sin \theta_{\max}} \quad (11)$$

With this sample spacing azimuthal modes may be determined with mode indices m

$$m = 0, \pm 1, \pm 2, \dots, \pm M \quad (12)$$

i.e. N_ϕ samples may result in $2M + 1$ mode coefficients, and M cannot exceed

$$M = \frac{N_\phi}{2} - 1 \quad (13)$$

Some information is lost as we determine $2M + 1$ coefficients from N_ϕ samples, and

$$2M + 1 = N_\phi - 1 \quad (14)$$

The lost information is the coefficients for the modes with $m = N_\phi/2$ and $m = -N_\phi/2$ which can not be separated as a consequence of the applied Fourier technique. If the sampling is too sparse the power content of the modes with $|m| = N_\phi/2$ will be important and the tabulated pattern values can not be reproduced from the input.

If the tabulated pattern is not available with spacing in θ and ϕ as fine as requested in Eqs. (9) and (11) the interpolated pattern will not be accurate. This is illustrated in a later section.

Maximum number of modes to be included

The data may be sampled in a denser grid, than that required for the interpolation procedure. This may be utilized in the spherical wave expansion by restricting the number of applied modes to the physically relevant. Irregularities in the sampled data, resulting from e.g. noise, instrumentation errors or other sources, may then be filtered out, provided they manifest themselves in the higher order harmonics.

In addition to saving computation time there is thus a reason of noise reduction to reduce the number of applied spherical modes. Further, care shall be taken such that N does not exceed kr , r being the input field distance Near-Field Distance (the measurement distance). This will not be a problem for far-field measurements.

The maximum value of the polar mode index (n) may thus be reduced from $N_\theta - 1$ to N as given by Eq. (6) by specifying

$$\text{Max n-Mode Index} = N = kr_0 + \max(10, 3.6\sqrt[3]{kr_0}) \quad (15)$$

Otherwise

$$\text{Max n-Mode Index} = N = N_\theta - 1 = \frac{180^\circ}{\Delta\theta} \quad (16)$$

will be applied.

Modes with polar index n may represent sources out to a radius r given by $n = kr$, i.e.

$$r = n/k \quad (17)$$

If the spacing is dense, N as given by Eq. (16) may represent sources further out than the measurement distance r (Near-Field Distance). But this is unphysical. The mode number must then be reduced to

$$\text{Max n-Mode Index} \leq kr \quad (18)$$

In order to ensure convergence N shall be determined from Eq. (15) and may hereby represent sources out to

$$r = N/k \quad (19)$$

As a consequence it is thus not possible to determine the field closer than 1.6λ from the source-surrounding minimum sphere given by the radius r_0 (the value 1.6λ corresponds to 10 is applied in Eq. (6) or Eq. (15)).

Correspondingly, a tabulated pattern shall be characterised from measurements at a distance which is at least a few wavelengths larger than r_0 .

For critical cases a careful analysis of the mode spectrum (printed in the standard output file) is necessary to determine the best value of N to be applied. For further discussions on an example of a mode spectrum see class *Spherical Wave Expansion (SWE)*.

As for the polar modes, the azimuthal mode index m may be reduced according to Eqs. (10) and (13)

$$\text{max_m_mode_index} = (\text{Max n-Mode Index}) \sin \theta_{\max} - 1 \quad (20)$$

for truncation at $\theta_n < 90^\circ$. Otherwise the program will apply

$$\text{max_m_mode_index} = \frac{N_\phi}{2} - 1 \quad (21)$$

cf. Eq. (13), or

$$\text{max_m_mode_index} = N \quad (22)$$

whichever is the smallest.

Illustration of too coarse azimuthal sampling (too few m -modes)

It happens that only a very few azimuthal cuts are available, for example the cuts in the principal planes of the pattern ($\Delta\phi = 90^\circ$).

A too coarse spacing in azimuth ($\Delta\phi$ is too large) will result in an insufficient number of m -modes to be calculated. The power of the omitted high-order

modes will be distributed in the calculated modes which therefore are incorrect. The phenomenon is well-known in Fourier analysis and is denoted aliasing.

The effect is illustrated in the following example where, for three different horns, the patterns have been accurately calculated, each in $N_\phi = 12$ asymmetric cuts from $\theta = 0^\circ$ to $\theta_{\max} = 90^\circ$ (see Figure 5). The sample spacing in ϕ , $\Delta\phi = 30^\circ$, means that the maximum azimuthal mode index is restricted to $M = 5$, cf. Eq. (13).

For each horn, the phase reference is at the centre of the aperture. The spherical modes have been calculated, and the pattern at $\phi = 45^\circ$ has been determined from the modes. A comparison to the directly calculated patterns at $\phi = 45^\circ$ then allows an estimate of the accuracy.

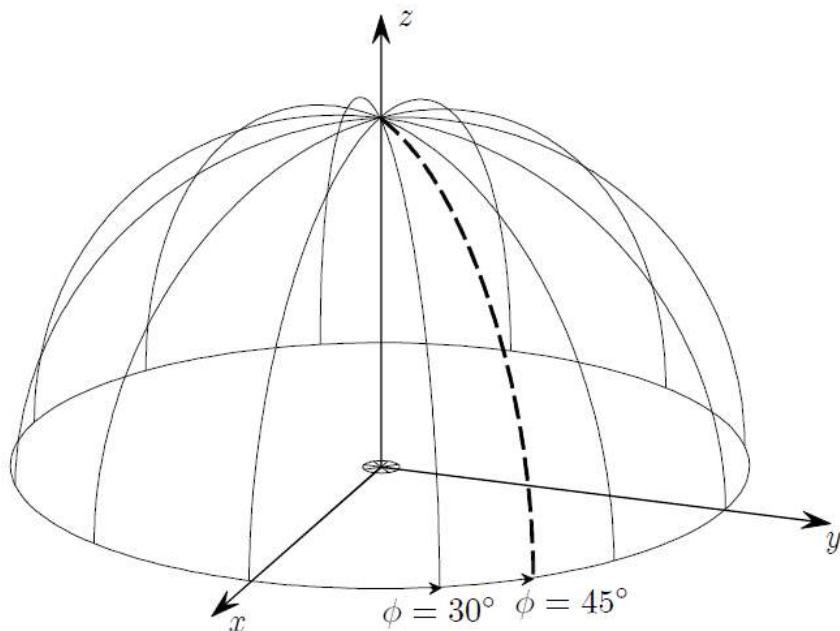


Figure 5

Polar pattern cuts from the pole at $\theta = 0^\circ$ to the equator plane, for every 30° in ϕ for azimuthal mode generation. The pattern is to be reconstructed at $\phi = 45^\circ$ as shown dashed. Even though the pattern cuts are shown in the near field, the calculations are carried out in the far field. A circular aperture is shown at origin.

In Figure 6 the three horns are shown: Horn 1 is a circular horn, 1λ in diameter, excited by the fundamental TE_{11} -mode, horn 2 a rectangular horn with side-lengths 4λ by 1λ excited with the fundamental TE_{10} -mode (H-plane horn), and finally horn 3 shows a square horn with a side-length of 1λ , also carrying a TE_{10} -mode.

The radius r_0 of the horns are given in Table 1 along with the theoretical maximum limits for the number of modes, N and M , as given by Eqs. (6) and (10), the latter with $\theta_n = 90^\circ$ resulting in the upper limit for M is N . It is seen that M shall be specified to at least 14, 23 and 15 for the three horns, if determined only by the size of the horn.

However for horn 1, the TE_{11} -mode excitation in the circular horn only contains azimuthal modes with $m = \pm 1$. Since the circular geometry cannot

generate higher order m -modes, $M = 1$ will be sufficient for this horn with this excitation.

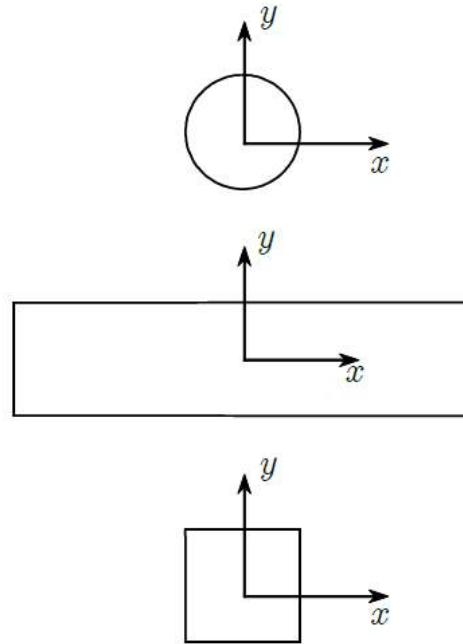


Figure 6 Aperture geometries of investigated horns: Circular, rectangular and square, respectively. The horns are y -polarised.

	Horn		
	Circular	Rectangular	Square
r_0	0.5λ	2.06λ	0.71λ
kr_0	3.1	13.0	4.4
$M = N = kr_0 + 10$	14	23	15
$\theta_{\max, M=5}$	21°	13°	19°

Table 1 Theoretical minimum number of modes for the three horns.

We can with the given ϕ -spacing determine azimuthal modes only up to $M = 5$ which is insufficient for the rectangular and the square horns.

The limit on the azimuthal mode index may be used to determine the maximum polar angle, up to which the field may be reconstructed, cf. Eq. (10)

$$\sin \theta_{\max, M=5} = 5/N \quad (23)$$

where $M = 5$ has been inserted. The values of $\theta_{\max, M=5}$ for the three horns are also given in Table 1.

The power content in each of the available m -modes has been determined and is listed in Table 2. As expected, it is seen that the circular horn contains modes for $m = \pm 1$ only.

	Horn		
	Circular	Rectangular	Square
0	0.000000	0.000000	0.000000
± 1	0.500000	0.257131	0.492881
± 2	0.000000	0.000000	0.000000
± 3	0.000000	0.126948	0.007031
± 4	0.000000	0.000000	0.000000
± 5	0.000000	0.115922	0.000088

Table 2 Power content in azimuthal modes.

We now compare the patterns reconstructed from the modes of Table 2 with the directly calculated patterns for $\phi = 45^\circ$. The total power is normalised to 4π for all patterns.

The pattern of the circular horn is completely described by modes with $m = \pm 1$. Hence, truncation of modes with indices larger than $M = 5$ does not deteriorate the pattern. This is confirmed by the plot in Figure 9-7(a) in which the patterns calculated by the two methods are identical.

With respect to the rectangular horn, Table 2 shows that the modes with $m = \pm 5$ contain almost half as much power as the modes with $m = \pm 1$. Since the required M is $23 \gg 5$, it is expected that there will be a significant difference between the true pattern at $\phi = 45^\circ$ and the pattern reconstructed from the azimuthal modes. This is seen in Figure 9-7(b). But a good correlation between the co-polar components exists close to boresight. According to the last line in Table 1 the patterns shall agree up to $\theta = 13^\circ$. To improve the correlation for larger θ the number of cuts from which the azimuthal modes are generated must be increased.

The pattern from the square horn also contains higher-order modes. However, the power content of the modes falls off much faster than that of the rectangular horn, Table 2. The comparison of the patterns, Figure 9-7(c), also shows that only a small discrepancy exists and only at the low level in the cross-polarisation.

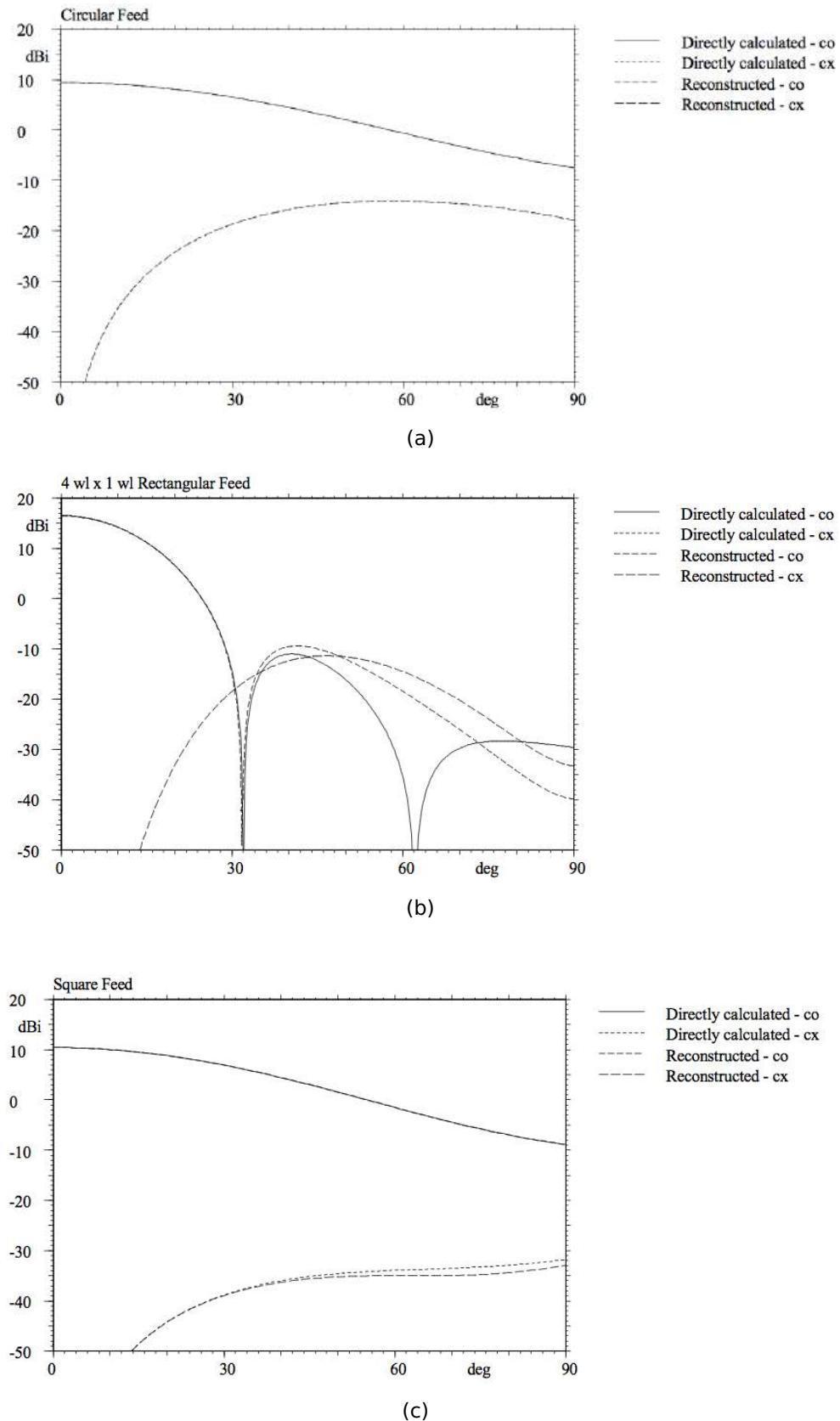


Figure 7

Comparison between horn patterns calculated directly and reconstructed from azimuthal mode expansion at $\phi = 45^\circ$.

TABULATED SWE COEFFICIENTS (tabulated_swe_coefficients)

Purpose

The class *Tabulated SWE Coefficients* defines a feed pattern by means of a spherical wave expansion. The coefficients to the expansion are read from a user supplied file. The spherical expansion may be evaluated at the appropriate distance, which ensures a correct near field.

In some GTD applications, it is desirable to have a near field phase that behaves as the phase from a point source. This can be accomplished by enforcing Far-Field Forced to 'on'. The near field is then simply calculated by multiplying the far-field pattern by the distance factor $\exp(-jkr)/kr$. This procedure is not recommended for PO calculations, which should use the correct near field to ensure maximum accuracy.

Links

Classes→*Electrical Objects*→*Feed*→*Tabulated Feed*→*Tabulated SWE Coefficients*

Remarks

Syntax

```
<object name> tabulated_swe_coefficients
(
    frequency                  : ref(<n>),
    coor_sys                   : ref(<n>),
    file_name                  : <f>,
    file_format                : <si>,
    power_norm                 : <si>,
    max_m_mode_index          : <i>,
    max_n_mode_index          : <i>,
    excluded_m_modes          : sequence(<i>, ...),
    power_threshold            : <r>,
    far_forced                 : <si>,
    factor                     : struct(db:<r>, deg:<r>),
    frequency_index_for_plot  : <i>,
    obsolete_coef_def          : <si>
)
```

where

<i> = integer

<n> = name of an object

<r> = real number

<f> = file name

<si> = item from a list of character strings

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the source operates. See the remarks below.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

File Name (*file_name*) [file name].

Name of file containing one or more sets of spherical wave coefficients. The contents and format of the file are described in the sections *Spherical Wave Q-Coefficients* and *Spherical Wave ab-Coefficients* and depend on the value of the attribute Coefficients Definition (Obsolete) below.

File Format (*file_format*) [item from a list of character strings], default: **TICRA**.

Format of file.

TICRA

Standard TICRA format, see Section *Spherical Wave Q-Coefficients*.

EDX

Read data in the Electromagnetic Data Exchange (EDX) format.

Power Normalisation (*power_norm*) [item from a list of character strings], default: **off**.

If on, the power contained in the SWE is equal to 4π , and the field is given in dBi.

off

The field is not normalised.

on

The power contained in the spherical wave expansion which forms the basis for the field calculation, is normalised to 4π , so that the calculated field is given relative to isotropic level.

Max m-Mode Index (*max_m_mode_index*) [integer], default: **-1**.

Maximum azimuthal mode index (m) to be included from the expansion. If negative, all azimuthal modes present in the file will be used.

Max n-Mode Index (*max_n_mode_index*) [integer], default: **-1**.

Maximum polar mode index (n) to be included from the expansion. If negative, all polar modes present in the file will be used.

Exclude m-Modes (*excluded_m_modes*) [sequence of integers], default: **-1**.

List of non-negative azimuthal mode indices, m , that shall be excluded from the expansion. Azimuthal modes with index m as well as $-m$ will be excluded. If a negative number is specified, the option is not in effect.

Power Threshold (*power_threshold*) [real number], default: **-1**.

Power threshold level given relative to the input power. Azimuthal modes with a relative power less than this threshold will be excluded in the field calculation. The azimuthal-mode power content for a given index $|m|$, is calculated by summing the power of all polar modes with index m and $-m$, see the remarks below. If a negative number is specified, no threshold is considered.

Far-Field Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on *Near-Field Calculations*):

off

Directly by the spherical wave expansion.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor (after a possible power normalisation).

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index for Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute Frequency above a sequence of frequencies (wavelengths) is listed. The Frequency Index for Plot points to the number to be applied of the frequency (wavelength) in this sequence.

Coefficients Definition (Obsolete) (*obsolete_coef_def*) [item from a list of character strings], default: **Q**.

The spherical wave coefficients of GRASP derives from two different formulations. The difference is due to the choice of the spherical vector wave functions. One is denoted the *Q*-notation, the other the *ab*-notation, with coefficients designated accordingly. The *Q*-coefficients provide higher numerical accuracy and stability, and can expand fields from larger structures than the *ab*-coefficients. See the GRASP Technical Description for a detailed documentation.

Q

The coefficients have been generated by an object of class *Spherical Wave Expansion (SWE)*, or by an external spherical wave expansion program, e.g. SWEPQ, SNIFTD, ROSCOE and CHAMP.

ab

The coefficients have been generated by an external spherical wave expansion program, e.g. SWEP (the SWEPQ predecessor). This format is obsolete.

Remarks

Spherical wave coefficients for a source may be generated by an object of class *Spherical Wave Expansion (SWE)*.

When Power Normalisation is specified to 'on', the feed pattern is normalised to dBi, i.e. to radiate 4π watts. The total radiated power will be 4π times the value of Factor (converted from dB to power).

Frequency specification

The attribute Frequency must refer to a *Frequency* object, which defines as many frequencies as there are sets of spherical wave coefficients in the file. For a field calculation at, say the third frequency, the third set of wave coefficients is employed. Note that there is no information about the frequency in the coefficient file. If the file contains only a single set of coefficients, then this set will be used at all field calculations, independent of the frequency.

Plotting

When a *Tabulated SWE Coefficients* source is visualised by *Feed Plot*, it will be drawn as a sphere of radius N_{\max}/k , which provides a convenient check on whether the scattering geometry penetrates this sphere, and hence violates the equation above. See the examples given in *Feed Plot* and in *Polygonal Struts Plot*.

Content of modes

The size of a radiating or scattering object determines the limits on the azimuthal (m) and polar (n) mode indices. For the polar modes, index n is in the range

$$1 \leq n \leq N$$

while for each polar index n , the azimuthal index m is within the range

$$-n \leq m \leq n$$

The value of N is

$$N = kr_0 + \max(10, 3.6\sqrt[3]{kr_0})$$

k being the wavenumber and r_0 the radius of the smallest sphere surrounding all sources and centred at the origin of the reference coordinate system.

The modes in a spherical wave expansion may be depicted as a distribution in the mn -plane, as shown in Figure 1 below. Each dot with indices (m,n) represents two modes, either two ab -modes, a_{mn} and b_{mn} , or two Q -modes, Q_{1mn} and Q_{2mn} , since the Q -coefficients have 3 indices, s , m , and n , where s may be 1 or 2.

A mode truncation of both azimuthal and polar modes may be employed, as shown in the following Figure 2, Figure 3 and Figure 4.

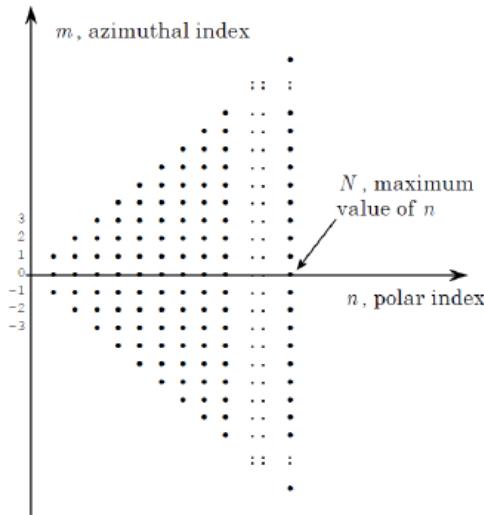


Figure 1 All mn -modes, limited by $n \leq N$, N given by the size of the radiating object.

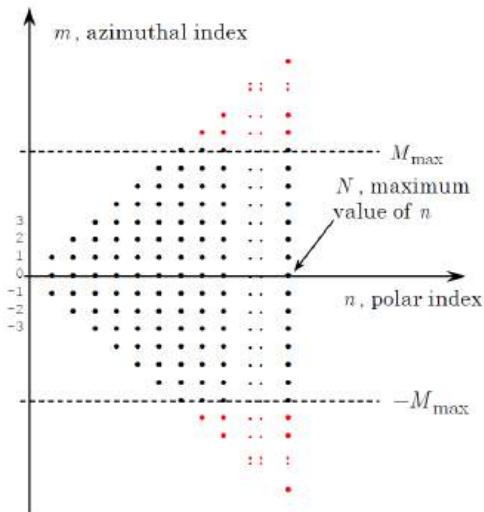


Figure 2 Max m-Mode Index is specified to M_{\max} , only the modes $-M_{\max} \leq m \leq M_{\max}$ are included (shown in black).

Finally, azimuthal modes with a relative power content less than the Power Threshold may be excluded in the field calculation. Let P_m denote the power of all modes with the same $|m|$ (the two rows for $+m$ and $-m$ in the mn -plane as in the above figures). Then all azimuthal modes for which the

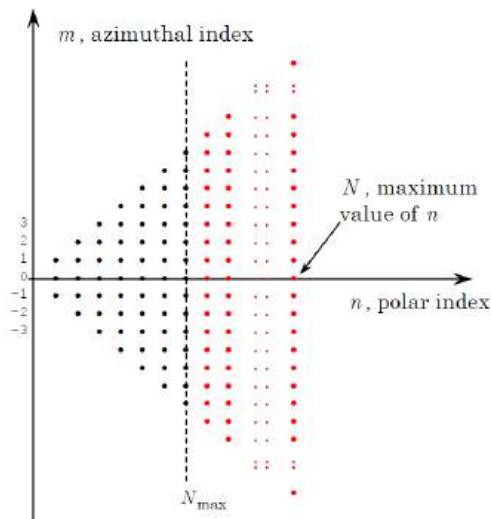


Figure 3 Max m-Mode Index is specified to N_{\max} , only the modes $n \leq N_{\max}$ are included (shown in black).

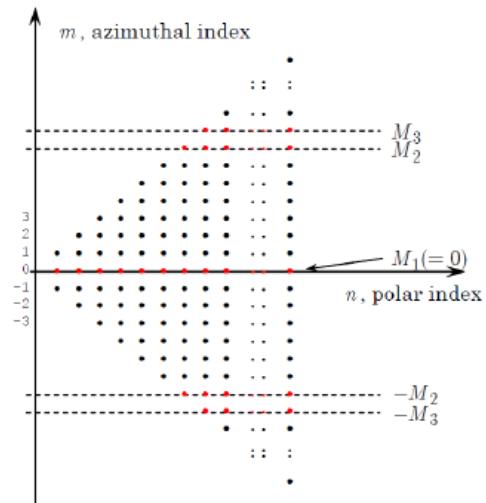


Figure 4 azimuthal_mode is specified to M_1, M_2 and M_3 . Only the modes with $|m| \neq M_1, M_2, M_3$ are included (shown in black, M_1 is in this example chosen to 0.)

relative power content P_m is below the Power Threshold

$$\frac{P_m}{\sum_{|m|} P_m} < \text{Power Threshold}$$

will be excluded.

When the set of modes has been defined, possibly after truncation and filtering, then the field may be computed from these modes at all points in space for which

$$r > N_{\max}/k$$

where N_{\max} is the highest value of the polar index n in the chosen mode set and k is the wavenumber. Depending on the sources from which the mode

set derives, the field may be determined for smaller values of r in some part of the space, but a warning will be issued.

PLANAR FEED

Purpose

This menu contains *Feeds* having a planar structure:

Microstrip Feed

Spiral Feed

Links

Classes→*Electrical Objects*→*Feed*→*Planar Feed*

MICROSTRIP FEED (microstrip_feed)

Purpose

The class *Microstrip Feed* provides a simple cavity model of a circular microstrip patch antenna.

Links

Classes→*Electrical Objects*→*Feed*→*Planar Feed*→*Microstrip Feed*

Remarks

Syntax

```
<object name> microstrip_feed
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    diameter             : <rl>,
    thickness             : <rl>,
    dielectric_constant   : <r>,
    component_along_x      : struct(amp:<r>, deg:<r>),
    component_along_y      : struct(amp:<r>, deg:<r>),
    far_forced            : <si>,
    factor                 : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>,
    gain_factor            : <r>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Diameter (*diameter*) [real number with unit of length].

Diameter of the circular patch.

Thickness (*thickness*) [real number with unit of length].

Thickness of the substrate.

Dielectric Constant (*dielectric_constant*) [real number].

Relative dielectric constant of the substrate.

Component Along x (*component_along_x*) [struct].

The relative excitation of the patch along the x_f -direction in amplitude and phase.

Amp (*amp*) [real number].

Amplitude (in linear scale) of the relative x_f excitation.

Phase in degrees (*deg*) [real number].

Phase of the relative x_f excitation, in degrees.

Component Along y (*component_along_y*) [struct].

The relative excitation of the patch along the y_f -direction in amplitude and phase.

Amp (*amp*) [real number].

Amplitude (in linear scale) of the relative y_f excitation.

Phase in degrees (*deg*) [real number].

Phase of the relative y_f excitation, in degrees.

Far Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on [Near-Field Calculations](#)):

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index For Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute *frequency* above a sequence of frequencies (wavelengths) is listed. The *frequency_index_for_plot* points to the number to be applied of the frequency (wavelength) in this sequence.

Gain Factor (Obsolete) (*gain_factor*) [real number], default: **1**.

Obsolete factor in linear scale to be multiplied on the field for simulating losses. It is recommended to specify the value to 1 and instead to apply the attribute *factor*. See also the remarks.

Remarks

The field is calculated by using a simple cavity model with a TM_{11} mode.

The radiated power is normalised to 4π watts, so that the pattern is normalised to dBi. If a *gain_factor*, V_{gn} , is specified the radiated power will be multiplied by a factor V_{gn}^2 and it will further be multiplied by the value of *factor* (converted from dB to power).

As the *Microstrip Feed* is placed in a ground plane the field only exists for $\theta \leq 90^\circ$. The spherical expansion requires the field on a full sphere, and a mirrored source radiating within $90^\circ \leq \theta \leq 180^\circ$ is added resulting in a continuous field across the ground plane. As the source itself is normalised to 4π watts the spherical expansion including the mirrored source contains coefficients representing 8π watts. The field determined for $\theta > 90^\circ$ is zero resulting in 4π watts radiated over $\theta \leq 90^\circ$.

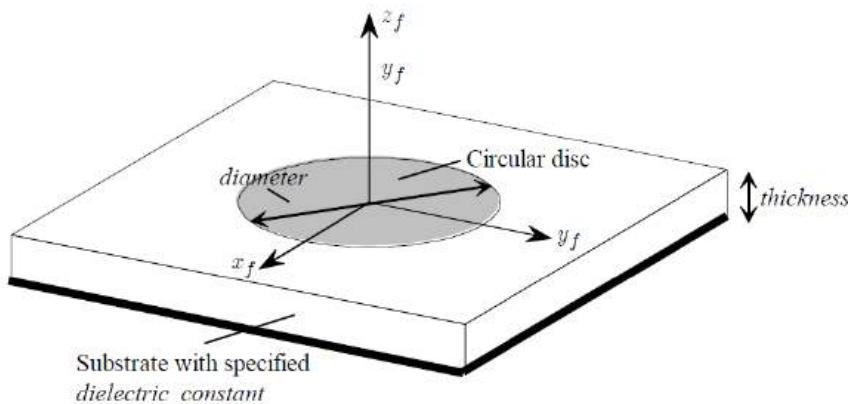


Figure 1 Circular disc microstrip antenna as modelled by the *Microstrip Feed*.

The two excitation coefficients *component_along_x* and *component_along_y* do not have any effect on the total radiated power. They only determine the relative distribution between the x_f -polarised and the y_f -polarised field components.

When *far-forced* is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

The number of azimuthal modes is due to the definition as a rotational structure limited to

$$M = 1$$

and the necessary number of ϕ -cuts over 360° is

$$N_\phi = 4$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by the size of the aperture

$$N_\theta - 1 = N = kr_0 + \max(10, 3.6\sqrt[3]{kr_0})$$

where k is the wavenumber and r_0 is the radius (half of the *diameter*) of the patch.

For a given N the field may be determined for distances larger than r_{swe}

$$r_{swe} = N/k$$

Example

An example of the pattern for a microstrip feed is shown in the following figure. The applied patch *diameter* is 0.6λ . The *thickness* is 0.02λ and the *dielectric_constant* is 2.5. The excitation is $1\angle 0^\circ$ along x and $0.5\angle -90^\circ$ along y. The *gain_factor* is 1.

For this case

$$N_\phi = 4$$

and

$$N_\theta - 1 = N = 12$$

Thus $r_{swe} = 1.9\lambda$. The near field will be determined at a distance of 2λ .

The correct near field is determined when *far_forced* is specified to 'off' and is shown as full-line patterns. When *far_forced* is specified to 'on' the near field is determined as the far field multiplied with the distance factor; these patterns are shown as dotted lines.

The near-field effects are small as the feed is small in terms of wavelengths. On the other hand the near-field effects are large taking into account that the near field is determined at a distance which is about three times the traditional far-field distance

$$r_{ff} = 2D^2/\lambda = 0.72\lambda$$

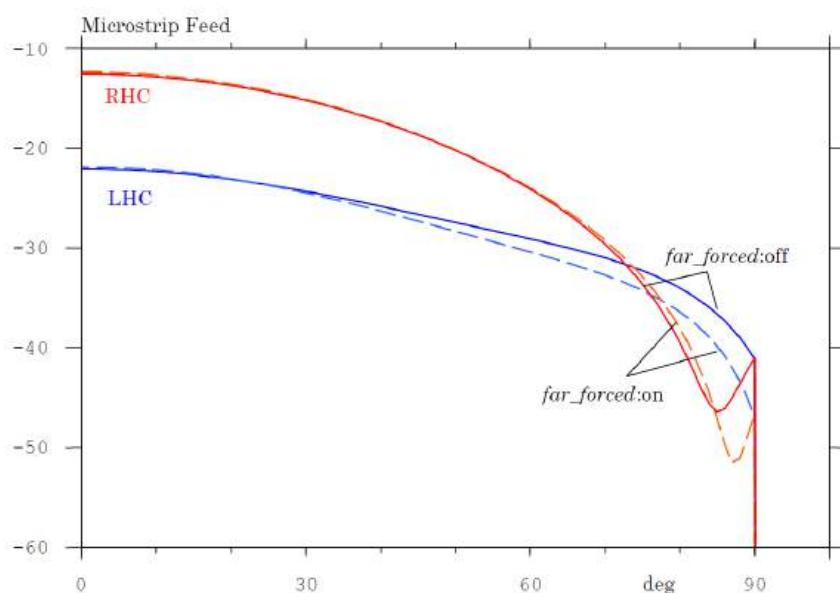


Figure 2

Near-field patterns, RHC and LHC at $\phi = 45^\circ$, for a **Microstrip Feed** of diameter 0.6λ . For patterns in full lines *far_forced* is specified to 'off'. For dashed patterns *far_forced* is specified to 'on' and these patterns are proportional to the far-field patterns (not shown).

SPIRAL FEED (spiral_feed)

Purpose

The class *Spiral Feed* approximates the radiation from a bifilar, planar equi-angular spiral antenna.

Links

Classes→*Electrical Objects*→*Feed*→*Planar Feed*→*Spiral Feed*

Remarks

Syntax

```
<object name> spiral_feed
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    external_radius     : <rl>,
    internal_radius     : <rl>,
    spiral_exponent    : <r>,
    polarisation        : <si>,
    height_over_ground  : <rl>,
    evaluation          : <si>,
    far_forced          : <si>,
    factor              : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>,
    loss                : <r>
)
```

where

<i> = integer

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

<si> = item from a list of character strings

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

External Radius (*external_radius*) [real number with unit of length].

External (maximum) radius of the spiral.

Internal Radius (*internal_radius*) [real number with unit of length].

Internal (minimum) radius of the spiral.

Spiral Exponent (*spiral_exponent*) [real number].

Determines the angle of the spiral windings (see the remarks below).
Must be positive.

Polarisation (*polarisation*) [item from a list of character strings], default: **rhc**.

Sense of the circular polarisation of the field in the main direction (along z_f).

rhc

The polarisation is right hand circular.

lhc

The polarisation is left hand circular.

Height Over Ground (*height_over_ground*) [real number with unit of length], default: **-1**.

Height of the spiral above a ground plane. If $height_over_ground \leq 0$, the ground plane is neglected.

Evaluation (*evaluation*) [item from a list of character strings], default: **asymptotic_with_end**.

Determines the method for evaluation of the radiation integral. See the Technical Description for details. An asymptotic evaluation is the fastest method, but less accurate than a numerical evaluation.

asymptotic_with_end

Asymptotic evaluation with saddle-point contribution (if any) and two end-point contributions.

numerical

More time consuming, but also more accurate.

asymptotic

Asymptotic evaluation with saddle-point contribution (if any), but without end-point contributions.

Far Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on [Near-Field Calculations](#)).

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (factor) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index For Plot (frequency_index_for_plot) [integer], default: 1.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute *frequency* above a sequence of frequencies (wavelengths) is listed. The *frequency_index_for_plot* points to the number to be applied of the frequency (wavelength) in this sequence.

Loss (Obsolete) (*loss*) [real number], default: 0.

Obsolete attribute which ought to be specified to 0 whereby the feed will be normalised to radiate 4π watts. The pattern may be multiplied by a factor (by specifying this in the attribute *factor*) representing a loss if desired, in dB.

Remarks

Each branch of the spiral is defined in polar coordinates (r_f, ϕ_f) in the $x_f y_f$ -plane of the feed coordinate system by the equation

$$r_f(\phi_f) = r_1 \cdot e^{\pm a\phi_f}$$

where r_1 is the *internal_radius* and a is the *spiral_exponent*. The sign in the exponential function depends on the chosen sense of the circular *polarisation*, plus for right hand polarisation, minus for left hand polarisation.

The spiral angle, given by the angle between radius vector and the tangent is given by ψ ,

$$\psi = \arctan \frac{1}{a}$$

The phase reference is the origin of the feed coordinate system for which the x_f, y_f -plane coincides with the ground plane and the z_f -axis passes through the centre of the spiral. The pattern is normalised to dBi.

The radiated power is normalised to 4π watts, so that the pattern is normalised to dBi. If a *loss* is specified the radiated power will be reduced with that amount and it will further be multiplied by the value of *factor* (converted from dB to power).

If the **Spiral Feed** is placed above a ground plane the far field only exists for $\theta \leq 90^\circ$. As the spherical expansion requires the field on a full sphere,

a mirrored source radiating within $90 \leq \theta \leq 180^\circ$ is added resulting in a continuous field across the ground plane. The source itself is normalised to 4π watts and the spherical expansion including the mirrored source contains coefficients representing 8π watts. The field determined for $\theta > 90^\circ$ is zero resulting in 4π watts radiated over $\theta \leq 90^\circ$.

When *far-forced* is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

Due to the rotational symmetry of the feed the number of azimuthal modes is limited to

$$M = 1$$

and the necessary number of ϕ -cuts over 360° is

$$N_\phi = 4$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by the size of the aperture

$$N_\theta - 1 = N = kr_0 + \max(10, 3.6\sqrt[3]{kr_0})$$

where k is the wavenumber and r_0 is the radius of the smallest sphere centred at the origin of the feed coordinate system and enclosing the *Spiral Feed*.

For a given N the field may be determined for distances larger than r_{swe}

$$r_{swe} = N/k$$

HELIX OR DIPOLE

Purpose

This menu contains helix and dipole *Feed*s:

Helix

Half Wave Dipole

Hertzian Dipole

Links

Classes→*Electrical Objects*→*Feed*→*Helix or Dipole*

HELIX (helix)

Purpose

The class **Helix** defines a feed model, which approximates the radiation from a monofilar helix antenna radiating in the axial mode.

Links

Classes→*Electrical Objects*→*Feed*→*Helix or Dipole*→*Helix*

Remarks

Syntax

```
<object name> helix
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    diameter            : <rl>,
    pitch_angle         : <r>,
    turns               : <r>,
    polarisation        : <si>,
    far_forced          : <si>,
    factor              : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>,
    loss                : <r>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a **Frequency** object, defining the frequencies at which the antenna operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the **Coordinate Systems** classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Diameter (*diameter*) [real number with unit of length].

Diameter, D , of the helix. The axial mode of radiation restricts the helix circumference to be in the range: $0.8\lambda \leq \pi D \leq 1.2\lambda$.

Pitch Angle (*pitch_angle*) [real number].

Pitch angle of helix in degrees. Must be positive.

Turns (*turns*) [real number].

Number of turns. Must be positive.

Polarisation (*polarisation*) [item from a list of character strings], default: **rhc**.

Definition of the main polarisation of the helix. This attribute determines how the helix is wound, so that right-hand polarisation is produced by a right-handed helix.

rhc

Right-hand circular polarisation.

lhc

Left-hand circular polarisation.

Far Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on *Near-Field Calculations*):

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index For Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute *frequency* above a sequence of frequencies (wavelengths) is listed. The *frequency_index_for_plot* points to the number to be applied of the frequency (wavelength) in this sequence.

Loss (Obsolete) (*loss*) [real number], default: **0**.

Obsolete attribute which ought to be specified to 0 dB whereby also this feed will be normalised to radiate 4π watts. The *Helix* pattern may be multiplied by a factor (by specifying this in the attribute *factor*) representing a loss if desired, in dB.

Remarks

The centre of the helix is the phase reference point. The axis of the helix is parallel to the z_f -axis of the feed coordinate system, and the helix will radiate in the positive z_f -direction. The parameters of the helix are illustrated in the following figure.

The radiated power is normalised to 4π watts, so that the pattern is normalised to dBi. If a *loss* is specified the radiated power will be reduced with that amount and it will further be multiplied by the value of *factor* (converted from dB to power).

When *far-forced* is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

Due to the rotational symmetry of the feed the number of azimuthal modes is limited to

$$M = 1$$

and the necessary number of ϕ -cuts over 360° is

$$N_\phi = 4$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by the size of the helix

$$N_\theta - 1 = N = kr_0 + \max(10, 3.6\sqrt[3]{kr_0})$$

where k is the wavenumber and r_0 is the radius of the smallest sphere centred at the origin of the feed coordinate system and enclosing the helix.

For a given N the field may be determined for distances larger than r_{swe}

$$r_{swe} = N/k$$

Example

An example of the radiation pattern of a helix is given in the last figure. The helix has 5 turns with a pitch angle of 12.8° and diameter 4.23 inches at 925 MHz (the diameter is then 0.3315λ). The default parameters are applied for the remaining attributes.

The height of this helix is

$$h = 5 \cdot 0.3315\lambda\pi \tan 12.8^\circ = 0.377\lambda$$

Then

$$r_0 = \sqrt{(h/2)^2 + (d/2)^2} = 0.251\lambda$$

d being the diameter of the helix. This results in

$$N_\theta - 1 = N = 12$$

and

$$r_{swe} = N/k = 1.9\lambda$$

The near field is determined at a distance of 2λ .

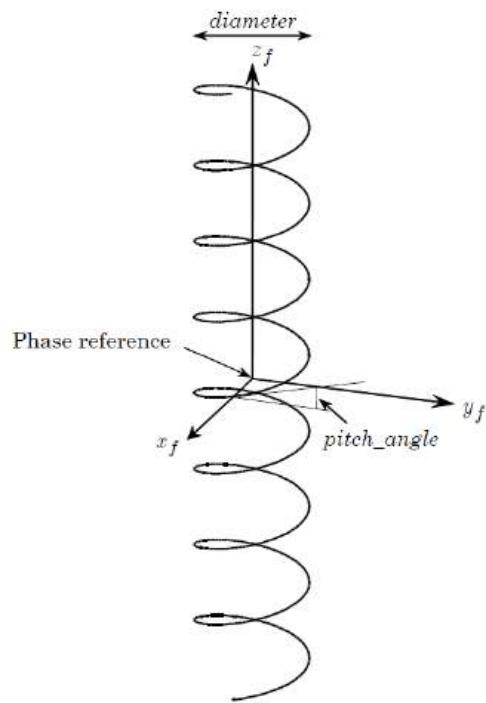


Figure 1 Monofilar helix with 8 turns

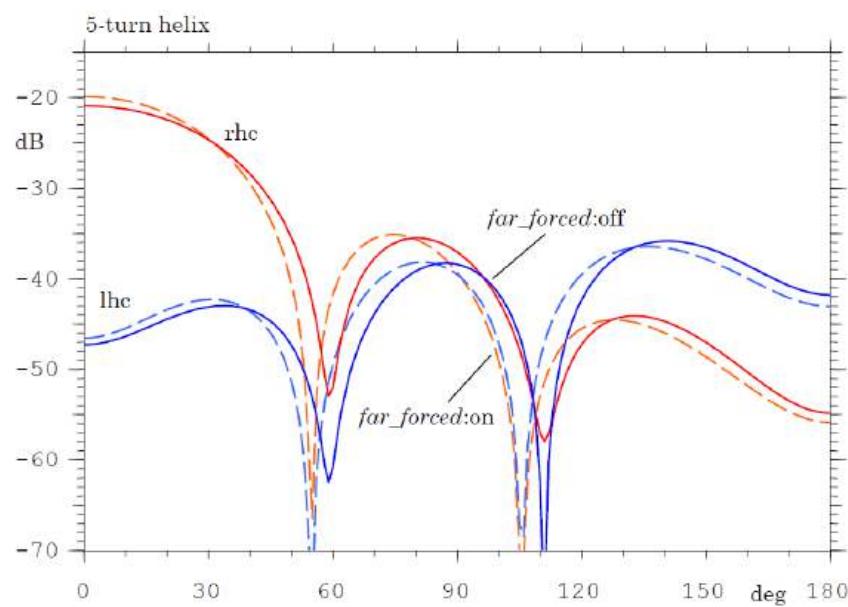


Figure 2 Near field patterns (linear polarisation according to Ludwig's 3rd definition) at $\phi = 0^\circ$ for a 5 turn helix with a pitch angle of 12.8° and diameter 0.3315λ . For patterns in full lines *far_forced* is specified to 'off'. For dashed patterns *far_forced* is specified to 'on' and these patterns are proportional to the far-field patterns (not shown).

HALF WAVE DIPOLE (half_wave_dipole)

Purpose

The class **Half Wave Dipole** defines the radiation from a dipole half a wavelength long.

Links

[Classes](#)→[Electrical Objects](#)→[Feed](#)→[Helix or Dipole](#)→[Half Wave Dipole](#)

Remarks

Syntax

```
<object name> half_wave_dipole
(
    frequency                  : ref(<n>),
    coor_sys                   : ref(<n>),
    far_forced                 : <si>,
    factor                     : struct(db:<r>, deg:<r>),
    frequency_index_for_plot   : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a **Frequency** object, defining the frequencies at which the dipole operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the **Coordinate Systems** classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Far Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on **Near-Field Calculations**):

off

As a spherical wave expansion derived from the far-field pattern.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index For Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute *frequency* above a sequence of frequencies (wavelengths) is listed. The *frequency_index_for_plot* points to the number to be applied of the frequency (wavelength) in this sequence.

Remarks

The half wave dipole is oriented with its axis along the z_f -axis of the feed coordinate system.

The feed pattern is normalised to dBi, i.e. to radiate 4π watts. The total radiated power will be 4π times the value of *factor* (converted from dB to power).

When *far-forced* is 'off' the feed is modelled by a spherical expansion based on a sampling of the far field. The minimum sampling is determined as follows.

Due to the rotational symmetry of the feed the number of azimuthal modes is limited to

$$M = 1$$

and the necessary number of ϕ -cuts over 360° is

$$N_\phi = 4$$

The minimum number of samples in θ is N_θ from 0° to 180° , inclusive, and $N_\theta - 1$ is equal to the maximum value of the polar spherical mode, N , given by the size of the feed

$$N_\theta - 1 = N = kr_0 + \max(10, 3.6\sqrt[3]{kr_0})$$

where k is the wavenumber and r_0 is the radius of the smallest sphere enclosing the dipole, $r_0 = \lambda/4$, resulting in

$$N_\theta - 1 = N = 11$$

(in this case the expression for N may be truncated to the nearest lower integer).

HERTZIAN DIPOLE (hertzian_dipole)

Purpose

The class *Hertzian Dipole* defines the radiation from an electric or magnetic Hertzian dipole (elementary dipole).

Links

Classes→*Electrical Objects*→*Feed*→*Helix or Dipole*→*Hertzian Dipole*

Remarks

Syntax

```
<object name> hertzian_dipole
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    dipole_type         : <si>,
    orientation         : <si>,
    plot_length         : <rl>,
    far_forced          : <si>,
    factor              : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the dipole operates.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The feed is located in the origin of this coordinate system (denoted x_f, y_f, z_f) and the feed axis points along the z_f -axis.

Dipole Type (*dipole_type*) [item from a list of character strings], default: **electric**.

The dipole may be either an electric or a magnetic elementary dipole.
electric

The dipole is an electric Hertzian dipole.

magnetic

The dipole is a magnetic Hertzian dipole.

Orientation (*orientation*) [item from a list of character strings], default: **z**.

The Hertzian dipole may be oriented with its dipole moment along either the positive x_f , y_f - or z_f -axis of the feed coordinate system.

z

The dipole is oriented along the positive z_f -axis.

y

The dipole is oriented along the positive y_f -axis.

x

The dipole is oriented along the positive x_f -axis.

Plot Length (*plot_length*) [real number with unit of length], default: **0**.

Since the Hertzian dipole is a theoretical quantity with no length, a Plot Length can be specified for drawing purposes. The length is from the centre to the top end of the dipole. If the value is 0, the length used in the drawing is a quarter of a wavelength. A negative length is not permitted.

Far Forced (*far_forced*) [item from a list of character strings], default: **off**.

Determines how a near field will be calculated (for details, see the section on *Near-Field Calculations*):

off

As the near field inherent in the model.

on

The far-field pattern is multiplied by a distance factor.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index For Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute *frequency* above a sequence of frequencies (wavelengths) is listed. The *frequency_index_for_plot* points to the number to be applied of the frequency (wavelength) in this sequence.

Remarks

The feed pattern is normalised to dBi, i.e. to radiate 4π watts. The total radiated power will be 4π times the value of *factor* (converted from dB to power).

The pattern is not expanded into spherical waves as the model includes a precise near-field description. Nevertheless, a minimum sphere may be plotted when requested in *Feed Plot*. The radius of this minimum sphere is $9/k = 1.43\lambda$.

Also a Rayleigh sphere may be drawn on request though a Rayleigh distance cannot be given. The radius of the drawn Rayleigh sphere is 2λ .

If *factor* is 0 dB then the far-field pattern of the Hertzian dipole is given by

$$\vec{E}_{far} = \sqrt{\frac{3}{2}} \sin \theta \hat{\theta}$$

PO ANALYSIS

Purpose

Physical Optics (PO) calculations may be carried out by one of the classes under *PO Analysis*.

In the classes under *PO Analysis* it is possible to describe how the PO and PTD currents shall be calculated on a specific *Scatterer*. The currents originate in the field from a specified Source and the calculated values are stored in the objects under *PO Analysis*. These currents can serve as a source in later calculations either for determining a field or for determining new induced currents on another scatterer.

The classes available under *PO Analysis* are the following:

PO, Single-Face Scatterer (for *Reflectors* and *Plates*)

PO, Multi-Face Scatterer (for any *Scatterer*)

PO, Panels in Polar Grid (for a *Reflector with Panels*)

QUAST add-on: In the QUAST add-on, the following additional PO-classes are available:

PO, Aperture in Screen (for *Aperture in Screen*)

PO, Lens (for *Simple Lens*)

The number of current elements must be specified such that the integration of the currents converges. This may be ensured by applying the auto-convergence feature of the command *Get Currents* which takes into account the illumination of the scatterer and the position of the requested points for the field calculation (or the current calculation upon another scatterer).

The class

PO Convergence (Obsolete)

is obsolete and is only included for compatibility reasons. Apply *Get Currents* for automatic determination of the integration grid.

Links

Classes→*Electrical Objects*→*PO Analysis*

PO, SINGLE-FACE SCATTERER (po_single_face_scatterer)

Purpose

The *PO, Single-Face Scatterer* class is used to specify how the PO and PTD currents are calculated on *Reflectors* and *Plates*. The currents are stored in the object, and can serve as a source in later calculations.

Links

Classes→*Electrical Objects*→*PO Analysis*→*PO, Single-Face Scatterer*

Remarks

Syntax

```
<object name> po_single_face_scatterer
(
    frequency           : ref(<n>),
    scatterer          : ref(<n>),
    method              : <si>,
    po_points          : struct(po1:<i>, po2:<i>),
    ptd_points         : sequence(
                            struct(edge:<i>,
                                   ptd:<i>),
                            ...),
    factor              : struct(db:<r>, deg:<r>),
    spill_over          : <si>,
    ray_output          : <si>,
    coor_sys            : ref(<n>),
    file_name           : <f>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the currents are calculated. The radiated field from the currents can only be calculated at these frequencies. Further, the *Frequency* class must agree with the *Frequency* class specified in the source object that generates the actual PO or PTD currents.

Scatterer (*scatterer*) [name of an object].

Reference to an object of one of the classes *Reflectors* and *Plates* on which the PO and PTD currents are computed.

Method (*method*) [item from a list of character strings], default: **po_plus_ptd**.

Determines if the PO and/or PTD contributions are included in the calculations (see the remarks below).

po_plus_ptd

PO as well as PTD contributions are included.

po

Only the PO contribution is included.

ptd

Only the PTD contribution is included.

PO Points (*po_points*) [struct].

The density of the PO integration grid is defined by the members **po1** and **po2**.

po1 (*po1*) [integer], default: **0**.

Number of current samples in first grid coordinate (see the remarks below for details).

po2 (*po2*) [integer], default: **0**.

Number of current samples in second grid coordinate (see the remarks below for details).

PTD Points (*ptd_points*) [sequence of structs].

A sequence of edges of the scatterer is specified on which the PTD currents are calculated.

Edge Number (*edge*) [integer], default: **-1**.

Number of an edge along which the PTD currents shall be calculated. If specified to -1 then all edges, including possible holes in the reflector, are included. If contributions are requested from specific edges these shall be specified individually. For the numbering of the edges the user is referred to the description of the individual *Scatterer*.

ptd (*ptd*) [integer], default: **0**.

Number of samples of the current along the edge (see the remarks below). If *edge* is specified as -1, the number applies to the entire length of all edges. The elements are distributed according to the length of the individual edges in order to obtain a nearly uniform distribution of the samples.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor:

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Spill-Over (*spill_over*) [item from a list of character strings], default: **off**.

Determines if spillover shall be calculated.

off

The spillover is not calculated.

on

The total power W incident on the scatterer is calculated and the spillover is computed as $4\pi/W$. Thus, the result reflects only the true spillover when the feed is power normalised to 4π (see the remarks below). The spillover calculation can significantly increase the computation time if the incident field comes from another PO source because both the incident E- and H-field are required in order to compute Poynting's vector.

Ray Output (*ray_output*) [item from a list of character strings], default: **none**.

This attribute defines how the field of the currents shall be calculated if the field is used as input to a GTD or PTD analysis. The attribute is also actual for a subsequent PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case, Ray Output has the same meaning as stated below for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that the field cannot be used as input to a GTD analysis. In a subsequent PTD analysis the direction of propagation is assumed to be the direction of Poynting's vector at the field point.

all

The field is computed as a ray field with one ray from each current element. A subsequent GTD or PTD analysis is then carried out for each ray separately. This is the most accurate method for GTD/PTD analysis, but it may be time consuming.

spherical

The field from the elements will be treated as a single ray which emanates from the origin of the coordinate system specified by Coordinate System. This procedure is only accurate if the scatterer in the following GTD/PTD analysis is in the far field from the currents or the currents are phased such that they radiate a field at the scatterer with a spherical phase front centred at the origin.

plane

The field from the currents will be treated as a single ray which is parallel to the z -axis of the coordinate system specified by Coordinate System. This procedure is only accurate if the currents radiate a field in the direction of the z -axis of the coordinate system Coordinate System and if this field has a nearly plane phase front at the scatterer to be analysed by GTD/PTD.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system of this PO source. The currents are calculated in this coordinate system.

File Name (*file_name*) [file name].

Name of file in which the PO and PTD currents are stored. When a File Name is not given, the currents are stored in an internal file, which is deleted at the end of the session. The recommended file extension is .cur, cf. *Standard Currents File*.

Remarks

This section contains remarks on the following topics:

- Handling of the PO and PTD contributions
- The number of integration points
- Estimates of the parameters *po1*, *po2* and *ptd*
- Spillover

In case the current elements are applied for determination of near fields, the number of required current elements necessary to obtain convergence may be very large, causing a long computation time. In such case it may be advantageous first to expand the field from the current elements into a plane-wave spectrum using the command *Get Plane Wave Expansion* and subsequently using the determined spectrum for calculating the required near fields.

Handling of the PO and PTD contributions

When the PO object is used as the target in a *Get Currents* command, the attribute Method selects the contributions to be calculated and stored in the currents file:

- If Method is specified to 'po' or 'po_plus_ptd', then the PO contribution is calculated. If auto-convergence is disabled (in the *Get Currents* command), then the PO calculation is performed with the number of PO points specified in the attribute PO Points. If auto-convergence is enabled, then the PO auto-convergence calculation is performed independent of the actually specified number of PO points.

- If Method is specified to ‘po_plus_ptd’ or ‘ptd’, then the PTD contribution is calculated. If auto-convergence is disabled, then the PTD calculation is performed for the edges specified in the attribute PTD Points with the number of PTD points specified in the same attribute. If auto-convergence is enabled, then the PTD auto-convergence calculation is performed for the edges specified in the attribute PTD Points, independent of the actually specified number of PTD points.

When the PO object is used as the source, e.g. in a *Get Field* command, the attribute Method selects those contributions from the currents file to be included in the numerical integration:

- If Method is specified to ‘po’, then only the PO contribution is calculated. The PO calculation is performed with the number of PO points available in the file, independent of the actually specified number of PO points in the attribute PO Points. An error is issued if the PO currents are not found in the file. A warning is issued if the file also contains PTD elements.
- If Method is specified to ‘ptd’, then only the PTD contribution is calculated. The PTD calculation is only performed for the edges specified in the attribute PTD Points, even if PTD elements for additional edges exist in the file. The PTD calculation is performed with the number of PTD points available in the file, independent of the actually specified number of PTD points in the attribute PTD Points. An error is issued if one of the specified edges is not found in the file. A warning is issued if the file also contains PO currents.
- If Method is specified to ‘po_plus_ptd’, then the PO and the PTD contributions are calculated. The PTD calculation is only performed for the edges specified in the PTD Points, even if PTD elements for additional edges exist in the file. The calculation is performed with the number of PO and PTD points available in the file, independent of the actually specified number of PO and PTD points in the attributes PO Points and PTD Points. An error is issued if the PO currents are not found in the file or if one of the specified edges is not found in the file.

The attribute Method thus yields a simple way of extracting the individual scattering contributions to the total field, as exemplified in the following:

First, calculate the total field by a *Get Currents* command followed by a *Get Field* command. Auto-convergence should be enabled in the *Get Currents* command, and the attribute Method in the PO object set to ‘po_plus_ptd’ to include all scattering effects. Further, the attributes PO Points and PTD Points should have their default settings, selecting all edges of the scatterer. Also, a file name must be specified in the attribute File Name, so that the currents calculated in this task are stored in the currents file.

Next, calculate only the PO contribution to the total field. This is achieved by changing the attribute Method in the PO object to ‘po’. The PO contribution could be calculated by re-running the *Get Currents* and *Get Field* commands. However, the same result is obtained by only re-running the *Get Field* command, as the PO currents are simply extracted from the currents file.

Finally, calculate the PTD contribution, by changing Method to 'ptd', and re-running the *Get Field* command.

The three results (PO+PTD, PO, PTD) were obtained by executing a single and not three *Get Currents* commands, thus significantly reducing the computation time. It is also possible to extract the contribution from a selected number of edges by specifying these edges in the attribute PTD Points. For example to calculate the contribution from edge 2 alone:

Calculate the PTD contribution from edge 2 alone, by selecting method to 'ptd', specifying edge to 2 in the PTD Points attribute, and re-running the *Get Field* command.

The number of integration points

The parameters:

- po1 and po2 of the attribute PO Points, and
- ptd in the attribute PTD Points

specify numbers of integration points in the PO or PTD integration grids. If one of these parameters is specified to zero then the corresponding contribution is not included in the analysis. The values of po1, po2 and ptd may be specified here, or the values needed for a specified accuracy may be determined automatically as specified in the *Get Currents* command.

Estimates of the parameters po1, po2 and ptd

It is essential that the number of integration points is so large that the integration will be correct. On the other hand, a too high value may cause an unacceptable long processing time. It is therefore necessary to perform a convergence test in which the parameters po1 and po2 are increased until the field calculation is independent on the parameters. This is easiest done by the auto convergence feature of the command *Get Currents* but when the user wants to control the convergence this may be performed by switching the auto convergence off in the command *Get Currents*. In that case po1, po2 and ptd shall here be specified to non-zero values (in order not to skip the calculations). The necessary values of the parameters po1 and po2 depend on the type of scatterer that is analysed as well as on the position of the field points. Estimated values will be given in the following. These may be applied as starting values when the user wants to control the convergence accuracy.

For the simple case of a reflector with an *Elliptical Rim* (possibly with a central hole) the integration grid is a polar grid when projected on the *xy*-plane of the reflector coordinate system. The centre of this grid is coincident with the centre of the rim. po1 gives the number of integration points in radial direction (along ρ) and po2 gives the number of points in azimuthal direction (along ϕ). If the rim is circular and if the aperture of the reflector is focused, the number of points necessary for convergence can be estimated by

$$\begin{aligned} \text{po1} &= \text{nint}(z/2.4) \\ \text{po2} &= \text{nint}(z) \end{aligned}$$

where

$$z = 1.09 \cdot \pi \frac{D}{\lambda} \sin \theta_0 + 10$$

The function `nint` means the nearest integer to the argument, and D is the reflector diameter, λ is the wavelength and θ_0 is the output angle from the beam centre out to which the integration should converge. For worst-case illumination and observation angles it may be necessary to increase z to $z = 2\pi D/\lambda$.

The estimate `po2` can also be used for the number `ptd` of PTD points along the outer rim.

If the scatterer is a *Reflector* without holes and the rim is a *Rectangular Rim*, then `po1` and `po2` give the number of integration points along x and y , respectively, of the reflector coordinate system. For a focused aperture the integration will converge if

$$\begin{aligned} \text{po1} &= \text{nint} \left(1.75 \frac{D_x}{\lambda} \sin \theta_x + 10 \right) \\ \text{po2} &= \text{nint} \left(1.75 \frac{D_y}{\lambda} \sin \theta_y + 10 \right) \end{aligned}$$

where D_x and D_y are the reflector dimensions along x and y , respectively (for the non-rotated rim). The PO integral will converge from the beam centre out to the angle θ_x towards the x -axis and out to the angle θ_y towards the y -axis. If the reflector is not focused values of `po1` and `po2` up to

$$\begin{aligned} \text{po1} &= 3.5 \frac{D_x}{\lambda} \\ \text{po2} &= 3.5 \frac{D_y}{\lambda} \end{aligned} \tag{1}$$

may be necessary.

These estimates can also be used for the number, `ptd`, of PTD points in the PTD integration. This means that for the edges parallel to the x -axis, `ptd` shall have the same value as `po1`, and for the edges parallel to the y -axis the `ptd` value shall be the same as for `po2`.

If the scatterer is a *Reflector* without holes and the rim is a *Triangular Rim*, `po2` signifies the number of integration points along the longest edge and `po1` the number of points along the height perpendicular to that edge and again the expressions above may be applied.

Also, when the scatterer is a *Triangular Plate*, `po2` signifies the number of integration points along the longest edge and `po1` the number of points along the height perpendicular to that edge. Since a plane will not in general focus the field, it is necessary with a fine integration grid in order to ensure convergence. Usually, `po1` and `po2` should be between $2D/\lambda$ and $4D/\lambda$, where D is the corresponding height, D_h , or edge length, D_e , respectively. If the height and the side length differ considerably, it may be necessary to increase `po1` above $4D_h/\lambda$; however, `po1` should never need to be greater than `po2`. The number of PTD samples, `ptd`, shall be at least `po2` along the longest edge. The number of PTD samples along the other two edges

shall result in a sampling density being approximately the same as along the longest edge.

If the scatterer is a plane *Parallelogram*, *po1* signifies the number of integration points along *vec_1*, and *po2* the number of integration points along *vec_2*, where *vec_1* and *vec_2* are attributes of the class *Parallelogram*. Since a plane will not in general focus the field, it is necessary with a fine integration grid in order to ensure convergence. Usually, *po1* and *po2* should be between $2D/\lambda$ and $4D/\lambda$, where *D* is the corresponding side length, i.e. the length of *vec_1* and *vec_2*, respectively.

If the scatterer is a plane *Rectangular Plate*, *po1* signifies the number of integration points along the edge from *corner_1* to *corner_2* and *po2* the number of points along the perpendicular edge. *corner_1* and *corner_2* are attributes of the class *Rectangular Plate*. Again, since a plane will not in general focus the field, it is necessary with a fine integration grid in order to ensure convergence. Therefore, *po1* and *po2* should usually be between $2D/\lambda$ and $4D/\lambda$, where *D* is the corresponding side length.

The number of PTD samples should follow the values of *po1* and *po2*.

For the general *Reflector* with a hole and not included in those mentioned above, an **advanced quadrilinear mesh** is used. Such cases include 1) a *Reflector* with central hole and with rims of types *Rectangular Rim* or *Triangular Rim*, 2) a *Reflector* with a hole defined by the *holes* attribute and 3) a *Reflector* with more than one hole . For such a mesh *po1* and *po2* indicate the number of integration points along the two sets of opposite edges of each patch of the quadrilinear mesh. As the size of the patches are small compared to the full reflector then *po1* and *po2* are correspondingly small compared to the *po1*- and *po2*-values for a similar reflector without hole(s). No estimates can be provided for *po1*, *po2*, and *ptd*, and these values should always be determined by using the auto convergence feature of the *Get Currents* command.

Spillover

In GRASP the spillover in dB is defined by

$$10 \log\left(\frac{4\pi}{W}\right)$$

where *W* is the total power hitting the scatterer. In the GRASP output file, the spillover is printed as well as the relative power hitting the scatterer, *W/4π*. See also the Technical Description for further information of PO convergence and spillover.

PO, MULTI-FACE SCATTERER (po_multi_face_scatterer)

Purpose

The *PO, Multi-Face Scatterer* class is used to specify how the PO and PTD currents shall be calculated on a specific *Scatterer* with one or more faces. By face is here understood a specific part of a *Scatterer* such as a facet of a strut or a panel of a reflector. The currents are stored in the object, and can serve as a source in later calculations.

PO, Multi-Face Scatterer can be applied to all scatterers (except *Circular Struts*, *Wires* and *DGR Intercostals*) as opposed to the other classes under *PO Analysis*, which can only be used for restricted classes of scatterers.

Links

Classes→*Electrical Objects*→*PO Analysis*→*PO, Multi-Face Scatterer*

Remarks

Syntax

```

<object name> po_multi_face_scatterer
(
    frequency : ref(<n>),
    scatterer : ref(<n>),
    method : <si>,
    po_points : sequence(
        struct(
            start_face:<i>,
            end_face:<i>,
            po1:<i>,
            po2:<i>),
        ...),
    ptd_contributions : <si>,
    edge_points : sequence(
        struct(
            start_edge:<i>,
            end_edge:<i>,
            ptd:<i>),
        ...),
    gap_points : sequence(
        struct(
            start_gap:<i>,
            end_gap:<i>,
            ptd1:<i>,
            ptd2:<i>,
            gap_method:<si>),
        ...),
    factor : struct(db:<r>, deg:<r>),
    spill_over : <si>,
    ray_output : <si>,
    coor_sys : ref(<n>),
    file_name : <f>
)

```

)
 where
 <i> = integer
 <n> = name of an object
 <r> = real number
 <f> = file name
 <si> = item from a list of character strings

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the currents are calculated. Further, the *Frequency* class must agree with the *Frequency* class specified in the source object that generates the actual PO or PTD currents.

Scatterer (*scatterer*) [name of an object].

Reference to an object of one of the *Scatterer* classes (except *Circular Struts*, *Wires* and *DGR Intercostals*) on which the PO and PTD currents is computed.

Method (*method*) [item from a list of character strings], default: **po_plus_ptd**.

Determines if the PO and/or PTD contributions are included in the calculations (see the remarks below).

po_plus_ptd

PO as well as PTD contributions are included.

po

Only the PO contribution is included.

ptd

Only the PTD contribution is included.

PO points (*po_points*) [sequence of structs].

Defines the PO integration grids used on the faces of the scatterer. The PO integration grids must be defined for all the illuminated faces of the scatterer. If two or more successive faces of the scatterer have identical PO integration grids, the grids may be specified for all these faces by defining an interval of faces by the members *start_face* and *end_face*. If there are no such identical PO grids, *po_points* must contain the same number of elements as there are illuminated faces. For each face contained in the interval of faces the density of the PO integration grid is defined by the members *po1* and *po2*:

Start Face (*start_face*) [integer], default: **-1**.

Number of the first face in the interval of faces. Must be positive or -1 (see the remarks below).

End Face (*end_face*) [integer], default: **-1**.

Number of the last face in the interval of faces. Must be positive or -1 (see the remarks below).

po1 (*po1*) [integer], default: **0**.

For each face in the interval of faces, the number of current samples in first grid coordinate. Must be positive or 0 (see the remarks below).

po2 (*po2*) [integer], default: **0**.

For each face in the interval of faces, the number of current samples in second grid coordinate. Must be positive or 0 (see the remarks below).

PTD Contributions (*ptd_contributions*) [item from a list of character strings], default: **all**.

The type of PTD contributions to be included in the calculations (see the remarks below).

all

All PTD contributions from edges and gaps are included.

edges

Only the PTD contribution from edges is included.

gaps

Only the PTD contribution from gaps is included.

Edge Points (*edge_points*) [sequence of structs].

A sequence of edges on the scatterer is specified on which the PTD elements shall be calculated. If two or more successive edges of the scatterer have identical number of PTD elements, the number of points may be specified for all these edges by defining an interval of edges by the members *start_edge* and *end_edge*. If there are no such edges, *edge_points* must contain the same number of elements as there are illuminated edges. For each edge contained in the interval of edges the number of PTD elements is defined by the member *ptd*.

Start Edge (*start_edge*) [integer], default: **-1**.

Number of the first edge in the interval of edges. Must be positive or specified to -1 (see the remarks below).

End Edge (*end_edge*) [integer], default: **-1**.

Number of the last edge in the interval of edges. Must be positive or specified to -1 (see the remarks below).

ptd (*ptd*) [integer], default: **0**.

The number of samples of PTD elements along each edge in the interval of edges (see the remarks below).

Gap Points (*gap_points*) [sequence of structs].

A sequence of gaps on the scatterer is specified on which the gap-PTD elements are calculated. If two or more subsequent gaps of the scatterer have identical number of gap-PTD elements, the number of elements may be specified for all these gaps by defining an interval of gaps by the members *start_gap* and *end_gap*. If there are no such gaps, *gap_points* must contain the same number of elements as there are illuminated gaps. For each gap contained in the interval of gaps, the number of gap-PTD elements is defined by the members *ptd1* and *ptd2*, and the gap method is defined by the member *gap_method*.

Start Gap (*start_gap*) [integer], default: **-1**.

Number of the first edge in the interval of edges. Must be positive or specified to -1 (see the remarks below).

End Gap (*end_gap*) [integer], default: **-1**.

Number of the last edge in the interval of edges. Must be positive or specified to -1 (see the remarks below).

ptd1 (*ptd1*) [integer], default: **0**.

For each edge in the interval of edges, the number of samples of gap-PTD elements along the length direction of the gap (see the remarks below).

ptd2 (*ptd2*) [integer], default: **0**.

For each edge in the interval of edges, the number of samples of gap-PTD elements across the gap (see the remarks below).

Gap Method (*gap_method*) [item from a list of character strings], default: **automatic**.

For each edge in the interval of edges, the type of gap-PTD elements to be used (see the remarks below).

automatic

The gap-PTD method is chosen automatically according to the width of the individual gaps.

narrow_gap

The narrow-gap approximation is used for the gap-PTD elements.

wide_gap

The wide-gap approximation is used for the gap-PTD elements.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Spill Over (*spill_over*) [item from a list of character strings], default: **off**.

Determines if spillover shall be calculated.

off

The spillover is not calculated.

on

The total power W incident on the scatterer is calculated and the spillover is computed as $4\pi/W$. Thus, the result reflects only the true spillover when the feed is power normalised to 4π (see the remarks below). The spillover calculation can significantly increase the computation time if the incident field comes from another PO source because both the incident E- and H-field are required in order to compute Poynting's vector.

Ray Output (*ray_output*) [item from a list of character strings], default: **none**.

This attribute defines how the field of the currents shall be calculated if the field is used as input to a GTD or PTD analysis. The attribute is also actual for a subsequent PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case, *ray_output* has the same meaning as stated below for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that the field cannot be used as input to a GTD analysis. In a subsequent PTD analysis the direction of propagation is assumed to be the direction of Poynting's vector at the field point.

all

The field is computed as a ray field with one ray from each current element. A subsequent GTD or PTD analysis is then carried out for each ray separately. This is the most accurate method for GTD/PTD analysis, but it may be time consuming.

spherical

The field from the elements will be treated as a single ray which emanates from the origin of the coordinate system specified by *coor_sys*. This procedure is only accurate if the scatterer in the following GTD/PTD analysis is in the far field from the currents or the currents are phased such that they radiate a field at the scatterer with a spherical phase front centred at the origin.

plane

The field from the currents will be treated as a single ray which is parallel to the z -axis of the coordinate system specified by *coor_sys*. This procedure is only accurate if the currents radiate a field in the direction of the z -axis of the coordinate system *coor_sys* and if this field has a nearly plane phase front at the scatterer to be analysed by GTD/PTD.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate Systems*s defining the coordinate system of this PO source. The currents are calculated in this coordinate system.

File Name (*file_name*) [file name].

Name of file in which the PO and PTD currents are stored. If a *file_name* is not given, the currents are stored in an internal file, which is deleted at the end of the session. The recommended file extension is *.cur*, cf. Section *Standard Currents File*.

Command Types

The *PO, Multi-Face Scatterer* is a class under *PO Analysis*, see under *PO Analysis* for available commands.

Remarks

Unless *auto_convergence_of_po* is specified to 'off' in the command *Get Currents* for the PO calculations, the user shall not be concerned with the details of the integration grids in PO as these are determined automatically. The default values of *po_points* and *edge_points* may thus be applied, details are given in the following.

However, the user may control the PO and PTD integrations by specifying *po_points* and *edge_points*, respectively. In that case, only the faces and the edges specified are included. For a *Box* and for *Polygonal Struts*, the program detects automatically if a face or an edge is on the shadow side, in which case the PO currents or the PTD currents, respectively, are set to zero.

This section contains remarks on the following topics:

- The *start* and *end* parameters
- Handling of the PO and PTD contributions
- The number of integration points
- Estimates of the parameters *po1*, *po2* and the *ptd* parameters
- The gap method
- Spillover

The **start** and **end** parameters

The parameters:

- *start_face* and *end_face* of the attribute *po_points*,
- *start_edge* and *end_edge* of the attribute *edge_points*, and
- *start_gap* and *end_gap* of the attribute *gap_points*

specify the number of scattering components to be included in the PO or PTD integration. If the start component and end component are both specified to -1, then all components of this type are included in the analysis. If for example, *start_face* and *end_face* are both -1, then all faces of the scatterer are included in the analysis.

Handling of the PO and PTD contributions

When the PO object is used as the target in a *Get Currents* command, the attribute *method* selects the contributions to be calculated and stored in the currents file:

- If *method* is specified to 'po' or 'po_plus_ptd', then the PO contribution is calculated. If auto-convergence is disabled, then the PO calculation is performed for the faces specified in the attribute *po_points* with the number of PO points specified in the same attribute. If auto-convergence is enabled, then the PO auto-convergence calculation is performed for the faces specified in the attribute *po_points*, independent of the actually specified number of PO points.
- If *method* is specified to 'po_plus_ptd' or 'ptd', then the PTD contribution is calculated. If auto-convergence is disabled, then the PTD calculation is performed for the edges and gaps specified in the attributes *edge_points* and *gap_points* with the number of PTD points specified in the same attribute. If auto-convergence is enabled, then the PTD auto-convergence calculation is performed for the edges and gaps specified in the attributes *edge_points* and *gap_points*, independent of the actually specified number of PTD points.

When the PO object is used as the source, e.g. in a Get Field command, the attribute *method* selects those contributions from the currents file to be included in the numerical integration:

- If *method* is specified to 'po', then only the PO contribution is calculated. The PO calculation is only performed for the faces specified in the attribute *po_points*, even if PO currents for additional faces exist in the file. The PO calculation is performed with the number of PO points available in the file, independent of the actually specified number of PO points in the attribute *po_points*. An error is issued if one of the specified faces is not found in the file. A warning is issued if the file also contains PTD elements.
- If *method* is specified to 'ptd', then only the PTD contribution is calculated. The PTD calculation is only performed for the edges and gaps specified in the attributes *edge_points* and *gap_points*, even if PTD elements for additional edges or gaps exist in the file. The PTD calculation

is performed with the number of PTD points available in the file, independent of the actually specified number of PTD points in the attribute *edge_points* and *gap_points*. An error is issued if one of the specified edges is not found in the file. A warning is issued if the file also contains PO currents.

- If *method* is specified to ‘po_plus_ptd’, then the PO and the PTD contributions are calculated. The calculation is only performed for the faces, edges and gaps specified in the attributes *po_points*, *edge_points* and *gap_points*, even if PO currents for additional faces or PTD elements for additional edges or gaps exist in the file. The calculation is performed with the number of PO and PTD points available in the file, independent of the actually specified number of PO and PTD points in the attributes *po_points*, *edge_points* and *gap_points*. An error is issued if one of the specified faces, edges or gaps is not found in the file.

The attribute *method* thus yields a simple way of extracting the individual scattering contributions to the total field, as exemplified in the following:

First, calculate the total field by a *Get Currents* command followed by a *Get Field* command. Auto-convergence should be enabled in the *Get Currents* command, and the attribute *method* in the PO object set to ‘po_plus_ptd’ to include all scattering effects. Further, the attributes *po_points* and *edge_points* should have their default settings, selecting all faces and edges of the scatterer. Also, a file name must be specified in the attribute *file_name*, so that the currents calculated in this task are stored in the currents file.

Next, calculate only the PO contribution to the total field. This is achieved by changing the attribute *method* in the PO object to ‘po’. The PO contribution could be calculated by re-running the *Get Currents* and *Get Field* commands. However, the same result is obtained by only re-running the *Get Field* command, as the PO currents are simply extracted from the currents file.

Finally, calculate the PTD contribution, by changing *method* to ‘ptd’, and re-running the *Get Field* command.

The three results (PO+PTD, PO, PTD) were obtained by executing a single and not three *Get Currents* commands, thus significantly reducing the computation time. It is also possible to extract the contribution from a selected number of faces, edges and/or gaps by specifying the numbers of these faces and edges in the attributes *po_points*, *edge_points* and *gap_points*. For example to calculate the contribution from edge 2 alone:

Calculate the PTD contribution from edge 2 alone, by selecting *method* to ‘ptd’, *ptd_contributions* to ‘edges’, specifying *edge* to 2 in the *edge_points* attribute, and re-running the *Get Field* command.

The attribute *ptd_contributions* determines the PTD contributions (from edges and/or from gaps) included in the *Get Currents* and *Get Field* commands. The functionality of *ptd_contributions* is similar to the functionality described for the attribute *method* above.

The individual contributions to the PTD field can be extracted from the currents file by changing the settings of the attribute *ptd_contributions* (to ‘all’,

'edges' or 'gaps' as requested) in the same way as described above for the attribute *method*.

The number of integration points

The parameters:

- *po1* and *po2* of the attribute *po_points*,
- *ptd* in the attribute *edge_points*, and
- *ptd1* and *ptd2* in the attribute *gap_points*

specify the number of integration points in the PO or PTD integration grids. If one of these parameters is specified to zero then the corresponding contribution is not included in the analysis.

Estimates of the parameters *po1*, *po2* and the *ptd* parameters

It is essential that the number of integration points is so large that the numerical integration has converged. On the other hand, a too high value may cause an unacceptable long processing time. It is therefore necessary to perform a convergence test in which the parameters *po1* and *po2* and the parameters *ptd*, *ptd1* and *ptd2* are increased until the field calculation is independent on the parameters. This is easiest done by the auto convergence feature of the command *Get Currents* but if the user wants to control the convergence this may be performed by switching the auto converge off in the command *Get Currents*. In that case *po1*, *po2* and *ptd* shall here be specified to non-zero values (in order not to skip the calculations).

The necessary values of the parameters *po1* and *po2* depend on the type of scatterer that is analysed as well as on the position of the field points. Estimated values will be given in the following. These may be applied as starting values when the user wants to control the convergence accuracy. The convergence analysis must be done for each face of the scatterer.

Polar integration grid

For the general case the integration grid is a polar grid when projected on the *xy*-plane of the reflector coordinate system (for the exceptions, see below). The centre of this grid is coincident with the centre of the rim. *po1* gives the number of integration points in radial direction (along ρ) and *po2* gives the number of points in azimuthal direction (along ϕ). If the rim is circular and if the aperture of the reflector is focused, the number of points necessary for convergence can be estimated by

$$\begin{aligned} po1 &= \text{nint}(z/2.4) \\ po2 &= \text{nint}(z) \end{aligned}$$

where

$$z = 1.09 \cdot \pi \frac{D}{\lambda} \sin \theta_0 + 10 \quad (1)$$

The function *nint* means the nearest integer to the argument, and *D* is the reflector diameter, λ is the wavelength and θ_0 is the output angle from the beam centre out to which the integration should converge. For worst-case

illumination and observation angles it may be necessary to increase z to $z = 2\pi D/\lambda$.

The estimate $po2$ can also be used for the number ptd of PTD points along the outer rim.

Non-polar integration grids

If the scatterer is a *Reflector* without holes and the rim is a *Rectangular Rim*, then $po1$ and $po2$ give the number of integration points along x and y , respectively, of the reflector coordinate system. For a focused aperture the integration will converge if

$$po1 = nint \left(1.75 \frac{D_x}{\lambda} \sin \theta_x + 10 \right)$$

$$po2 = nint \left(1.75 \frac{D_y}{\lambda} \sin \theta_y + 10 \right)$$

where D_x and D_y are the reflector dimensions along x and y , respectively. The PO integral will converge from the beam centre out to the angle θ_x towards the x -axis and out to the angle θ_y towards the y -axis. If the reflector is not focused values of $po1$ and $po2$ up to

$$po1 = 3.5 \frac{D_x}{\lambda}$$

$$po2 = 3.5 \frac{D_y}{\lambda} \quad (2)$$

may be necessary.

These estimates can also be used for the number, ptd , of PTD points in the PTD integration. This means that for the edges parallel to the x -axis, ptd shall have the same value as $po1$, and for the edges parallel to the y -axis the ptd value shall be the same as for $po2$.

If the scatterer is a *Reflector* without holes and the rim is a *Triangular Rim*, $po2$ signifies the number of integration points along the longest edge and $po1$ the number of points along the height perpendicular to that edge and again the expressions above may be applied.

In case the scatterer is 1) a *Reflector* with central hole and with rims of types *Rectangular Rim* or *Triangular Rim*, or 2) a *Reflector* with a hole defined by the `holes` attribute, an advanced quadrilinear mesh is used. For this mesh `po1` and `po2` indicate the number of integration points along two perpendicular directions defined individually on each patch in the quadrilinear mesh. No estimates can be provided for `po1`, `po2`, and `ptd`, and these values should always be determined by using the auto convergence feature of the *Get Currents* command.

Also, when the scatterer is a *Triangular Plate*, $po2$ signifies the number of integration points along the longest edge and $po1$ the number of points along the height perpendicular to that edge. Since a plane will not in general focus the field, it is necessary with a fine integration grid in order to ensure convergence. Usually, $po1$ and $po2$ should be between $2D/\lambda$ and $4D/\lambda$, where D is the corresponding height, D_h , or edge length, D_e , respectively. If the height and the side length differ considerably, it may be necessary to

increase *po1* above $4D_h/\lambda$; however, *po1* should never need to be greater than *po2*. The number of PTD samples, *ptd*, shall be at least *po2* along the longest edge. The number of PTD samples along the other two edges shall result in a sampling density being approximately the same as along the longest edge.

If the scatterer is a plane *Parallelogram*, *po1* signifies the number of integration points along *vec_1*, and *po2* the number of integration points along *vec_2*, where *vec_1* and *vec_2* are attributes of the class *Parallelogram*. Since a plane will not in general focus the field, it is necessary with a fine integration grid in order to ensure convergence. Usually, *po1* and *po2* should be between $2D/\lambda$ and $4D/\lambda$, where *D* is the corresponding side length, i.e. the length of *vec_1* and *vec_2*, respectively.

If the scatterer is a plane *Rectangular Plate*, *po1* signifies the number of integration points along the edge from *corner_1* to *corner_2* and *po2* the number of points along the perpendicular edge. *corner_1* and *corner_2* are attributes of the class *Rectangular Plate*. Again, since a plane will not in general focus the field, it is necessary with a fine integration grid in order to ensure convergence. Therefore, *po1* and *po2* should usually be between $2D/\lambda$ and $4D/\lambda$, where *D* is the corresponding side length.

A *Box* has six faces which are numbered as shown in the following table with face 1 corresponding to *z_max*, the face with maximum *z*-value, etc., see also the illustration in the remarks section of the description of class *Box*.

For each illuminated face, *po1* and *po2* may be specified denoting the number of integration points along *edge1* and *edge2*, respectively. For each illuminated edge, *ptd* may be specified denoting the number of integration points along that edge. The edge numbers are related to the faces of the box according to the following table and are illustrated in the description of class *Box*.

Face of box	edge1 (related to <i>po1</i> and <i>ptd</i>)	edge2 (related to <i>po2</i> and <i>ptd</i>)
Face number	Edge number	
1 (<i>z_max</i>)	1 and 3	2 and 4
2 (<i>x_max</i>)	10 and 2	6 and 7
3 (<i>y_max</i>)	11 and 3	8 and 7
4 (<i>x_min</i>)	12 and 4	5 and 8
5 (<i>y_min</i>)	9 and 1	5 and 6
6 (<i>z_min</i>)	9 and 11	12 and 10

Usually, *po1* and *po2* should be between $\frac{2D}{\lambda}$ and $\frac{4D}{\lambda}$, with *D* being the corresponding side length.

The number of PTD samples should follow the respective values of *po1* and *po2*.

Gap method

The struct member *gap_method* in the attribute *gap_points* selects the analysis method used in the analysis of the scattering from the gaps. Observe that only scatterers of the type *Panels in Polar Grid* can be analysed with gap-PTD.

There are three settings for the *gap_method*:

- ‘narrow_gap’, in which case the narrow-gap approximation is used. A warning is issued if part of the gap is wider than the narrow-gap limit of 0.32λ , but the program is continuing.
- ‘wide_gap’, in which case the wide-gap approximation is used. A warning is issued if part of the gap is narrower than the narrow-gap limit of 0.32λ , but the program is continuing
- ‘automatic’. If the entire gap is narrower than the narrow-gap limit of 0.32λ , then the narrow-gap approximation is used. Otherwise, the wide-gap approximation is used. A warning is issued if some part of the gap is narrower and another part of the gap is wider than the narrow-gap limit, but the program is continuing.

Spillover

In GRASP the spillover in dB is defined by

$$10 \log(4\pi/W)$$

where W is the total power hitting the scatterer. In the GRASP output file, the spillover is printed as well as the relative power hitting the scatterer, $W/4\pi$. See also the Technical Description for further information of PO convergence and spillover.

PO, PANELS IN POLAR GRID (po_panels_in_polar_grid)

Purpose

The *PO, Panels in Polar Grid* class is used to specify how the PO and PTD currents shall be calculated on a *Panels in Polar Grid* reflector. The currents are stored in the object, and can serve as a source in later calculations.

The *PO, Panels in Polar Grid* class provides an easy definition of the PO and PTD calculation for the entire *Panels in Polar Grid* reflector. If the contribution from a single panel or a single gap of the reflector is wanted, the *PO, Multi-Face Scatterer* should be used instead.

In the specification of PO calculations, the panels are referred to as faces.

Links

Classes→*Electrical Objects*→*PO Analysis*→*PO, Panels in Polar Grid*

Remarks

Syntax

```
<object name> po_panels_in_polar_grid
(
    frequency           : ref(<n>),
    panels_in_polar_grid : ref(<n>),
    method              : <si>,
    po_points           : sequence(
        struct(
            start_ring:<i>,
            end_ring:<i>,
            po1:<i>,
            po2:<i>),
            ...),
    ptd_contributions   : <si>,
    edge_points         : sequence(
        struct(
            start_edge:<i>,
            end_edge:<i>,
            ptd:<i>),
            ...),
    circular_gap_points : sequence(
        struct(
            start_gap:<i>,
            end_gap:<i>,
            ptd1:<i>,
            ptd2:<i>),
            ...),
    radial_gap_points   : sequence(
        struct(
            start_ring:<i>,
            end_ring:<i>,
            ptd1:<i>,
            ptd2:<i>),
            ...),
```

```

factor                      : struct(db:<r>, deg:<r>),
spill_over                  : <si>,
coor_sys                    : ref(<n>),
file_name                   : <f>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings

```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the currents are calculated. Further, the *Frequency* class must agree with the *Frequency* class specified in the source object that generates the actual PO or PTD currents.

Panels In Polar Grid (*panels_in_polar_grid*) [name of an object].

Reference to an object of the *Panels in Polar Grid* class on which the PO and PTD currents are computed.

Method (*method*) [item from a list of character strings], default: **po_plus_ptd**.

Determines if the PO and/or PTD contributions are included in the calculations (see the remarks below).

po_plus_ptd

PO as well as PTD contributions are included.

po

Only the PO contribution is included.

ptd

Only the PTD contribution is included.

Po Points (*po_points*) [sequence of structs].

Defines the PO integration grids used on the faces of the *Panels in Polar Grid* reflector. The faces of each ring of the reflector have almost identical size. Identical PO grids are therefore defined on each of these faces. If two or more successive rings of the *Panels in Polar Grid* reflector have identical PO grids, the grids may be specified for all these rings by defining an interval of rings by the members *start_ring* and *end_ring*. If there are no such identical PO grids, *po_points* must contain the same number of elements as there are illuminated face rings on the *Panels in Polar Grid* reflector. For each face contained in the interval of rings, the density of the PO grid is defined by the members *po1* and *po2*.

Start Ring (*start_ring*) [integer], default: **-1.**

Number of the first ring in the interval of faces. Must be positive or -1 (see the remarks below).

End Ring (*end_ring*) [integer], default: **-1.**

Number of the last ring in the interval of faces. Must be positive or -1 (see the remarks below).

po1 (*po1*) [integer], default: **0.**

For each face in the interval of faces, the number of current samples in first grid coordinate. Must be positive or 0. (See the remarks below).

po2 (*po2*) [integer], default: **0.**

For each face in the interval of faces, the number of current samples in second grid coordinate. Must be positive or 0. (See the remarks below).

Ptd Contributions (*ptd_contributions*) [item from a list of character strings], default: **all.**

Determines which PTD contributions are included in the calculations (see the remarks below). The selection has only effect if 'ptd' has been selected by the attribute *method*.

all

All PTD contributions from edges and gaps are included. PTD contribution from the edge of a central hole is not included.

edges

Only the PTD contribution from edges is included. PTD contribution from the edge of a central hole is not included.

gaps

Only the PTD contribution from gaps is included.

Edge Points (*edge_points*) [sequence of structs].

A sequence of edges on the scatterer is specified on which the PTD elements are calculated. If two or more successive edges of the Panels in Polar Grid reflector have identical number of PTD elements, the number of points may be specified for all these edges by defining an interval of edges by the members *start_edge* and *end_edge*. If there are no such edges, *edge_points* must contain the same number of elements as there are illuminated edges. For each edge contained in the interval of edges, the number of PTD elements is defined by the member *ptd*.

Start Edge (*start_edge*) [integer], default: **-1**.

Number of the first edge in the interval of edges. Must be positive or specified to -1 (see the remarks below).

End Edge (*end_edge*) [integer], default: **-1**.

Number of the last edge in the interval of edges. Must be positive or specified to -1 (see the remarks below).

ptd (*ptd*) [integer], default: **0**.

The number of samples of PTD elements along each edge in the interval of edges (see the remarks below).

Circular Gap Points (*circular_gap_points*) [sequence of structs].

A sequence of circular gaps on the *Panels in Polar Grid* scatterer is specified on which the gap-PTD elements are calculated. If two or more successive circular gaps of the *Panels in Polar Grid* scatterer have identical number of gap-PTD elements, the number of elements may be specified for all these gaps by defining an interval of gaps by the members *start_gap* and *end_gap*. If there are no such circular gaps, *circular_gap_points* must contain the same number of elements as there are illuminated circular gaps. For each gap contained in the interval of gaps, the number of gap-PTD elements is defined by the members *ptd1* and *ptd2*.

Start Gap (*start_gap*) [integer], default: **-1**.

Number of the first gap in the interval of circular gaps. Must be positive or -1 (see remarks below).

End Gap (*end_gap*) [integer], default: **-1**.

Number of the last gap in the interval of circular gaps. Must be positive or -1 (see remarks below).

ptd1 (*ptd1*) [integer], default: **0**.

For each gap in the interval of circular gaps, the number of samples of gap-PTD elements in the tangent direction of the gap, i.e. along the central circle defining the centre of the gap (see also the remarks below).

ptd2 (*ptd2*) [integer], default: **0**.

For each gap in the interval of circular gaps, the number of samples of gap-PTD elements across the gap (see remarks below).

Radial Gap Points (*radial_gap_points*) [sequence of structs].

A sequence of radial gaps on the *Panels in Polar Grid* scatterer is specified on which the gap-PTD elements are calculated. All radial gaps in a ring of panels have the same number of PTD elements. If different numbers of PTD elements needs to be specified for radial gaps in the same ring, the *PO, Multi-Face Scatterer* must be used instead. If two or more successive rings of the *Panels in Polar Grid* scatterer have radial gaps with identical number of gap-PTD elements, the number of elements may be specified for all these gaps by defining an interval of rings by the members *start_ring* and *end_ring*. If there are no such rings, *radial_gap_points* must contain the same number of elements as there are illuminated rings. For each gap contained in the interval of rings, the number of gap-PTD elements is defined by the members *ptd1* and *ptd2*.

Start Ring (*start_ring*) [integer], default: **-1**.

Number of the first ring in the interval of radial gaps. Must be positive or -1 (see the remarks below).

End Ring (*end_ring*) [integer], default: **-1**.

Number of the last ring in the interval of radial gaps. Must be positive or -1 (see the remarks below).

ptd1 (*ptd1*) [integer], default: **0**.

For each gap in the interval of rings, the number of samples of gap-PTD elements along the tangent direction of the gap, i.e. along the radial line defining the centre of the gap (see also the remarks below).

ptd2 (*ptd2*) [integer], default: **0**.

For each gap in the interval of rings, the number of samples of gap-PTD elements across the gap (see also the remarks below).

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Spill Over (*spill_over*) [item from a list of character strings], default: **off**.

Determines if spillover shall be calculated.

off

The spillover is not calculated.

on

The total power W incident on the scatterer is calculated and the spillover is computed as $4\pi/W$. Thus, the result reflects only the true spillover when the feed is power normalised to 4π (see the remarks below). The spillover calculation can significantly increase the computation time if the incident field comes from another PO source because both the incident E- and H-field are required in order to compute Poynting's vector.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate System*. The currents are calculated in this coordinate system.

File Name (*file_name*) [file name].

Name of file in which the PO and PTD currents are stored. If a *file_name* is not given, the currents are stored in an internal file, which is deleted at the end of the session. The recommended file extension is *.cur*, cf. Section *Standard Currents File*.

Command Types

The *PO, Panels in Polar Grid* is a class under *PO Analysis*, see under *PO Analysis* for available commands.

Remarks

The *PO, Panels in Polar Grid* class yields an easy way to specify the parameters for the PO analysis of a *Panels in Polar Grid* reflector.

The faces forming a ring of the reflector have almost identical size. The *PO, Panels in Polar Grid* class therefore defines identical PO grids on all faces belonging to the same ring. If different PO grids are desired for faces in the same ring, or if the PO scattered field from a single face in a ring is desired, the *PO, Multi-Face Scatterer* class must be used instead.

Similarly, the radial gaps in a single ring have almost identical size. The *PO, Panels in Polar Grid* class therefore defines identical gap-PTD grids on all radial gaps belonging to the same ring. If different gap-PTD grids are desired for gaps in the same ring, or if the PO scattered field from a single gap in a ring is desired, the *PO, Multi-Face Scatterer* class must be used instead.

The circular gaps and the edges can be defined individually.

It should be noted that when the *PO, Panels in Polar Grid* class is used as a target in a *Get Currents* command, where the automatic PO convergence procedure is enabled, then the faces, edges and gaps are treated individually. In this case, faces belonging to the same ring may have different PO grids, and radial gaps may have different gap-PTD grids. The information about the different grids will be stored in the PO file and reported to the log-file.

This section further contains remarks on the following topics:

- The *start* and *end* parameters

- Handling of the PO and PTD contributions
- The number of integration points
- The gap method
- Spillover

The **start** and **end** parameters

The parameters:

- *start_ring* and *end_ring* of the attributes *po_points* and *radial_gap_points*,
- *start_edge* and *end_edge* of the attribute *edge_points*, and
- *start_gap* and *end_gap* of the attribute *circular_gap_points*

specify the number of scattering components to be included in the PO or PTD integration. If the start component and end component are both specified to -1, then all components of this type are included in the analysis. If for example, *start_ring* and *end_ring* in *po_points* are both -1, then all rings of the panelled reflector are included in the analysis.

Handling of the PO and PTD contributions

When the PO object is used as the target in a *Get Currents* command, the attribute *method* selects the contributions to be calculated and stored in the currents file:

- If *method* is specified to 'po' or 'po_plus_ptd', then the PO contribution is calculated. If auto-convergence is disabled, then the PO calculation is performed for the faces specified in the attribute *po_points* with the number of PO points specified in the same attribute. If auto-convergence is enabled, then the PO auto-convergence calculation is performed for the faces specified in the attribute *po_points*, independent of the actually specified number of PO points.
- If *method* is specified to 'po_plus_ptd' or 'ptd', then the PTD contribution is calculated. If auto-convergence is disabled, then the PTD calculation is performed for the edges and gaps specified in the attributes *edge_points*, *circular_gap_points* and *radial_gap_points* with the number of PTD points specified in the same attribute. If auto-convergence is enabled, then the PTD auto-convergence calculation is performed for the edges and gaps specified in the attributes *edge_points*, *circular_gap_points* and *radial_gap_points*, independent of the actually specified number of PTD points.

When the PO object is used as the source, e.g. in a *Get Field* command, the attribute *method* selects those contributions from the currents file to be included in the numerical integration:

- If *method* is specified to 'po', then only the PO contribution is calculated. The PO calculation is only performed for the faces specified in the attribute *po_points*, even if PO currents for additional faces exist in the file. The PO calculation is performed with the number of PO points available in the file, independent of the actually specified number of

PO points in the attribute *po_points*. An error is issued if one of the specified faces is not found in the file. A warning is issued if the file also contains PTD elements.

- If *method* is specified to ‘ptd’, then only the PTD contribution is calculated. The PTD calculation is only performed for the edges and gaps specified in the attributes *edge_points*, *circular_gap_points* and *radial_gap_points*, even if PTD elements for additional edges or gaps exist in the file. The PTD calculation is performed with the number of PTD points available in the file, independent of the actually specified number of PTD points in the attribute *edge_points*, *circular_gap_points* and *radial_gap_points*. An error is issued if one of the specified edges is not found in the file. A warning is issued if the file also contains PO currents.
- If *method* is specified to ‘po_plus_ptd’, then the PO and the PTD contributions are calculated. The calculation is only performed for the faces, edges and gaps specified in the attributes *po_points*, *edge_points*, *circular_gap_points* and *radial_gap_points*, even if PO currents for additional faces or PTD elements for additional edges or gaps exist in the file. The calculation is performed with the number of PO and PTD points available in the file, independent of the actually specified number of PO and PTD points in the attributes *po_points*, *edge_points*, *circular_gap_points* and *radial_gap_points*. An error is issued if one of the specified faces, edges or gaps is not found in the file.

The attribute *method* thus yields a simple way of extracting the individual scattering contributions to the total field, as exemplified in the following:

First, calculate the total field by a *Get Currents* command followed by a *Get Field* command. Auto-convergence should be enabled in the *Get Currents* command, and the attribute *method* in the PO object set to ‘po_plus_ptd’ to include all scattering effects. Further, the attributes *po_points* and *edge_points* should have their default settings, selecting all faces and edges of the scatterer. Also, a file name must be specified in the attribute *file_name*, so that the currents calculated in this task is stored in the currents file.

Next, calculate only the PO contribution to the total field. This is achieved by changing the attribute *method* in the PO object to ‘po’. The PO contribution could be calculated by re-running the *Get Currents* and *Get Field* commands. However, the same result is obtained by only re-running the *Get Field* command, as the PO currents are simply extracted from the currents file.

Finally, calculate the PTD contribution, by changing *method* to ‘ptd’, and re-running the *Get Field* command.

The three results (PO+PTD, PO, PTD) were obtained by executing a single and not three *Get Currents* commands, thus significantly reducing the computation time. It is also possible to extract the contribution from a selected number of faces, edges and/or gaps by specifying the numbers of these faces and edges in the attributes *po_points*, *edge_points*, *circular_gap_points* and *radial_gap_points*. For example to calculate the contribution from edge 2 alone:

Calculate the PTD contribution from edge 2 alone, by selecting *method* to ‘ptd’, *ptd_contributions* to ‘edges’, specifying *edge* to 2 in the *edge_points* attribute, and re-running the [Get Field](#) command.

The attribute *ptd_contributions* determines the PTD contributions (from edges and/or from gaps) included in the [Get Currents](#) and [Get Field](#) commands. The functionality of *ptd_contributions* is similar to the functionality described for the attribute *method* above.

The individual contributions to the PTD field can be extracted from the currents file by changing the settings of the attribute *ptd_contributions* (to ‘all’, ‘edges’ or ‘gaps’ as requested) in the same way as described above for the attribute *method*.

The number of integration points

The parameters:

- *po1* and *po2* of the attribute *po_points*,
- *ptd* in the attribute *edge_points*,
- *ptd1* and *ptd2* in the attribute *circular_gap_points*, and
- *ptd1* and *ptd2* in the attribute *radial_gap_points*

All specify numbers of integration points in the PO or PTD integration grids. If one of these parameters is specified to zero then the corresponding contribution is not included in the analysis.

It is essential that the number of integration points is so large that the numerical integration has converged. On the other hand, a too high value may cause an unacceptable long processing time. It is therefore necessary to perform a convergence test in which the above-mentioned parameters are increased until the field calculation is independent on the parameters. This is easiest done by the auto convergence feature of the command [Get Currents](#) but if the user wants to control the convergence this may be performed by switching the auto converge off in the command [Get Currents](#). In that case *po1*, *po2* and *ptd* shall be specified to non-zero values (in order not to skip the calculations).

Gap method

For each gap, GRASP selects automatically whether the narrow_gap or wide_gap approximation is used in the analysis. If the projected gap width is smaller than the narrow_gap limit of 0.32λ , then the narrow_gap approximation is used. Otherwise, the wide_gap approximation is used. If the user wants to control the gap method used, the [PO, Multi-Face Scatterer](#) class must be used instead.

Spillover

In GRASP the spillover in dB is defined by

$$10 \log(4\pi/W)$$

where *W* is the total power hitting the scatterer. In the GRASP output file, the spillover is printed as well as the relative power hitting the scatterer, *W*/ 4π . See also the Technical Description for further information of PO convergence and spillover.

PO, APERTURE IN SCREEN (po_aperture_in_screen)

Purpose

The analysis by class *PO, Aperture in Screen* is performed using Babinet's principle by which the aperture is replaced by a planar conducting mirror that fills the aperture in the screen. Hereafter, the mirror is analysed by Physical Optics as described in the class *PO, Single-Face Scatterer* and the field is finally determined by Babinet's principle. More details can be found in the remarks below.

The equivalent currents on the aperture are stored in the object, and can serve as a source in later calculations.

This class is only available with the Quast add-on.

Links

Classes→*Electrical Objects*→*PO Analysis*→*PO, Aperture in Screen*

Remarks

Syntax

```
<object name> po_aperture_in_screen
(
    frequency          : ref(<n>),
    scatterer         : ref(<n>),
    method            : <si>,
    po_points         : struct(po1:<i>, po2:<i>),
    ptd_points        : sequence(
                            struct(edge:<i>,
                                   ptd:<i>),
                            ...),
    factor            : struct(db:<r>, deg:<r>),
    spill_over        : <si>,
    ray_output        : <si>,
    coor_sys          : ref(<n>),
    file_name         : <f>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the equivalent currents are calculated. The radiated field from the currents can only be calculated at these frequencies. Further, the

Frequency class must agree with the *Frequency* class specified in the source object that generates the equivalent currents.

Scatterer (*scatterer*) [name of an object].

Reference to an object of the class *Aperture in Screen* on which the equivalent currents are computed.

Method (*method*) [item from a list of character strings], default: **po_plus_ptd**.

Determines if the PO and/or PTD contributions are included in the calculations (see the remarks below).

po_plus_ptd

PO as well as PTD contributions are included.

po

Only the PO contribution is included.

ptd

Only the PTD contribution is included.

PO Points (*po_points*) [struct].

The density of the PO integration grid is defined by the members *po1* and *po2*.

po1 (*po1*) [integer], default: **0**.

Number of current samples in first grid coordinate (see the remarks below for details).

po2 (*po2*) [integer], default: **0**.

Number of current samples in second grid coordinate (see the remarks below for details).

PTD Points (*ptd_points*) [sequence of structs].

A sequence of edges of the aperture is specified on which the PTD currents are calculated. Each struct member contains:

Edge Number (*edge*) [integer], default: **-1**.

Number of an edge along which the PTD currents shall be calculated. If specified to -1 then all edges are included. If contributions are requested from specific edges these shall be specified individually. For the numbering of the edges the user is referred to the description of the individual *Rim* classes.

ptd (*ptd*) [integer], default: **0**.

Number of samples of the current along the edge (see the remarks below). If *edge* is specified as -1, the number applies to the entire length of all edges. The elements are distributed according to the length of the individual edges in order to obtain a nearly uniform distribution of the samples.

Factor (*factor*) [struct].

The radiated field through the aperture will be multiplied by a complex factor:

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Spill Over (*spill_over*) [item from a list of character strings], default: **off**.

Determines if spillover shall be calculated.

off

The spillover is not calculated.

on

The total power W incident on the scatterer is calculated and the spillover is computed as $4\pi/W$. Thus, the result reflects only the true spillover when the feed is power normalised to 4π (see the Remarks below). The spillover calculation can significantly increase the computation time if the incident field comes from another PO source because both the incident E- and H-field are required in order to compute Poynting's vector.

Ray Output (*ray_output*) [item from a list of character strings], default: **none**.

This attribute defines how the field of the equivalent currents shall be calculated if the field is used as input to a GTD or PTD analysis. The attribute is also actual for a subsequent PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case, Ray Output has the same meaning as stated below for PTD. The following may be specified:

none

The direction of propagation of the radiated field will not be computed. This means that the field cannot be used as input to a GTD analysis. In a subsequent PTD analysis the direction of propagation is assumed to be the direction of Poynting's vector at the field point.

all

The field is computed as a ray field with one ray from each current element. A subsequent GTD or PTD analysis is then carried out for each ray separately. This is the most accurate method for GTD/PTD analysis, but it may be time consuming.

spherical

The field from the elements will be treated as a single ray which emanates from the origin of the coordinate system specified by Coordinate System. This procedure is only accurate if the scatterer in the following GTD/PTD analysis is in the far field from the currents or the currents are phased such that they radiate a field at the scatterer with a spherical phase front centred at the origin.

plane

The field from the currents will be treated as a single ray which is parallel to the z -axis of the coordinate system specified by Coordinate System. This procedure is only accurate if the currents radiate a field in the direction of the z -axis of the coordinate system Coordinate System and if this field has a nearly plane phase front at the scatterer to be analysed by GTD/PTD.

Coordinate System (coor_sys) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system of this source. The equivalent currents are calculated in this coordinate system.

File Name (file_name) [file name].

Name of file in which the equivalent currents are stored. When a File Name is not given, the equivalent currents are stored in an internal file, which is deleted at the end of the session. The recommended file extension is .cur, cf. *Standard Currents File*.

Remarks

This section contains remarks on the following topics:

- Handling of the PO and PTD contributions
- The number of integration points
- Estimates of the parameters po1, po2 and ptd
- Spillover

Handling of the PO and PTD contributions

The attribute Method works in the same way as described in the remarks for the *PO, Single-Face Scatterer* class.

The number of integration points

The parameters:

- po1 and po2 of the attribute PO Points, and
- ptd in the attribute PTD Points

specify numbers of integration points in the PO or PTD integration grids. If one of these parameters is specified to zero then the corresponding contribution is not included in the analysis.

Normally the necessary values of `po1`, `po2` and `ptd` are determined automatically by the `Get Currents` command, but the user may bypass the automatic convergence test and insert appropriate values directly. Estimates are given in the remarks to the class `PO, Single-Face Scatterer`.

Spillover

In GRASP the spillover in dB is defined by

$$10 \log\left(\frac{4\pi}{W}\right)$$

where W is the total power hitting the scatterer. In the GRASP output file, the spillover is printed as well as the relative power hitting the scatterer, $W/4\pi$. See also the Technical Description for further information of PO convergence and spillover.

Babinet's Principle

This principle states that the radiation through an aperture is equal to the scattered field from a planar mirror with the same size and shape as the aperture. The principle is exact for an aperture in an infinite conducting screen of zero thickness. Furthermore, the role of E- and H-fields must be interchanged in the two configurations, but this is handled automatically in the program. A simple example on an aperture in a screen is shown in Figure 1.

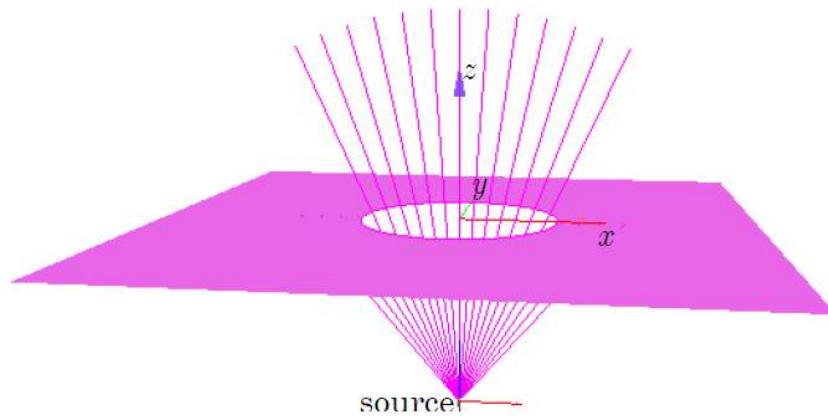


Figure 1 A source radiating through an aperture in a screen.

The total field above the screen is given by the radiation of the equivalent currents in the `PO, Aperture in Screen` object. The incident field from the feed shall not be added.

The field scattered from the aperture below the screen can also be found, but that is somewhat more complicated. Below the screen the equivalent currents in the `PO, Aperture in Screen` do not directly radiate the scattered field, but the combination

$$\vec{E}_{below} = \vec{E}_r - \vec{E}_s$$

where \vec{E}_s is the wanted scattered field from the screen and \vec{E}_r is the reflection of the incident field in an infinite screen without aperture(s).

The complications arise in determining the reflected field \vec{E}_r . Applying PO on an infinite screen is not possible and if the screen is truncated to a finite size then edge effects will occur unless the edge illumination is very low or the screen must be very large which demands long computation time. The best solution may be to apply GO (applying *Single-Reflector GTD*) on a screen which is large enough to include all reflected rays over the full region of interest.

In the example shown in Figure 1, the wavelength is 1 mm and the distance from the feed to the screen is 100 mm. The radius of the circular aperture is 50 mm and the feed is a Gaussian beam with a far-field taper of -12 dB at 30° . The far-field above the screen is shown in Figure 2 and compared to the direct feed radiation without the screen. The effect of the screen is clearly seen.

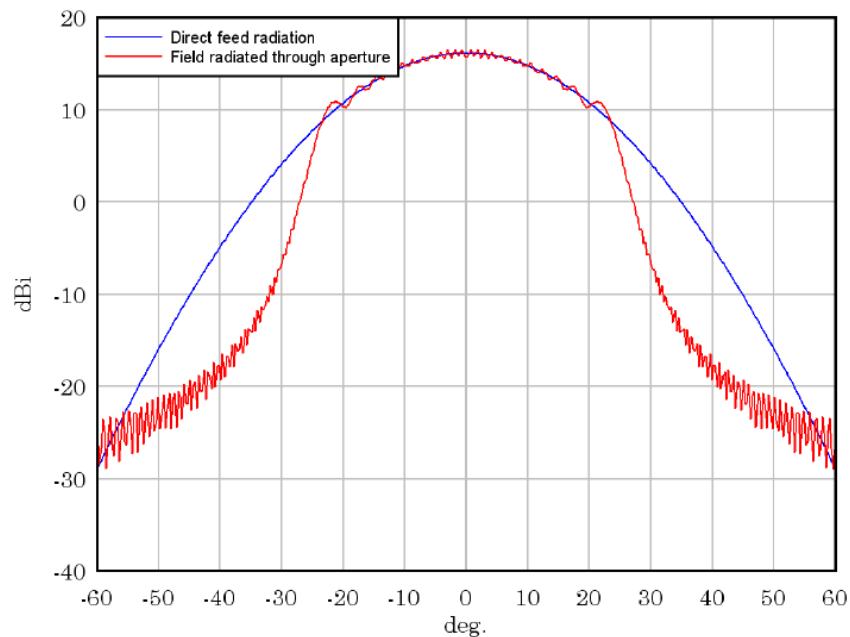


Figure 2

The direct feed radiation (blue) and the field radiated through the aperture (red).

PO, LENS (po_lens)

Purpose

The **PO, Lens** class is used to specify PO analysis of a *Simple Lens*. A set of equivalent currents on each face of the lens is calculated and stored in the object. These currents can serve as a source in later calculations.

This class is only available with the Quast add-on.

Links

Classes→*Electrical Objects*→*PO Analysis*→**PO, Lens**

Remarks

Syntax

```
<object name> po_lens
(
    frequency           : ref(<n>),
    lens                : ref(<n>),
    get_field           : <si>,
    method              : <si>,
    waist_radius        : sequence(<rl>, ...),
    gauss_laguerre_mode_selection : struct(m_max:<i>, n_max:<i>),
    po_points           : struct(face1_po1:<i>, face1_po2:<i>,
                                  face2_po1:<i>, face2_po2:<i>),
    factor              : struct(db:<r>, deg:<r>),
    spill_over          : <si>,
    coor_sys            : ref(<n>),
    current_file_face1 : <f>,
    current_file_face2 : <f>,
    gbc_file            : <f>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a **Frequency** object, defining the frequencies at which the currents are calculated. Further, the **Frequency** class must agree with the **Frequency** class specified in the object used as source to generate the incident field on the lens.

Lens (*lens*) [name of an object].

Reference to an object of class **Simple Lens** on which the equivalent PO currents are computed.

Get Field (*get_field*) [item from a list of character strings], default: **lens_in_screen**.

The following values can be selected, which affect how the radiated field from the lens is calculated.

lens_in_screen

This option is the simplest and should be chosen in most cases. Only the equivalent currents on the lens face opposite to the source of the incident field are retained in the calculation of the radiated field and correspond to the situation where the lens is mounted in an opaque screen. The equivalent currents will radiate the total field in the forward half space and the incident field from the source should not be added.

lens_in_free_space

This option should be chosen if the backward reflection from the lens is asked for or if the mounting structure can be neglected. Equivalent currents on both faces of the lens are used in the calculation of the radiated field corresponding to a lens in free space without any mounting structure. The total field is obtained by adding the incident field from the source, analogous to standard PO on a reflector. The scattered field is valid in both the forward and backward half space.

face1

This option is only useful in special situations. Only the equivalent currents on the lens face 1 are retained in the calculation of the radiated field.

face2

This option is only useful in special situations. Only the equivalent currents on the lens face 2 are retained in the calculation of the radiated field.

Method (*method*) [item from a list of character strings], default: **go_plus_po**.

Determines the method used for calculation of equivalent currents on the lens.

go_plus_po

This is the default method and useful down to lens diameters of approximately 15 wavelengths. Equivalent currents are calculated on both faces of the lens. GO (Geometrical Optics) is used to find the ray paths to as well as the field at the faces of the lens.

po_plus_po

This method should be used if the diameter of the lens is below approximately 15 wavelengths. Equivalent currents are calculated on both faces of the lens. The radiation integral is used to propagate the field from the front face to the back face of the lens.

gauss_laguerre

This method gives a very fast field calculation and is useful for lenses with diameters larger than approximately 20 wavelengths. The incident field must resemble a Gaussian beam so that expansion in a Gauss-Laguerre series is possible.

Waist Radius (*waist_radius*) [sequence of real numbers with unit of length], default: **0**.

Only used if the attribute `Method` is specified to 'gauss_laguerre'. The sequence must contain the waist radii for each of the frequencies in the analysis. The Gauss Laguerre analysis is primarily intended to be activated through the Frame Design Tool by which the waists are calculated automatically.

Gauss Laguerre Mode Selection (*gauss_laguerre_mode_selection*) [struct].

Only used if the attribute `Method` is specified to 'gauss_laguerre'. The Gauss Laguerre analysis is primarily intended to be activated through the Frame Design Tool by which the modes are selected in the Tool. Specification of the number of modes in the Gauss-Laguerre expansion. The two struct members must be given for each of the frequencies in the analysis:

M Max (*m_max*) [integer], default: **0**.

Maximum value of the *m*-index (angular index) in the expansion, $0 \leq m \leq m_{max}$. Must be zero or positive.

N Max (*n_max*) [integer], default: **0**.

Maximum value of the *n*-index (angular index) in the expansion, $0 \leq n \leq n_{max}$. Must be zero or positive.

PO Points (*po_points*) [struct].

The density of the PO integration grid is defined by the members of this struct. Normally these values will be calculated in the auto-convergence carried out by the [Get Currents](#) command. A general discussion on the parameters can be found in the description of class [PO, Single-Face Scatterer](#).

Face1-po1 (*face1_po1*) [integer], default: **0**.

Number of current samples in the radial grid coordinate on face 1.

Face1-po2 (*face1_po2*) [integer], default: **0**.

Number of current samples in the angular grid coordinate on face 1.

Face2_po1 (*face2_po1*) [integer], default: **0**.

Number of current samples in the radial grid coordinate on face 2.

Face2-po2 (*face2_po2*) [integer], default: **0**.

Number of current samples in the angular grid coordinate on face 2.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor:

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Spill Over (*spill_over*) [item from a list of character strings], default: **off**.

Determines if spillover shall be calculated.

off

The spillover is not calculated.

on

The total power W incident on the scatterer is calculated and the spillover is computed as $4\pi/W$. Thus, the result reflects only the true spillover when the feed is power normalised to 4π (see the remarks below). The spillover calculation can double the computation time if the incident field comes from another PO source because both the incident E- and H-field are required in order to compute Poynting's vector.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system of this PO source. The currents are calculated in this coordinate system.

Current File Face1 (*current_file_face1*) [file name].

Name of file in which the equivalent currents on face 1 are stored. When Current File Face1 is not given, the currents are stored in an internal file, which is deleted at the end of the session. The recommended file extension is .cur, cf. *Standard Currents File*.

Current File Face2 (*current_file_face2*) [file name].

Name of file in which the equivalent currents on face 2 are stored. When Current File Face2 is not given, the currents are stored in an internal file, which is deleted at the end of the session. The recommended file extension is .cur, cf. *Standard Currents File*.

Gbc File (*gbc_file*) [file name].

Name of file in which the Gauss-Laguerre beam coefficients are stored. When `Gbc File` is not given, the coefficients are stored in an internal file, which is deleted at the end of the session. The recommended file extension is `.gbc`, cf. [Gauss-Laguerre Beam Coefficients](#).

Remarks

Expansion of the field in Gauss-Laguerre modes is a very fast method, especially for large lenses. The method is well suited for propagation through lenses as part of an optical system as the main beam is well predicted while the edge diffractions are not included.

Commands

The lens analysis is performed by the commands [Get Currents](#) and [Get Field](#). First, the command [Get Currents](#) is used to calculate equivalent currents on the faces of the lens. It is recommended to use the automatic convergence facility for automatic determination of the density of the current integration grids on the faces. Hereafter, the [Get Field](#) command can be issued to compute the radiated field at the desired output cuts or grids. If `Method` is specified to the default value '`lens_in_screen`' then the incident field should not be added. If `Method` is specified to '`lens_in_free_space`' the incident field must be added by an [Add Field](#) command to get the total field.

Spillover

In GRASP the spillover in dB is defined by

$$10 \log\left(\frac{4\pi}{W}\right)$$

where W is the total power hitting the scatterer. In the GRASP output file, the spillover is printed as well as the relative power hitting the scatterer, $W/4\pi$. See also the GRASP Technical Description.

PO CONVERGENCE (OBSOLETE) (po_convergence)

Purpose

The *PO Convergence (Obsolete)* class can calculate PO currents on a sequence of scatterers and the radiation from the currents. The class is obsolete and it is recommended to apply the auto convergence in the *Get Currents* command instead.

However, the class is included for backward compatibility only, and it is not applicable to the new *PO Analysis* objects of GRASP.

When applying PO it is important to ensure that the currents on the scatterer are determined in a grid sufficiently fine to yield convergence from the current integration. The integration grid is two-dimensional over the surface of the scatterer with *po1* points in one dimension and *po2* points in the other. Further, *ptd* current points on the edge may be introduced.

The surface integration can be carried out applying the standard *PO Analysis* objects scatterer by scatterer. However, the present source class can calculate minimum values of *po1* and *po2* when a required field accuracy is specified. Several scatterers may be involved and a grid is determined on each. The currents are stored in the different *PO Analysis* objects for subsequent field calculations. Data for the integration grid that achieved convergence are listed.

Links

[Classes](#)→[Electrical Objects](#)→[PO Analysis](#)→[PO Convergence \(Obsolete\)](#)

Remarks

Syntax

```
<object name> po_convergence
(
    source                      : ref(<n>),
    po_objects                  : sequence(ref(<n>), ...),
    field_accuracy              : <r>,
    max_bisections              : <i>,
    integration_grid_limit     : <si>,
    ptd                         : <si>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<si> = item from a list of character strings
```

Attributes

Source (*source*) [name of an object].

Reference to an object of the class **Source** (except another object of class **PO Convergence (Obsolete)**, see also the subsection Limitations in the Remark section below) used as the incident field on the first scatterer in the following sequence of Po Objects.

Po Objects (*po_objects*) [sequence of names of other objects].

A sequence of one or more references to objects of a class under **PO Analysis** shall be specified. Each object defines a scatterer on which the current will be determined. The scatterers are illuminated as given in the sequence: The source illuminates the first object, which scatters to the second etc. The last object radiates to a **Field Storage** object. The objects in the sequence must all be different. A converged integration grid is calculated for each object and the resultant currents are stored in the respective objects.

Field Accuracy (*field_accuracy*) [real number], default: **-80**.

The PO integration has converged when a small change in 'po1' and 'po2' will cause a pattern change less than the specified Field Accuracy. The pattern change is calculated as the complex field difference between two successive pattern calculations based on two sets of 'po1' and 'po2'. The Field Accuracy shall be given in dB relative to the maximum power level in the pattern. See also the remarks below.

Max Bisections (*max_bisections*) [integer], default: **6**.

The attribute Max Bisections controls how precise po1 and po2 are determined. The value must be a positive integer giving the maximum number of bisections in the integration grid evaluation. If zero, no bisections are performed. See the remarks below.

Integration Grid Limit (*integration_grid_limit*) [item from a list of character strings], default: **on**.

This attribute is used to stop the optimization if the field accuracy limit can not be obtained.

on

The maximum values of po1 and po2 are given by simple estimates for the different scatterers and then multiplied by 10. See the remarks below.

off

No maximum values for po1 and po2.

PTD (*ptd*) [item from a list of character strings], default: **on**.

This attribute is used to determine if PTD calculations shall be included in the convergence determination. The PTD calculation is controlled by the ptd parameter (an integer) in the attribute 'ptd_points' of the associated **PO Analysis** object. This ptd parameter will be denoted 'ptd' (in quotes) below.

on

If 'ptd' is zero then no PTD calculations are carried out. If 'ptd' is not zero then 'ptd' gets the same value as either 'po1' or 'po2' depending on the type of the scatterer. For example, for a **Reflector** with elliptical rim it is 'po2' because this relates to the number of samples in the circumferential direction. This also holds when more than one 'ptd'-parameter (in the sequence of 'ptd_points') is specified. The value assigned is the determined value of 'po1' or 'po2' in the convergence test.

off

PTD calculations are included in the convergence calculations applying the 'ptd'-values specified in the **PO Analysis** object. The 'ptd'-values are not optimized.

Remarks

Limitations

The object **PO Convergence (Obsolete)** is a source but it does not have the same range of applications as a general source. It is a source intended for the specific purpose of calculating a field scattered in one or more scatterers. The source can thus not be applied in e.g. GTD calculations or in current calculations, or as a source for another **PO Convergence (Obsolete)**.

Parameters

The parameters *po1*, *po2* and *ptd* for specifying the current integration grid depend on the type of scatterer analysed and on the position of the field points. They are described in details in the **PO Analysis** objects for the respective scatterers. The PO and PTD current optimization is then controlled through the specified **PO Analysis** objects. For structures with several faces or edges, only those defined in the respective **PO Analysis** objects will have their grid parameters optimized. The PTD values are optimized concurrently with the values of *po1* or *po2* according to the type of the scatterer.

po1 and *po2* on the last **PO Analysis** object in a scattering sequence depend on the position of the field points (which are defined in the **Field Storage** object referred to in the command). It is often the outermost field points that require the most dense integration grid and thus defines *po1* and *po2*. If the outermost field points are changed, a new PO convergence is required. On the other hand, intermediate field points may be calculated with an unchanged grid. It is recommended that the field points in the beam peak direction is included when *po1* and *po2* are determined because the accuracy is specified relative to the largest determined field value.

It is often advantageous to perform a PO convergence test with very few output points (covering the area of interest). This will give a substantial saving of computation time compared to a dense grid of output points. Hereafter, the radiated field from the last set of currents in the sequence Po Objects

can be calculated with the desired density of output points. It is not necessary to re-compute the currents since they have been stored on file during the convergence test process.

Note, however, if several cuts are specified in the *Field Storage* object, the PO convergence will be carried out cut by cut, and it is the currents corresponding to the convergence test for the last cut which are retained in the file. The resulting values for *po1* and *po2* are the highest values obtained for any of the cuts and these values may not be those for the last cut. If they are not, the currents retained in the file can then only be re-used for this last cut.

The applied method for determining *po1* and *po2* shall shortly be explained in the following. Hereby also the controlling attributes are presented.

po1 and *po2* are determined by varying them independently, starting at 10, 20, 40, 80, ... and so forth. When the Field Accuracy criteria is satisfied, the interval between the last two values is bisected a number of times in order to determine the lowest value of *po1* (*po2*, respectively) satisfying the Field Accuracy criteria. The number of bisections is limited by the attribute Max Bisections. During the process, the applied values of *po1* and *po2* are written in the log file.

The field accuracy may turn out to be specified to a level that cannot be achieved. In order to stop the optimization in such a case, the integration grid parameters *po1* and *po2* can be limited by setting the attribute Integration Grid Limit to 'on'. The applied maximum limits for the grid parameters are found using the following simple estimates from the *PO Analysis* object descriptions and multiplying these by 10:

If the scatterer is a reflector with a polar integration grid, the number of points for convergence over the whole sphere can be estimated by

$$po2 = \frac{2\pi D}{\lambda} \quad (1)$$

$$po1 = \frac{po2}{2.4} \quad (2)$$

where *D* is the maximum reflector dimension and λ is the wavelength.

If the scatterer is a rectangular reflector, *po1* and *po2* give the number of points along *x* and *y*, respectively. For a non-focused aperture the integration limits are

$$po1 = 3.5 \frac{D_x}{\lambda} \quad (3)$$

$$po2 = 3.5 \frac{D_y}{\lambda} \quad (4)$$

where *D_x* and *D_y* are the reflector dimensions along *x* and *y*.

If the scatterer is a triangle, *po2* signifies the number of points along the longest edge and *po1* the number of points along the height perpendicular to the longest edge,

$$po2 = 3.5 \frac{L_{max}}{\lambda} \quad (5)$$

$$po1 \equiv po2 \quad (6)$$

where L_{\max} is the length of the longest edge.

The accuracy of the field, given by the level *field_accuracy* in dB, corresponds to a power accuracy ε (maximum error in power),

$$\varepsilon = 10^{\text{field_accuracy}/10} \quad (7)$$

and to the requirement

$$|E_1 - E_2|^2 / P_{\max} < \varepsilon \text{ at all field points}$$

Here, E_1 and E_2 are the field values at the same field point determined by the two last integration grids. P_{\max} is here the maximum power level in the converged pattern. An accuracy requirement given as a maximum error of $\pm X$ dB at Y dB below peak ($Y > 0$) can be related to a *field_accuracy* by

$$\text{field_accuracy} = \dots \quad (8)$$

$$= 20 \log \left\{ [10^{(-Y+X)/20} - 10^{(-Y-X)/20}] / 2 \right\} \quad (9)$$

$$= -Y + 20 \log \left\{ [10^{X/20} - 10^{-X/20}] / 2 \right\} \quad (10)$$

$$\cong -Y + 20 \log \left\{ \left[1 + \frac{X}{20 \log e} - \left(1 - \frac{X}{20 \log e} \right) \right] / 2 \right\} \quad (11)$$

$$= -Y + 20 \log \left\{ \frac{X}{20 \log e} \right\} \quad (12)$$

$$= -Y - 18.8 \text{dB} + 20 \log \{X\} \quad (13)$$

$$\cong -Y - 20 \text{dB} + 20 \log \{X\} \quad (14)$$

Some examples are given in the following table.

<i>X</i>	<i>Y</i>	<i>field_accuracy</i> (dB)
1 dB	40 dB	-60
0.1 dB	40 dB	-80
1 dB	60 dB	-80

The value of *field_accuracy* can be derived from an accuracy requirement of $\pm X$ dB at Y dB below peak.

In the following example we assume that P_{\max} is the maximum level in dB in the calculated pattern. The pattern of a reflector with diameter $D = 40\lambda$ shall be determined with an accuracy of ± 1 dB at -60 dB. The *field_accuracy* is therefore specified to -80 dB. The directivity of the reflector antenna is, say, 38 dBi which means that P_{\max} is 38 dBi and the accuracy specification will then allow inaccuracies at a level of 38 dBi - 80 dB = -42 dBi.

The field for a side-lobe region will be determined to the same accuracy if it is calculated in the same cut as the field calculation of the main beam. But if the side-lobe region is calculated separately then the *field_accuracy* of -80 dB will result in a more accurate (and time consuming) calculation as P_{\max} within the side-lobe region has a lower value than P_{\max} of the main beam.

Command Types

The PO and PTD currents are generated by the command *Get Currents*. This command is available for the PO classes listed above.

Remarks

PO/PTD is applied as follows. A scatterer is specified in an object using one of the classes under *PO Analysis*, and the currents on the scatterer are determined by a *Get Currents* command in which also the *Source* illuminating the scatterer is specified. The determined currents are stored in the *PO Analysis* object which may be used as *Source* for further computations of, for instance, the field using *Get Field*.

If the field of the scatterer illuminates a second scatterer and the field from the latter is to be determined, a new object of class under *PO Analysis* must be defined and a new *Get Currents* command must be issued with the *PO Analysis* object of the first scatterer as *Source*. The *PO Analysis* object of the second scatterer must finally be used as *Source* in the *Get Field* command for computations of the secondary scattered field.

MOM ANALYSIS

Purpose

The classes defined under *MoM Analysis* are used to specify the calculation of Method of Moments (MoM) currents on various scatterers.

The following classes are available and only available with the MoM add-on:

Scattering determined by a 3D-MoM calculation is defined under

MoM.

Scattering from Body of Revolution (BoR) scatterers determined by a BoR Method of Moments calculation is defined under

BoR-MoM.

In a MoM analysis (not available in a *BoR-MoM* analysis), the full S-parameter matrix of a general multi-port microwave circuit may be determined by introducing in the circuit a

Voltage Generator.

Links

Classes→*Electrical Objects*→*MoM Analysis*

MOM (mom)

Purpose

Method of Moments (MoM) calculations are carried out by one or more objects of the class **MoM**. This is a very accurate method as it includes all electromagnetic interactions.

The objects of class **MoM** specify how the currents on a **Scatterer** shall be determined by means of MoM. The calculation of the currents is activated by the command **Get Currents**, and in this command the **Sources** illuminating the scatterer are also specified. The induced currents on the scatterer due to the field from the sources are calculated and stored in the **MoM** object. These currents can serve as a source in later calculations.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Electrical Objects](#)→[MoM Analysis](#)→[MoM](#)

Remarks

Syntax

```
<object name> mom
(
    frequency : ref(<n>),
    scatterer : ref(<n>),
    polynomial_precision : <i>,
    integration_precision : <i>,
    max_mesh_length : <r>,
    iterative_solution : struct(use_iterative:<si>,
                                 relative_error:<r>, group_size:<r>),
    interbody_coupling : <si>,
    keep_matrix : <si>,
    performance_optim : <si>,
    advanced_polynomial : struct(edge:<i>, wedge:<i>,
                                  junction:<i>, pec:<i>,
                                  dielectric:<i>, wire:<i>),
    factor : struct(db:<r>, deg:<r>),
    ray_output : <si>,
    coor_sys : ref(<n>),
    file_name : <f>,
    colour_plot_file : <f>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the currents are calculated. The radiated field from the currents can only be calculated at these frequencies. Further, the *Frequency* object must agree with the *Frequency* object to which reference is given in the source object that is used as a source in the *Get Currents* command.

Scatterer (*scatterer*) [name of an object].

Reference to a *Scatterer* object on which the currents are computed.

Polynomial Precision (*polynomial_precision*) [integer], default: **0**.

Determines the polynomial expansion order of the unknown currents. The expansion order is automatically adapted to the electrical size of each patch used in the MoM mesh (the internally generated mesh for the MoM computations) of the scatterer. This default level of precision is specified by the value 0. The expansion order may be increased which increases the accuracy of the computed currents but also lead to longer computation time and higher memory requirements. An increased accuracy is obtained by specifying 1 or 2 which increases all expansion orders by 1 or 2, respectively. To increase the order by more than 2 is rarely needed. See the remarks below for a further discussion of convergence issues.

Integration Precision (*integration_precision*) [integer], default: **0**.

Determines the accuracy of the integration algorithms used in the MoM solution. The default level of precision is 0 which is sufficient for most purposes. The integration precision can be increased by specifying 1 or 2. This increases the computation times but not the memory requirements. See the remarks below for a further discussion of convergence issues.

Max Mesh Length (*max_mesh_length*) [real number], default: **2**.

Determines the maximum allowed dimension of the patches used in the MoM mesh of the scatterer (the internally generated mesh for the MoM computations). It is recommended to use as large patches as possible, but not larger than 2 wavelengths; a value of *max_mesh_length* larger than 3 (wavelengths) is not allowed. It is possible to decrease the value to check the accuracy of the geometrical approximation introduced by the meshing procedure. See the remarks below for a further discussion of convergence issues.

Iterative Solution (*iterative_solution*) [struct].

The linear equations set up in the MoM analysis can be solved by either a direct solution (LU-decomposition) or by an iterative solution. The attribute *iterative_solution* determines if and how an iterative solution shall be applied.

Use Iterative (*use_iterative*) [item from a list of character strings], default: **automatic**.

Determines if an iterative solution should be used.

automatic

This is the recommended setting. The program estimates whether an iterative solution is feasible. The estimation is initially based on a number of factors, such as the number of right-hand sides in the system of linear equations to be solved (i.e. the number of beams) and the layout of the mesh. If these factors indicate that an iterative solution is feasible a few iterations are performed and the convergence speed of the iterative algorithm evaluated. If the iterative solution still seems feasible, it will be used. If the iterative solution does not seem feasible the program will revert to a direct solution.

on

The iterative solution is used.

off

The direct solution is used.

Relative Error (*relative_error*) [real number], default: **0.001**.

Sets the acceptable relative error used in the iterative solution. Must be larger than or equal to 10^{-6} . The value is not used in the direct solution. The iterative algorithm stops when the actual relative error becomes below the error level specified by *relative_error*, or it stops after maximum N iterations where N is the number of unknowns in the problem. A warning is issued if the algorithm did not achieve the desired accuracy. The default value of the *relative_error* is 10^{-3} which is usually the best choice.

Group Size (*group_size*) [real number], default: **4**.

Specifies the group size for the preconditioner which is used for solving the MoM matrix. The group size is specified in wavelengths and the default group size is the best choice for most problems.

Interbody Coupling (*interbody_coupling*) [item from a list of character strings], default: **automatic**.

Some scatterer classes may define several scattering bodies in a single object, e.g. *Circular Struts*, *Polygonal Struts*, *Circular Stiffeners*, *Polygonal Stiffeners* and *Scatterer Cluster*.

If the scatterer referred to in the attribute *scatterer* belongs to one of these classes then the *interbody_coupling* attribute specifies if the coupling between the individual scattering bodies shall be included or

if each individual body should be treated as an independent scatterer (see also the remarks below).

If the scatterer referred to in the attribute *scatterer* belongs to a *Scatterer* class other than those mentioned above then the scatterer includes only a single body and the value of *interbody_coupling* is not used.

automatic

Coupling effects are determined by the type of the scatterer. For *Circular Struts*, *Polygonal Struts*, *Circular Stiffeners* and *Polygonal Stiffeners* the coupling is neglected. For *Scatterer Cluster* the coupling is included. If a *Scatterer Cluster* contains *Circular Struts*, *Polygonal Struts*, *Circular Stiffeners* or *Polygonal Stiffeners*, all coupling effects are included.

on

Coupling effects between all scattering bodies are included. This setting results in the longest computation time and the highest memory requirement.

off

Coupling effects between scattering bodies are neglected. This setting results in the shortest computation time and the lowest memory requirement.

Keep Matrix (keep_matrix) [item from a list of character strings], default: **off**.

This setting is useful if more than one *Get Currents* command is issued with the same *MoM* target object but different *Source* objects, i.e. different incident fields. This allows the MoM matrix to be reused since only the incident field changes. The matrix can only be kept if the currents are computed at a single frequency and on a single body, or on multiple bodies with the attribute *interbody_coupling* set to 'on'. Furthermore, if a new *Get Currents* command is issued with another *MoM* object as the target, a new matrix is computed (which may then be kept) but any previously kept matrix will be deleted from memory. A typical configuration where this is useful is a dual reflector antenna system where the subreflector (but not the main reflector) is analysed by MoM. Multiply scattered fields bounce between the subreflector and the main reflector. These fields can then be included with relative low computational effort since the matrix may be stored in the memory.

off

The matrix will not be kept. Any new *Get Currents* command will request the matrix to be computed again.

on

The matrix is kept in memory for later use if possible.

Performance Optim (*performance_optim*) [item from a list of character strings], default: **automatic**.

Determines whether the execution of a *Get Currents* command should be optimized for the shortest computation time or the lowest memory requirement. In particular, the MoM matrix is symmetric for perfect electrically conducting scatterers, which allows for storing the matrix in compressed format saving 50 percent of the memory. However, the compressed matrix format slows down the solution and should only be used if memory resources are low. In addition, the matrix fill time can be reduced by using the memory more aggressively.

automatic

The program will automatically determine the amount of physical memory available in the computer. If the size of the matrix is less than 80 percent of the physical memory the *Get Currents* command will be optimized for the shortest possible solution time. This implies that the full matrix will be stored even if it is symmetric. If the matrix takes more space, then the *Get Currents* command will be optimized for low memory availability.

memory

The program execution is optimized for low memory requirements. This should be used if memory resources are low.

speed

The *Get Currents* command is optimized for the shortest possible solution time. It is then assumed that memory is abundant. Note that this setting results in a very long computation time if memory resources are low since this will force the program to use virtual memory (swap to disk).

Advanced Polynomial (*advanced_polynomial*) [struct].

This attribute is similar to the Polynomial Precision attribute except that the polynomial expansion order may here be controlled on different parts of the scatterer independently. This is an advantage if an increased expansion order is only required on a small part of the scatterer. See the Remarks below for a further discussion of convergence issues. The default values of all the following struct members are 0. By specifying 1 or 2 the order of the polynomial expansion is increased by 1 or 2, respectively.

Edge (*edge*) [integer], default: **0**.

Controls the polynomial expansion order on all patches along external edges (an external edge limits the surface, see *MoM Plot* for examples). The value of *edge* may be increased to 1 or 2 if one suspects that knife edge singularities are a problem. However, this is usually not the case unless the edges are located in close vicinity of other scatterers.

Wedge (wedge) [integer], default: **0**.

Controls the polynomial expansion order on all patches along wedges. A wedge is an edge shared by two patches at an angle of 45 degrees or more. The value of *wedge* may be increased to 1 or 2 if one suspects that wedge singularities are a problem. However, this is usually not the case.

Junction (junction) [integer], default: **0**.

Controls the polynomial expansion order on all patches along junctions. A junction is an edge shared by more than two patches. This may happen for *Tabulated Meshes* and for *Scatterer Clusters*. To increase the order by more than 2 is rarely needed.

Pec (pec) [integer], default: **0**.

Controls the polynomial expansion order on all perfectly conducting parts of the scatterer. The value of *pec* may be increased to 1 or 2. Higher values are rarely needed.

Dielectric (dielectric) [integer], default: **0**.

Controls the polynomial expansion order on all dielectric parts of the scatterer. The value of *dielectric* may be increased to 1 or 2. Higher values are rarely needed.

Wire (wire) [integer], default: **0**.

Controls the polynomial expansion order on all wire parts of the scatterer. The value of *wire* may be increased to 1 or 2. Higher values are rarely needed.

Factor (factor) [struct].

The field radiated by the MoM currents will be multiplied by a complex factor,

Amplitude in dB (db) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (deg) [real number], default: **0**.

Phase of the factor, in degrees.

Ray Output (ray_output) [item from a list of character strings], default: **none**.

This attribute defines how the field radiated by the currents shall be calculated when used as source in a GTD or PTD analysis. The attribute is also actual for a subsequent PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular case, *ray_output* has the same meaning as stated below for PTD.

none

The direction of propagation of the radiated field is not computed. This implies that the field cannot be used as input to a GTD analysis. In a subsequent PTD analysis the direction of propagation is assumed to be the direction of Poynting's vector at the field point.

all

The field is computed as a ray field with one ray from each current element. A subsequent GTD or PTD analysis is then carried out for each ray separately. This is the most accurate method for GTD/PTD analysis, but it may be time consuming.

spherical

The radiated field will be treated as a single ray which emanates from the origin of the coordinate system specified by *ray_coor_sys*. This procedure is only accurate if the scatterer in the following GTD/PTD analysis is in the far field region of the currents or the currents are phased such that they radiate a field at the scatterer with a spherical phase front centred at the origin of the coordinate system specified by *ray_coor_sys*.

plane

The field radiated by the currents will be treated as a single ray which is parallel to the *z*-axis of the coordinate system specified by *ray_coor_sys*. This procedure is only accurate if the currents radiate a field in the direction of the *z*-axis of the coordinate system *ray_coor_sys* and if this field has a nearly plane phase front at the scatterer to be analysed by GTD/PTD.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system in which the MoM currents are calculated.

File Name (*file_name*) [file name].

Name of the file in which the MoM currents are stored. When a *file_name* is not given, the currents are stored in an internal file, which is deleted at the end of the session. The recommended file extension is *.cur*, cf. *Standard Currents File*.

Colour Plot File (*colour_plot_file*) [file name].

Name of the file in which the information for colour current plots is stored. The recommended file extension is *.cpf*, cf. *Currents Colour Plot File*.

Command Types

The MoM currents are generated by the command

Get Currents.

Remarks

The MoM class has a reference to a single scatterer only. If the MoM currents on several scatterers are desired, this may be specified in two different ways:

The currents may be determined scatterer by scatterer with one **MoM** object for each scatterer and by consecutive executions of the command **Get Currents**. The currents on one scatterer are then determined independently of the currents on the other scatterers.

Alternatively, the scatterers may be connected to a single scatterer of the class **Scatterer Cluster**. This scatterer may be specified in a single **MoM** object and a single **Get Currents** command may be executed. Then the mutual coupling between the scatterers is included. It involves, however, a large matrix to be solved which may be time consuming.

See also the application of the attribute Interbody Coupling below.

The interbody coupling

The attribute **Interbody Coupling** determines how the coupling between individual bodies shall be handled. As an example consider a double-reflector antenna system with a feed illuminating a subreflector supported by four circular struts, as illustrated in the figure below. The struts will be modelled in a single object of class **Circular Struts**; and this object will thus define four bodies.

The struts will influence the radiation from the antenna, and we will use MoM to determine this influence. Two contributions will be considered:

1. The field from the feed will illuminate the struts and cause a scattering via the subreflector to the main reflector, (1) in Figure 1.
2. The field from the feed reflected by the subreflector will illuminate the struts and in this way contribute to the illumination of the main reflector, (2) in Figure 1.

The strong contribution due to the blocking of the field from the main reflector by the struts will not be considered in this example illustrating the application of the attribute **Interbody Coupling**.

We will consider three different cases with different use of interbody coupling.

(A) No interbody coupling: All **MoM** objects have the attribute **Interbody Coupling** specified to 'off', meaning that no coupling effects are included in the analysis. The PO-currents on the main reflector are determined from three contributions to the field incident on the main reflector:

(A1) The nominal incident subreflector field determined by MoM on the subreflector with the feed as the source.

(A2) Strut contribution (1) determined by first MoM on the struts with the feed as the source, and subsequently MoM on the subreflector with MoM on the struts as the source.

(A3) Strut contribution (2) determined by first MoM on the subreflector with the feed as the source, and subsequently MoM on the struts with MoM on the subreflector as the source.

(B) Interbody coupling for individual scatterers. All **MoM** objects have the attribute **Interbody Coupling** specified to 'on', meaning that coupling effects are included for the individual scatterers. The subreflector contains

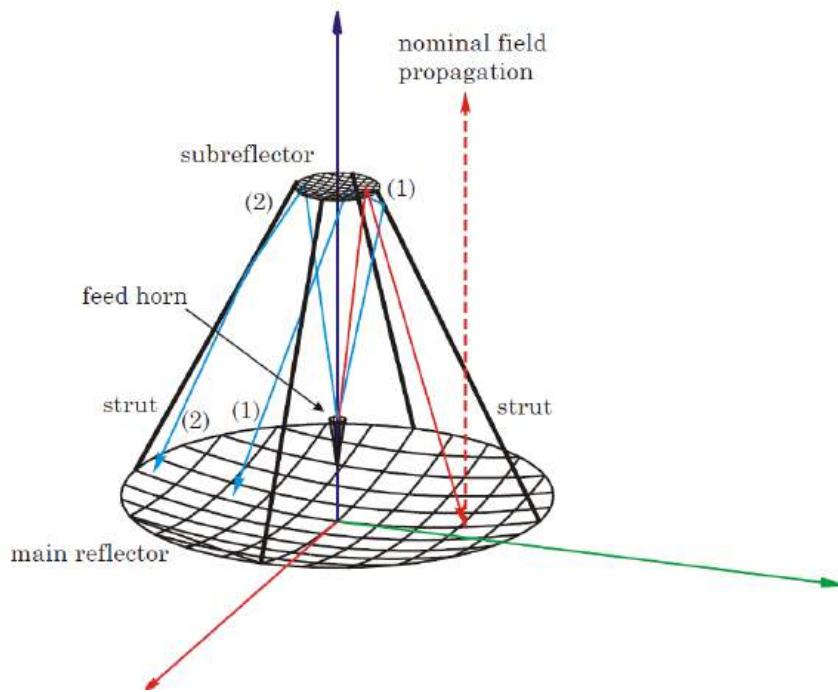


Figure 1

A double-reflector system with the nominal field propagation shown in red, and the two strut contributions considered shown in blue.

only a single body, hence the value of Interbody Coupling has no effect for this scatterer. The strut scatterer contains four bodies, and the coupling between the struts is included in the analysis. The PO-currents on the main reflector are determined from three contributions to the field incident on the main reflector, which are the same as specified in item (A1), (A2) and (A3) above.

(C) Interbody coupling for all scatterers. The subreflector and the struts are grouped to a single scatterer using the Scatterer Cluster class. The corresponding MoM object has the attribute Interbody Coupling specified to 'on', meaning that all coupling effects between the five bodies in the *Scatterer Cluster* object are included. The PO-currents on the main reflector are determined from a single contribution to the field incident on the main reflector:

(C1) Nominal field as well as strut contribution (1) and (2) determined by MoM on the *Scatterer Cluster* with the feed as the source.

The orientation of the struts will not cause a strong scattering from one strut to the others. Thus, case (A) and (B) will yield practically identical results for the present configuration.

It is emphasized that the computation time and the memory requirement increase considerably with the number of patches to be handled in the MoM computation as the size of the matrix equation increases significantly. Dividing the geometry into a number of subsystems and solving for each of these subsystems by MoM (and thereby neglecting the coupling between the subsystems) is faster than solving MoM for the full geometry. Case (A) above is thus much faster to solve with less storage requirements than case (B) which again is faster and requires less storage than case (C).

Convergence accuracy

The user should always verify that the computed currents are converged to a satisfactory accuracy. This is checked by comparing the result to a more accurate result computed with enhanced accuracy.

Four factors contribute to the overall accuracy of the currents calculation:

1. Geometrical discretization error. The scatterer is discretized using curved patches which may be up to $2\lambda \times 2\lambda$ if Max Mesh Length is set to the default value, 2 (wavelengths). The surfaces of the patches are described by polynomials of up to 3rd order. If the surface curvature is significant within a single patch this geometrical approximation may be too crude. The size of the patches can then be decreased by decreasing the attribute Max Mesh Length until the 3rd order polynomial approximation describes the surface with sufficient accuracy.
2. Polynomial expansion order. The current is expanded in polynomials of up to 10th order. The polynomial order on each patch is automatically selected based on the electrical size of the patch. However, the automatic selected expansion order is intentionally chosen to yield sufficient accuracy when the patch is near-rectangular and a part of a large smooth surface. If this is not the case, the polynomial expansion order may need to be increased by increasing the value of the attribute Polynomial Precision to 1 or 2 which will add 1 or 2, respectively, to the automatically chosen order. Alternatively, one may use the attribute Advanced Polynomial to increase the expansion order locally on a part of the scatterer.
3. Integration precision. The MoM algorithm requires a large number of double surface integrals to be evaluated which is done by Gauss-Legendre integration. The order of the integration rules is automatically selected based on the electrical size of each patch. However, the automatic selected integration order is intentionally chosen to yield sufficient accuracy if the patch is a part of a large smooth surface and the default polynomial expansion is used. If this is not the case, the integration order may need to be increased by increasing the value of the attribute Integration Precision, e.g., by 1 or 2.
4. Density of output grid. The MoM algorithm results in a set of coefficients which can be used to evaluate the current at any point of the scatterer. However, this representation of the current is not suitable for subsequent field evaluation since the expansion polynomials would be recomputed for each observation point. Instead, the currents are evaluated in an integration grid that is stored in a current file which may be used for later field computations. The density of this grid is automatically chosen based on the electrical size of each patch, assuming that the observation point in subsequent computations is not too close to the patch. If the near field is evaluated relatively close to the surface the density of the output grid may need to be increased which is done by increasing the value of the attribute Integration Precision, e.g., by 1 or 2.

It is seen that the above parameters are suggested increased by 1 or 2. Higher values may be specified but this will normally not be necessary.

It is up to the user to ensure that the values of the three attributes mentioned above, Polynomial Precision, Integration Precision, and Max Mesh Length, are set to yield a satisfactory accuracy. This can be checked by running at least two consecutive computations with different settings and observe if the changes in the computed field are within the desired accuracy.

As an example, one may try to run the analysis with Polynomial Precision and Integration Precision both set to the default value zero and then repeat the analysis with both parameters set to one. If the changes in the computed fields are insignificant the value zero is sufficient in future computations.

If the surface of the scatterer exhibits a rapid variation, the value of Max Mesh Length must also be checked by performing the analysis with different settings.

Mesh rules

The following rules apply for a MoM mesh:

1. Two neighbouring patches are considered connected if an edge on one patch is identical to an edge on the other patch within a tolerance of $10^{-5}\lambda$. This implies that the two edges have the same number of nodes and that the nodes coincide in pairs. If the rule is fulfilled the edge is modelled as a single shared edge in the MoM computations and the currents are allowed to flow from one patch to the neighbour patch. If the rule is not fulfilled, the edges will be regarded as external edges and the currents are not allowed to flow between the patches. The *MoM Plot* class can be used to check if edges are considered as connected or external edges.
2. An edge can be shared by no more than 4 patches. An error is issued during the MoM computation if this condition is violated.
3. Dielectric regions must be closed. An error is issued during the MoM computation if this condition is violated.
4. A closed metallic region should be assigned to be region number -1 if it is in contact with a dielectric region. If the region number is not assigned to -1 a significant overhead is introduced in the computations since unnecessary basis functions are used to model the zero currents on the internal side of the closed metallic region. If the closed metallic region is not in contact with a dielectric region, there is no such overhead in the computations, and the region number -1 needs not to be specified.

BOR-MOM (bor_mom)

Purpose

Body of Revolution Method of Moments (BoR-MoM) calculations are carried out by one of more objects of the class *BoR-MoM*. BoR-MoM applies to *Body of Revolution* scatterers, and it is a very accurate method since it includes all electromagnetic interactions.

The objects of class *BoR-MoM* specify how the currents on a *Body of Revolution* scatterer are to be determined by means of BoR-MoM. The calculation of the currents is activated by the command *Get Currents*, and in this command the *Sources* illuminating the scatterer are also specified. The induced currents on the scatterer due to the field from the sources are calculated and stored in the *BoR-MoM* object. These currents can serve as a source in later calculations.

BoR-MoM can only be applied to *Body of Revolution* scatterers or *Scatterer Clusters* consisting of *Body of Revolution* scatterers. Since *BoR-MoM* utilises the rotational symmetry of *Body of Revolution* scatterers, it is possible to reduce the computation time as well as the memory and thereby handle larger scatterers than possible with the general 3D-MoM. Although the scatterers have to be rotationally symmetric, there are no restrictions on the *Sources*.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Electrical Objects](#)→[MoM Analysis](#)→[BoR-MoM](#)

[Remarks](#)

Syntax

```
<object name> bor_mom
(
    frequency           : ref(<n>),
    scatterer          : ref(<n>),
    max_mesh_length    : <r>,
    expansion_accuracy : <si>,
    total_power_percentage : <r>,
    min_power_percentage_per_mode<r>,
    factor              : struct(db:<r>, deg:<r>),
    coor_sys            : ref(<n>),
    file_name           : <f>,
    colour_plot_file   : <f>,
    min_m_mode          : <i>,
    max_m_mode          : <i>
)
where
<i> = integer
<n> = name of an object
```

$\langle r \rangle$ = real number
 $\langle f \rangle$ = file name
 $\langle si \rangle$ = item from a list of character strings

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the currents are calculated. The radiated field from the currents can only be calculated at these frequencies. Further, the *Frequency* object must agree with the *Frequency* object to which reference is given in the source object that is used as a source in the *Get Currents* command.

Scatterer (*scatterer*) [name of an object].

Reference to a *Body of Revolution* scatterer object on which the currents are computed. The referenced scatterer can also be a *Scatterer Cluster* consisting of *Body of Revolution* scatterers.

Max Mesh Length (*max_mesh_length*) [real number], default: **2**.

Determines the maximum allowed dimension in wavelengths of the segments used in the BoR-MoM mesh to discretize the generatrix of the BoR-scatterer. It is recommended to use as large segments as possible, but not larger than 2 wavelengths; a value of Max Mesh Length larger than 3 (wavelengths) is not allowed. If the generatrix is highly curved, a segment size of 2 wavelengths may cause a large geometrical approximation error, and the segment size is therefore automatically decreased to ensure a low geometrical approximation error. It is possible to check if this automatic reduction of the segment size for curved segments is accurate by manually decreasing the value of Max Mesh Length, see Remarks below for a further discussion of convergence issues.

Expansion Accuracy (*expansion_accuracy*) [item from a list of character strings], default: **Normal**.

Determines the accuracy of the polynomial expansion used to represent the unknown currents on the scatterers. The polynomial expansion order is automatically adapted to the electrical size of each segment used in the mesh (the internally generated mesh for the BoR-MoM calculation). This default level of precision is sufficient for normal accuracy. The expansion order can be increased which may lead to an improved accuracy of the solution but also lead to significantly longer computation time and higher memory requirements. The normal or the enhanced settings are sufficient for most applications. If large variation of the unknown current occurs on parts of the scatterers, the high or extreme accuracy setting may improve the solution.

Normal

The default level of precision sufficient for normal accuracy.

Enhanced

Increased polynomial order for enhanced accuracy.

High

Further increased polynomial order for high accuracy.

Extreme

Highest polynomial order for extreme accuracy.

Total Power Percentage (*total_power_percentage*) [real number], default: **100**.

The desired power fraction content in percent of the included azimuthal modes. This value is only used if Min m-Mode and Max m-Mode both equal -1.

Minimum Power Percentage per Mode (*min_power_percentage_per_mode*) [real number], default: **0.0001**.

An analysis of an azimuthal mode is only performed if the power fraction of this mode in percent is larger than or equal to the value given by Minimum Power Percentage per Mode,

Factor (*factor*) [struct].

The field radiated by the BoR-MoM currents will be multiplied by a complex factor,

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system in which the BoR-MoM currents are calculated.

File Name (*file_name*) [file name].

Name of the file in which the BoR-MoM currents are stored. When a File Name is not given, the currents are stored in an internal file, which is deleted at the end of the session. The recommended file extension is *.cur*, , cf. *Standard Currents File*.

Colour Plot File (*colour_plot_file*) [file name].

Name of the file in which the information for colour current plots is stored. The recommended file extension is *.cpf*, cf. *Currents Colour Plot File*.

Min m-Mode (*min_m_mode*) [integer], default: **-1**.

The minimum azimuthal mode number to be analysed. If Min m-Mode as well as Max m-Mode equal -1 (the default value), the values of Min m-Mode and Max m-Mode are determined automatically such that the power fraction given by the attribute Total Power Percentage is included.

Max m-Mode (*max_m_mode*) [integer], default: **-1**.

The maximum azimuthal mode number to be analysed. If Min m-Mode as well as Max m-Mode equal -1 (the default value), the values of Min m-Mode and Max m-Mode are determined automatically such that the power fraction given by the attribute Total Power Percentage is included. The maximum allowable value of Max m-Mode is 250.

Command Types

The BoR-MoM currents are generated by the command

Get Currents.

Remarks

The *BoR-MoM* class has a reference to a single scatterer only. If the *BoR-MoM* currents on several scatterers are desired, this may be specified in two different ways:

The currents may be determined scatterer by scatterer with one *BoR-MoM* object for each scatterer and by consecutive executions of the command *Get Currents*. The currents on one scatterer are then determined independently of the currents on the other scatterers, which constitutes an approximation.

Alternatively, the scatterers may be connected to a single scatterer of the class *Scatterer Cluster*. This scatterer may also be specified in a single *BoR-MoM* object and a single *Get Currents* command may be executed. Then the mutual coupling among the scatterers is included, constituting the most accurate solution. It involves, however, a larger matrix to be solved which is more time consuming. The *Scatterer Cluster* must consist of *Body of Revolution* scatterers and they all have to be specified in the same coordinate system.

Convergence accuracy

The user should always verify that the computed currents are converged to a satisfactory accuracy. This is checked by comparing the result to a more accurate solution.

Two factors contribute to the overall accuracy of the currents calculation:

1. Geometrical discretization error. The generatrix of the *Body of Revolution* scatterers is discretized using curved segments which may be up to 2λ if Max Mesh Length is set to the default value, 2 (wavelengths). The geometrical segments are described by polynomials of up to 3rd order. If the curvature is significant within a single segment this geometrical approximation may be too crude, and the segment size is automatically decreased to reduce this geometrical approximation error. It is possible to check if the automatic decrease of the segment size is accurate by

manually decreasing the value of Max Mesh Length. This is done by reducing Max Mesh Length from 2 to, say, 1.5 and checking if the result is unchanged. If so, the value of Max Mesh Length of 2 provides sufficiently low geometrical approximation error. If not, Max Mesh Length is gradually reduced until convergent results are obtained.

2. **Expansion Accuracy.** The current is expanded in polynomials of up to 10th order. The polynomial order on each segment is automatically selected based on the electrical size of the segment. However, the automatic selected expansion order is intentionally chosen to yield sufficient accuracy when the segment is part of a large smooth segment. If this is not the case, the polynomial expansion order may need to be increased by changing the settings of the Expansion Accuracy attribute. The 'Normal' and 'Enhanced' settings are adequate for most applications. However, if large variation of the unknown current occurs on parts of the scatterers, 'High' or 'Extreme' settings may be required. For convergence analysis, one should manually change the Expansion Accuracy from 'Normal' to 'Enhanced', from 'Enhanced' to 'High', and from 'High' to 'Extreme', and stop when convergent results are achieved.

Automatic power determination

The *BoR-MoM* utilises the rotational symmetry of the *Body of Revolution* scatterers to decrease the computation time. However, the *Sources* illuminating the scatterer may be arbitrary, e.g. an offset *Feed*. This entails that several azimuthal modes are required to include the total power of the incident field.

The attributes Min m-Mode and Max m-Mode are used to control the number of azimuthal modes to include in the analysis. The default values for both attributes are -1, meaning that Min m-Mode and Max m-Mode are determined automatically. In this case, the algorithm starts with azimuthal mode 0 and then increases the azimuthal mode number until the power fraction specified in the attribute Total Power Percentage is included, and the analysis stops. After each *BoR-MoM* computation, a list of included azimuthal modes and its corresponding power fraction is shown.

The attribute Minimum Power Percentage per Mode is used to determine how much power to be included in an azimuthal mode before an analysis for that particular mode is initiated. If the power fraction in a given azimuthal mode is less than the value specified by Minimum Power Percentage per Mode, the analysis of that specific mode will not be performed, and this can imply that the value in Total Power Percentage cannot be reached. For such cases, the value in Minimum Power Percentage per Mode should be reduced.

Alternatively, the user can manually specify non-negative values of Min m-Mode and Max m-Mode.

Application example

Consider a front-fed parabolic reflector as shown in Figure 1.

The parabolic reflector is generated using a *Spline Reflector*, and has an aperture diameter of 1 m. The frequency is 12 GHz. Using *MoM* with default

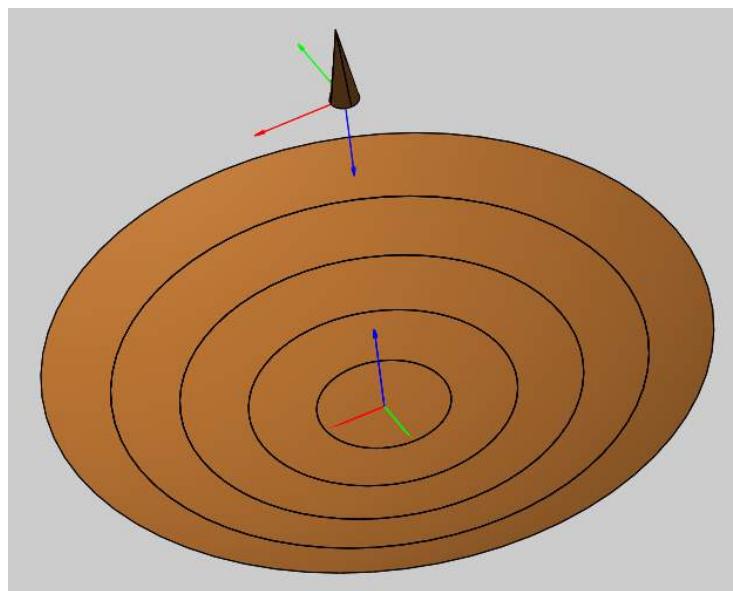


Figure 1 A front-fed parabolic reflector.

settings, the analysis requires a memory storage of 8.7 GB and the computation time is approximately 3 minutes on a standard laptop computer.

Using *BoR-MoM* with Expansion Accuracy selected to 'Normal', the memory requirement is below 1 MB and the analysis takes 0.6 seconds on the same laptop computer used to carry out the *MoM* calculations. Since the feed is on-axis, only one azimuthal mode is required to obtain a Total Power Percentage of 100. For this specific test case, the 'Normal' Expansion Accuracy was sufficient, and increasing the Expansion Accuracy to 'Enhanced' does not improve the accuracy.

VOLTAGE GENERATOR (voltage_generator)

Purpose

The *Voltage Generator* class is used to define voltage sources on wires or on wire-plate junctions. *Voltage Generators* can be used as sources in a Method of Moments (*MoM*) analysis and the S-, Y-, or Z-parameters of the voltage sources can be stored in a file. This makes it possible to retrieve the full S-parameters matrix of a general multi-port microwave circuit.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Electrical Objects](#)→[MoM Analysis](#)→[Voltage Generator](#)

Remarks

Syntax

```
<object name> voltage_generator
(
    generators           : sequence(
        struct(x:<rl>,
               y:<rl>,
               z:<rl>,
               amplitude:<rv>,
               phase:<r>),
        ...),
    parameter_selection : <si>,
    parameter_file      : <f>,
    composite_beam       : <si>,
    power_norm           : <si>,
    coor_sys             : ref(<n>),
    line_impedance       : <r>
)
where
<n> = name of an object
<r> = real number
<rv>= real number with unit of volt
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

Generators (*generators*) [sequence of structs].

Definition of the location and excitation of each voltage generator.

Position of the generator in the coordinate system specified by the *coor_sys* attribute. If no coordinate system is defined, the (x,y,z)-coordinates are assumed to be in the global coordinate system. The generator location must coincide with the location of a wire node shared by two wires, or with the location of a wire-plate junction. An error will be issued if the generator location is illegal. Please see the remarks below for a discussion of the generator orientation.

x (x) [real number with unit of length].

 x-coordinate of position.

y (y) [real number with unit of length].

 y-coordinate of position.

z (z) [real number with unit of length].

 z-coordinate of position.

Amplitude (*amplitude*) [real number with unit of volt], default: **1**.

 The amplitude of the excitation of the voltage generator in Volt.

Phase (*phase*) [real number], default: **0**.

 The phase of the excitation of the voltage generator in degrees.

Parameter Selection (*parameter_selection*) [item from a list of character strings], default: **none**.

 Selects the type of circuit or scattering parameters to compute.

none

 No parameters will be computed and the attributes *parameter_file* and *line_impedance* will be ignored.

s_parameters

 The scattering matrix of the N-port circuit will be computed (where N is the number of generators defined in the attribute *generators*). The value of the attribute *line_impedance* below specifies the characteristic impedance of the transmission lines used in the computation of the S-parameter. The S-parameter matrix is stored as specified in the attribute *parameter_file*. Please consult the MoM add-on manual for details on the S-parameter computation.

z_parameters

 The impedance matrix of the N-port circuit will be computed and stored in the file specified by the attribute *parameter_file*. The attribute *line_impedance* is not used.

y_parameters

 The admittance matrix of the N-port circuit will be computed and stored in the file specified by the attribute *parameter_file*. The attribute *line_impedance* is not used.

Parameter File (*parameter_file*) [file name].

Name of the file in which the S-, Z-, or Y-parameters are stored. The format of the file is described in Section [S, Y, and Z-Parameters](#). Need not to be specified when *parameter_selection* is specified to 'none'.

Composite Beam (*composite_beam*) [item from a list of character strings], default: **yes**.

Specifies whether the currents produced by the generators produce a single composite beam or one individual beam per generator.

yes

The currents produced by all generators are added and radiate a single composite beam.

no

The currents produced by each individual generator produce a single beam. The total number of beams equals the number of generators.

Power Norm (*power_norm*) [item from a list of character strings], default: **on**.

Specifies if the radiated power should be normalized. By the normalization the pattern is normalised to dBi.

on

The radiated power is normalised to 4π Watts. If *composite_beam* is set to 'no', the power radiated into each beam will be 4π Watts. Normalization is default.

off

The radiated power is not normalised.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes [Coordinate System](#) defining the coordinate system in which the positions of the generators are defined.

Line Impedance (*line_impedance*) [real number], default: **50**.

Specifies the characteristic impedance (in ohms) of the transmission lines feeding the generators. This number is required for computing the S-parameters but is not used for computing Z- or Y-parameters. Please consult the MoM add-on manual for a description of the S-parameter computation. Need not to be specified when *parameter_selection* is specified to 'none'.

Command Types

The MoM currents are generated by the command

[Get Currents](#).

Remarks

A *Voltage Generator* acts as a source which generates currents on scatterers. The currents are calculated by the command *Get Currents* in which the *Voltage Generator* is the source and an object of class MoM is the target. Activating the *Get Currents* command also invokes the calculation of the S-(or Z- or Y-) parameters when *parameter_selection* is specified accordingly.

Orientation of voltage generators

Each voltage generator defines an infinitesimal voltage gap that sets up an electric field vector on the wire where the generator is located. The direction of the electric field vector is usually not important when a single generator is defined. However, if multiple generators are defined, the orientation of the electric field vector determines whether the generators are fed in phase or in opposite phases.

The electric field vector, defined by each generator, is computed as the product of the complex generator voltage and a unit vector along the wire. The orientation of unit vector is chosen as follows:

- The unit vector is oriented along the wire and has a positive *z*-component in the global coordinate system.
- If the generator is located on a wire that has zero *z*-component in the global coordinate system, the unit vector along the wire is instead defined with a positive *x*-component.
- If the generator is located on a wire that has zero *x*- and *z*-components in the global coordinate system, the unit vector along the wire is instead oriented along the positive *y*-direction.

Accuracy of S-, Y-, and Z-parameters

Accurate computation of the S-, Y-, and Z-parameters usually requires a better solution accuracy than required for computing the radiated field. It is therefore recommended, that the user checks the convergence of the MoM currents by adjusting the attributes *polynomial_precision* and *integration_precision* in the *MoM* object. Please consult the Remarks section in the *MoM* class description for a discussion of these two attributes.

The accuracy of the S-, Y-, and Z-parameters also depends on the quality of the mesh. If the voltage generator is placed on a wire-plate junction, the following guidelines should be observed:

- The radius of the wire should be smaller than one quarter of the side length of the patch where the wire is attached.
- The wire-plate attachment point should preferably be placed close to the centre of the patch, or close to the centre point of one of the patch edges.

If the generator is placed between two wires, it is recommended not to place the generator at a bend. The wire orientation should be the same on both sides of the generator.

Command Types

For *MoM* and *BoR-MoM*, the MoM currents are generated by the command *Get Currents*.

GTD ANALYSIS

Purpose

The menu *GTD Analysis* contains in the standard version of GRASP the class

Single-Reflector GTD

for determination by GTD of the scattering by a single reflector.

Multi GTD add-on: If more reflectors than one are involved,

Multi-Reflector GTD may be applied.

Scattering involving more general scatterers (including reflectors) may be modelled using classes in the menu

Multi-GTD

Links

Classes→*Electrical Objects*→*GTD Analysis*

SINGLE-REFLECTOR GTD (single_reflector_gtd)

Purpose

Objects of the source class *Single-Reflector GTD* can calculate the GO/GTD field generated by an input source and a scattering object of the *Reflector* class. The rays can only be traced through one reflector. A typical application is to find the currents on a main reflector applying *Single-Reflector GTD* on the subreflector.

Links

[Classes](#)→[Electrical Objects](#)→[GTD Analysis](#)→[Single-Reflector GTD](#)

Remarks

Syntax

```
<object name> single_reflector_gtd
(
    frequency          : ref(<n>),
    source             : ref(<n>),
    reflector          : ref(<n>),
    go                 : <si>,
    ray_tracing        : <si>,
    go_warning         : <si>,
    gtd                : <si>,
    gtd_edge_selection : sequence(<i>, ...),
    direct              : <si>,
    factor              : struct(db:<r>, deg:<r>),
    ray_output          : <si>,
    coor_sys            : ref(<n>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the field is calculated.

Source (*source*) [name of an object].

reference to an object of class *Source* (except *PO Convergence (Obsolete)*) and another object of a class under *GTD Analysis*) specifying the source illuminating the *Reflector* of the single reflector system.

Reflector (*reflector*) [name of an object].

Reference to an object of the class *Reflector* for which the GO/GTD field is computed.

GO (*go*) [item from a list of character strings], default: **off**.

Determines if the GO contribution (reflected field) is included.

Ray Tracing (*ray_tracing*) [item from a list of character strings], default: **simple**.

The search for reflection points can be done in two different ways.

simple

This is a fast method which will work for all common configurations.

advanced

A more time-consuming method, which is able to find reflection points also in very complicated cases.

GO Warning (*go_warning*) [item from a list of character strings], default: **off**.

A warning message may be printed if reflected rays cannot be found for all output field points. This option can be useful e.g. for the sub-reflector analysis in a dual reflector system to test if all points on the main reflector are illuminated by rays reflected from the subreflector.

GTD (*gtd*) [item from a list of character strings], default: **off**.

Determines if the GTD contribution (diffracted field) is included.

GTD Edge Selection (*gtd_edge_selection*) [sequence of integers], default: **-1**.

Sequence of identification numbers for edges for which the GTD field is calculated. Specifying edge number -1 means that GTD is calculated for all outer edges on the reflector and the sequence must then only contain one number. The numbering of the edges is defined in the corresponding *Rim* classes. This attribute is only relevant for reflectors with a circular hole or having a piecewise defined rim such as the *Rectangular Rim* (see the remarks below). For a smooth rim, the edge number should be specified to -1. Further, the attribute has no effect if the above attribute *gtd* is set to off.

Direct (*direct*) [item from a list of character strings], default: **off**.

Determines if the direct field from the source is included.

off

The direct field from the source is not computed.

on

The direct field from the source is added to the GO/GTD field.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Ray Output (*ray_output*) [item from a list of character strings], default: **none**.

This attribute is only important if the *Single-Reflector GTD* object is used as input (as *Source*) to an analysis procedure where the direction of propagation must be known. This is for example the case in a PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case Ray Output has the same meaning as stated below for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that if the field is used as input to an analysis procedure where this direction is required, it will be assumed to be the direction of Poynting's vector at the field point. Such analysis procedures are e.g. PO with surface materials and PTD.

all

The field is computed as a number of ray fields according to the GO/GTD ray tracing. A subsequent analysis by PO with surface materials or by PTD is then carried out for each ray separately. This is an accurate method but it can be time consuming.

spherical

The total field from the reflector will be treated as a single ray, which starts from the origin of the coordinate system defined by the attribute *coor_sys*. This procedure is only accurate if the scatterer in the following analysis is in the far field from the reflector or the radiated field has a spherical phase front.

plane

The total field from the reflector will be treated as a single ray, which is parallel to the *z*-axis of the coordinate system specified by the attribute *coor_sys*. This procedure is only accurate if the reflector radiates a field in the direction of the *z*-axis of *coor_sys* and if this field has a nearly plane phase front at the next scatterer to be analysed.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system of this source. The choice of coordinate system is normally not important. Exceptions are when a spherical expansion of the *Single-Reflector GTD* object shall be calculated (cf. *Get SWE*) or when the object is used as a source in an *Array*.

Remarks

Only ray tracing through a single reflector is implemented in this class. This means that the input source must not be another *Single-Reflector GTD* object since the rays should then be traced through two reflectors.

When the *Rectangular Rim* is used in GTD calculations, it shall be observed that corner diffractions are not included. The effect of corner diffractions is usually low but it shall be noted that the edge-diffracted rays from a rim with straight edges will radiate only in bands over the far-field sphere. Outside these bands, there will be no edge diffracted field and a far-field cut may then show a null field in such regions.

The diffracted field from specific rim sections (edge between two corner points) can be selected by the attribute *gtd_edge_selection*. For instance in *Rectangular Rim* one can select diffraction from side 2 and side 4 by specifying

```
gtd_edge_selection: sequence(2, 4)
```

The diffraction from a hole in the reflector is also selected using the attribute *gtd_edge_selection* by specifying a number one larger than the index of the last outer rim as this is the convention for numbering of a hole in a reflector. For the *Rectangular Rim* this would be 5 and for a smooth rim the number must be 2.

It shall further be noticed that a curved rim may cause caustics in the diffracted field. Caustics can usually be identified as spikes in the pattern. The field computations near caustics will be inaccurate.

MULTI-REFLECTOR GTD (multi_reflector_gtd)

Purpose

Objects of the class *Multi-Reflector GTD* define a scattering process in a specified sequence of *Reflector* and *Circular Struts*. The ray tracing to the field point is carried out and the scattered field is determined by GO/GTD.

For a more general scattering determination class *Multi GTD* is recommended.

This class is only available with the Multi GTD add-on.

Links

[Classes](#)→[Electrical Objects](#)→[GTD Analysis](#)→[Multi-Reflector GTD](#)

Remarks

Syntax

```
<object name> multi_reflector_gtd
(
    frequency           : ref(<n>),
    source              : ref(<n>),
    scatterers          : sequence(
        struct(
            scatterer:ref(<n>),
            method:<si>,
            n_points:<i>),
        ...),
    max_searches        : <i>,
    max_rays            : <i>,
    blocking            : <si>,
    blocking_scatterers: sequence(ref(<n>), ...),
    amplitude_only      : <si>,
    slope               : <si>,
    factor              : struct(db:<r>, deg:<r>),
    ray_output          : <si>,
    trace_list          : <si>,
    trace_file          : struct(status:<si>, name:<f>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the field is calculated.

Source (*source*) [name of an object].

Reference to an object of class *Source* (except *PO Convergence (Obsolete)* and another object of a class under *GTD Analysis*). This is the source illuminating the scattering reflectors and struts.

Scatterers (*scatterers*) [sequence of structs].

The search for ray traces will be performed in this sequence of scatterers, in the order given and according to the specified methods.

Scatterer (*scatterer*) [name of an object], default: **blank**.

Reference to an object of class *Reflector* for which the reflections or diffractions should be computed, or to an object of class *Circular Struts* for determination of reflections.

Method (*method*) [item from a list of character strings], default: **reflection**.

Specifies the type of scattering in the actual *scatterer*.

reflection

Rays are reflected in the surface of the scatterer.

diffraction

Rays are diffracted in the rim of the reflector.

transmission

Rays are transmitted through the surface of a semi-permeable reflector according to its specified *Electrical Properties*.

diffraction_in_edge

Rays are diffracted in a specific indexed edge of a reflector with a rim with numbered edges (such an edge is characterised by having corners).

diffraction_in_hole

Rays are diffracted in the edge of the reflector hole.

N Points (*n_points*) [integer], default: **100**.

Number of equally distributed sample points on the surface of the reflector or along its rim, or along the circular strut. The points are used as starting points for trial ray traces for the numerical ray tracing process. The total number of trial ray traces for a given ray type is found as the product of the specified values of *n_points* for all scatterers along the ray trace.

If *n_points* is specified to zero for reflection on a reflector (*method*:reflection) then a semi-theoretical method is applied to find a single reflection point only.

For diffraction in the rim of a reflector for which the rim has numbered edges (*method*:diffraction_in_edge) the value of *n_points* is the index of the diffracting edge. At most one diffraction point may be found in this edge.

For reflection on circular struts a theoretical method is used and the value of *n_points* is not used.

Max Searches (*max_searches*) [integer], default: **5**.

The ray tracing search for an actual ray type is stopped when the number of unsuccessful trial paths has reached the maximum number of searches given by *max_searches*. The number of unsuccessful trial paths is reset to zero when a ray-trace solution has been found. If *max_searches* is specified to zero, the option is disabled (the search continues). The default value is 5. A higher value is only needed if the user from knowledge to the geometry judges that not all scattering points have been found in a previous run.

Max Rays (*max_rays*) [integer], default: **-1**.

The search for rays is stopped when the number of found rays reaches the value of *max_rays*. If *max_rays* is specified negative or zero, the value is disabled.

Blocking (*blocking*) [item from a list of character strings], default: **on**.

Some ray traces may be obstructed by a scatterer. The check for such blocking is controlled here.

on

All the determined ray traces are checked for blocking in the scatterers referred to in *blocking_scatterers*. Blocked ray traces are then discarded.

off

No check for blocking of the ray traces.

Blocking Scatterers (*blocking_scatterers*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class **Scatterer**. When *blocking* is specified to 'on' the possible ray traces are checked for blocking in the scatterers specified in this sequence of references.

The name of the reference may be 'all' which comprises references to all **Scatterer** objects of the project.

The value of *blocking_scatterers* is not used when *blocking* is specified to 'off'.

Amplitude Only (*amplitude_only*) [item from a list of character strings], default: **off**.

The field contributions of the different rays reaching a field point may be added to a worst-case field (see the Remarks below for applications):

off

The GTD field contributions are added in amplitude and phase.

on

The phase of the GTD fields from the different ray traces are neglected and the fields are added in amplitude only.

Slope (*slope*) [item from a list of character strings], default: **on**.

Controls the handling of the slope-diffracted field.

on

The field contributions from slope-diffractions are included.

off

The slope-diffracted field is neglected.

Factor (*factor*) [struct].

The total radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Ray Output (*ray_output*) [item from a list of character strings], default: **none**.

This attribute is only important if the *Multi-Reflector GTD* object is used as input (as *Source*) to an analysis procedure where the direction of propagation must be known. This is for example the case in a PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case Ray Output has the same meaning as stated below for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that if the field is used as input to an analysis procedure where this direction is required, it will be assumed to be the direction of Poynting's vector at the field point. Such analysis procedures are e.g. PO with surface materials and PTD.

all

The field is computed as a number of ray fields according to the GO/GTD ray tracing. A subsequent analysis by PO with surface materials or by PTD is then carried out for each ray separately. This is an accurate method but it can be time consuming.

Trace List (*trace_list*) [item from a list of character strings], default: **off**.

All ray traces may be listed in the standard output file by their intersection points with the scatterers. The intersection points are given in the global coordinate system. For blocked rays the blocking scatterer is listed.

off

No list is generated.

on

The list is generated.

Trace File (*trace_file*) [struct].

Ray traces may be stored on file and reused for identical configurations but at other frequencies. The recommended file extension is *.trc*, cf. Section *Ray Traces*.

Status (*status*) [item from a list of character strings], default: **off.**

Determines the storing and possible reusing of ray traces:

off

No file activation. In this case *name* need not be specified.

read

Ray traces are read from the file and reused. No ray tracing is performed and it is not checked whether the trace file agrees with the current case.

generate

Only valid ray traces are stored and reused for the next frequencies specified by the attribute *frequency*.

generate_all_rays

All ray traces – including rays which turned out to be blocked – are stored and reused for the next frequency as specified by the attribute *frequency*.

Name (*name*) [file name].

Name of the file in which the ray trace coordinates are stored. The value is not applied when *status* is specified to 'off'.

Remarks

Scatterers

If no scatterers are defined, the direct field from the illuminating source is calculated with possible blocking from the scatterers as specified in the project.

Ray types

The rays are characterised by their type, the ray type, determined by the interactions as the ray passes to the field point. The ray types are different when the sequence of scatterers or when the sequence of interaction types

along the ray is different. Thus, only one ray type may be handled in each object of class *Multi-Reflector GTD*.

Determination of ray traces

The sequence of the scattering interactions for the rays to be determined are defined in the sequence *scatterers* and each interaction point (reflection or diffraction) may be determined by a numerically search which may be more or less complex.

The simple numerical method is based on the assumption that only one interaction point exists and it is a fast and recommended method. It is applied when *n_points* is specified to zero.

The simple method is always applied for a strut.

If more interaction points may exist (e.g. more reflection points on a reflector surface) then a more thorough numerical method is applied. A number (*n_points*) of sample points are distributed uniformly over the scatterer and a search for a ray path is started from each sample point. Local interaction points may hereby be found.

However, the position of two interaction points following each other on the same ray can not be determined by the simple numerical method and at least every second interaction point must be determined by the more complex method (i.e, having *n_points* defined positive).

Caustics

If the number of rays to a given field point becomes large the field point may be a caustic which means that a small but finite part of the reflecting surface (or the diffracting edge) focuses all rays which hit here to the field point. Caustics can usually be identified as spikes in the pattern. The GO/GTD field computations near caustics will be inaccurate and another method such as PO should be applied instead.

Adding of field contributions of different rays

Often more than one GTD ray trace leads to a given field point. The fields propagating along these rays shall then be added in order to give the total field. The correct way to add the field contributions is to add the complex field values, i.e. to add in amplitude and phase.

For some applications, however, a worst case contribution is desirable. It might be in a case in which the phases of the rays are known to be uncertain because the distance between the scatterers is not known with an accuracy being within a fraction of a wavelength. The highest – worst-case – field contribution is then found by specifying *amplitude_only* to 'on'.

Another example is the determination of the far-field side-lobe level of a reflector which is very large in terms of wavelengths. The far-out side lobes are very narrow and the field must be determined at many points in order to present the lobe maxima. However, the side lobes are in general given by the interference from two – on the reflector opposite situated – diffraction points, i.e. only two rays interfere. The maxima of the side lobes occur when the fields of the two diffractions are in phase and this 'in-phase value' may be determined by specifying *amplitude_only* to 'on'. The 'in-phase value'

value is the upper envelope for the side lobes and this is a slowly varying function which can be determined by a very sparse set of field points.

Reflected and diffracted fields from the same scatterer shall usually be added in amplitude and phase as this results in a smooth field transition across the reflection boundaries.

However, when the worst case interference between two omnidirectional antennas mounted on each side of a spacecraft shall be determined taking into account the scattering in the spacecraft then the following procedure may be applied.

First, the field with adjoined scattering in the spacecraft is determined for the one antenna and next for the other antenna. Finally these two fields shall be added in amplitude, but that can not be done by a *GTD Analysis* object. Instead the two patterns may be stored on a file as power patterns (which is specified in the *Field Storage* object). The fields are then stored in amplitudes and these may be added (or subtracted) by the command *Add Pattern* and the resulting function will be the upper (or lower, respectively) envelope for the interferences between the two patterns.

Limitations

Only ray tracing, which emanates from a specific source point or from a specific direction, can be calculated in this class. This means that the input source must not be a *GTD Analysis* object since the rays should then be traced through the source object as well.

MULTI-GTD

Purpose

This menu defines the tools applied for determining general scattering by GTD.

The scatterers and involved ray traces are defined in a *Source* object of class

Multi GTD.

The scattering itself (such as reflection or diffraction) and the method for the ray tracing are defined in objects of the classes described under

GTD Scatterer.

Links

Classes→*Electrical Objects*→*GTD Analysis*→*Multi-GTD*

MULTI GTD (multi_gtd)

Purpose

The *Source* class *Multi GTD* defines a scattering process to be determined by GO/GTD. One or more scatterers may be involved and the necessary ray tracing to the field point is carried out according to the specifications.

The determination of a scattering (reflection or diffraction) from a scatterer is specified in an object of class *GTD Scatterer*.

This class is only available with the Multi GTD add-on.

Links

[Classes](#)→[Electrical Objects](#)→[GTD Analysis](#)→[Multi-GTD](#)→[Multi GTD](#)

Remarks

Syntax

```
<object name> multi_gtd
(
    frequency           : ref(<n>),
    source              : ref(<n>),
    gtd_scatterers      : sequence(ref(<n>), ...),
    combinations         : struct(type:<si>, from_order:<i>,
                                    to_order:<i>),
    max_searches        : <i>,
    blocking            : <si>,
    blocking_scatterers : sequence(ref(<n>), ...),
    factor              : struct(db:<r>, deg:<r>),
    ray_output          : <si>,
    trace_list          : <si>,
    trace_file          : struct(status:<si>, name:<f>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the field is calculated.

Source (*source*) [name of an object].

Reference to an object of class *Source* (except of PO Convergence as well as objects of class *GTD* or its subclasses). This is the source illuminating the scattering structure.

GTD Scatterers (*gtd_scatterers*) [sequence of names of other objects], default: **blank**.

References to objects of class *GTD Scatterer* in which the reflection or diffraction for selected scatterers is defined. The search for ray traces will be performed in the given sequence of *GTD Scatterer* objects. See the remarks below for considerations in building a *Multi GTD* object.

Combinations (*combinations*) [struct].

Determines the combinations of ray traces to be searched for between the scatterers.

Type (*type*) [item from a list of character strings], default: **sequential**.

Selection of ray type

sequential

Only one ray type will be traced from the source via the respective scatterers (in the sequence given in the attribute GTD Scatterers) to the field point. Values of From Order and To Order below are not used.

all

All scattering combinations of ray traces from the defined From Order and up to and including the defined To Order will be included.

From Order (*from_order*) [integer], default: **0**.

Defining the lowest order of scattering involved in each ray trace.

The order of scattering is defined as follows:

0 The direct field from the source to the field point is calculated.

1 The ray trace involves one scattering.

2 The ray trace involves two scatterings

3 etc...

The value is not applied when Type is set to 'sequential'.

To Order (*to_order*) [integer], default: **0**.

Defining the highest order of scattering involved in each ray trace.

The order of scattering is defined as above. The value is not applied when Type is set to 'sequential'.

Max Searches (*max_searches*) [integer], default: **5**.

The ray tracing process for an actual ray type is stopped when the number of unsuccessful trial paths has reached the maximum number of searches given by Max Searches. The number of unsuccessful trial paths is reset to zero when a ray trace solution has been found or when a new ray type is investigated. If Max Searches is specified to zero, the option is disabled (the search continues). The default value is 5. A higher value is only needed if the user, based on knowledge of the geometry, estimates that not all scattering points have been found in a previous run.

Blocking (*blocking*) [item from a list of character strings], default: **on**.

Some ray traces may be blocked by a scatterer. The check for such blocking is controlled here.

on

All the possible ray traces are checked for blocking in the scatterers referred to in Blocking Scatterers. Blocked ray traces are then discarded.

off

No check for blocking of a ray trace.

Blocking Scatterers (*blocking_scatterers*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Scatterer*. When *Blocking* is specified to 'on' the possible ray traces are checked for blocking in the scatterers specified in this sequence of references.

The name of the reference may be 'all' which comprises references to all *Scatterer* objects of the project.

The value of *Blocking Scatterers* is not used when *Blocking* is specified to 'off'.

Factor (*factor*) [struct].

The total radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Ray Output (*ray_output*) [item from a list of character strings], default: **none**.

This attribute is only important if the *Multi GTD* object is used as input (as *Source*) to an analysis procedure where the direction of propagation must be known. This is for example the case in a PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case Ray Output means the same as stated below for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that if the field is used as input to an analysis procedure where this direction is required, it will be assumed to be the direction of Poynting's vector at the field point. Such analysis procedures are, e.g., PO with surface materials and PTD.

all

The field is computed as a number of ray fields according to the GO/GTD ray tracing. A subsequent analysis by PO with surface materials or by PTD is then carried out for each ray separately. This method is accurate but can be time consuming.

Trace List (*trace_list*) [item from a list of character strings], default: **off**.

All ray traces may be listed in the standard output file by their intersection points with the scatterers. The intersection points are given in the global coordinate system. For blocked rays the blocking scatterer is listed.

off

No list is generated.

on

The list is generated.

Trace File (*trace_file*) [struct].

Ray traces may be stored on file and reused for identical configurations but at other frequencies.

Status (*status*) [item from a list of character strings], default: **off**.

Determines the storing and possible reusing of ray traces:

off

No file activation. In this case Name need not to be specified.

read

Ray traces are read from the file and reused. No ray tracing is performed and it is not checked whether the trace file agrees with the current case.

generate

Only valid ray traces are stored and reused for the next frequencies specified by the attribute Frequency.

generate_all_rays

All ray traces – including rays which turned out to be blocked – are stored and reused in a frequency band as specified by the attribute Frequency.

Name (*name*) [file name].

Name of the file in which the ray trace coordinates are stored.

Not used when Status is specified to 'off'.

Remarks

Ray types

The rays are characterised by their type, the ray type, determined by the interactions as the ray passes to the field point. The ray types are different when the sequence of scatterers or when the sequence of interaction types along the ray is different. Some different ray types for a double reflector system are:

- direct ray (0)
- reflection in the subreflector (1)
- diffraction in the subreflector edge (1)
- diffraction in the subreflector edge followed by reflection in the main reflector (2)
- diffraction in the subreflector edge followed by reflection in the main reflector followed by diffraction in the subreflector edge (3)

The number after each of the ray descriptions is the order of the ray, cf. the attribute *combinations*.

Normally, only a few of the ray types will exist for a given field point but in some cases many ray types occur.

The actual ray types to be included are selected by the attribute *combinations*. If *type* in this attribute is specified to 'sequential' only one ray type is searched for and this ray type is identified by the sequence of scattering processes in attribute *gtd_scatterers*. If *type* in attribute *combinations* is specified to 'all', all ray types of orders between *from_order* and *to_order* (*from_order* and *to_order* inclusive) are traced and the field contributions are found and included.

Thus, to determine the scattered field from a rectangular flat plate including reflection and single diffractions both *from_order* and *to_order* shall be specified to '1' and *type* to 'all'. To include the direct field *from_order* must be specified to '0'. In the latter case we then have the following ray types:

- direct ray,
- reflection in the plate,
- diffraction in edge 1 of the plate,
- diffraction in edge 2,
- diffraction in edge 3, and
- diffraction in edge 4

In total, six ray types.

More rays of same type

For geometries consisting of plane surfaces and straight edges only one ray of a given ray type will exist for a field point. In such cases the number of searches for the ray can be limited to *max_searches*: 1.

Caustics

If the number of rays of the same type at a given field point is large the field point may be a caustic. This occurs when a small but finite region of a reflecting surface (or a diffracting edge) focuses all the rays, hitting the region, to the field point. Caustics can usually be identified as spikes in the pattern. The GO/GTD field computations near caustics will be inaccurate and another method such as PO should be applied instead.

Considerations for building a specific ray

When a scattering environment is built, only important scatterers need to be taken into account. Structures with a size less than 10 wavelengths are usually not suited for field determination by GO/GTD and may be neglected. If structures built by many small parts (e.g. support structures) will scatter the field in many directions, the field level may then become lower than a certain level of interest and the influence of these structures may be ignored. Also highly curved surfaces will scatter the field over a wide range of directions with a low field level as result whereas a plane surface reflects the field without changing the field intensity. Convex surfaces spread the power over wider solid angles and the field level may, again, be low. Concave surfaces (and curved edges), on the other hand, need special attention because such surfaces may focus the field or at least increase the field level at field points of interest.

The GO/GTD calculation may be carried out without much engineering effort by specifying *type* to 'all' in the attribute *combinations*, but if a detailed knowledge of the contributions of the various scatterers is required, the specific ray types may be selected by specifying *type* to 'sequential' and to list the *GTD Scatterers* in the order they are hit by this ray type.

Before the field is determined, rays will be traced according to the specified ray types. The scattering interactions are defined in the sequence *gtd_scatterers* and each scattering point (attributes *reflection* and *diffraction*) may be determined numerically or theoretically (*method:numerical* and *method:theoretical*). When *method* is specified to 'automatic' the program automatically chooses the most effective method.

The theoretical method is based on solving an equation for the angle of reflection (or diffraction) is the same as the angle of incidence. For some *GTD Scatterers*, the method is a simple numerical method taking advantage of the fact that only one interaction point exists. This is a fast method which is recommended when possible.

However, the position of two interaction points following each other on the same ray can not be determined by a theoretical method and at least every second interaction point must be determined numerically which involves a search for the interaction point and is a more time consuming method. An exception to this rule is the reflection in a plane *Plate* where the incoming ray is simply mirrored in the plane and the mirrored ray therefore shall pass straight through the plate. The reflection point may be taken out of the ray tracing procedure and out of the counting for every second interaction point to be determined numerically.

Limitations

Only rays, which emanates from a specific source point or a specific direction, can be traced in this class. This means that the input source must not be a *GTD Analysis* object because the rays should then be traced through the source object as well.

Illustration of rays

An example of the use of *Multi GTD* is shown in the following figure. The case considered is a satellite with a hexagonal body and two rectangular solar panels all modelled by *Plates* apart from the two booms, modelled as *Circular Struts*, carrying the solar panels. The GO/GTD modelling is obtained by applying

```
combinations: struct (type:all, from_order:0, to_order:1)
```

which includes the direct ray (of order 0) and all singly scattered rays (order 1) to the particular field point which here is a far-field direction.

The attribute *blocking* is specified to ‘on’ which causes the ray diffracted in left edge of the right solar panel is blocked by the body of the satellite after the diffraction. It is therefore excluded. In the left solar panel a diffracted ray in each of the four edges of the panel is observed as well as a reflected ray.

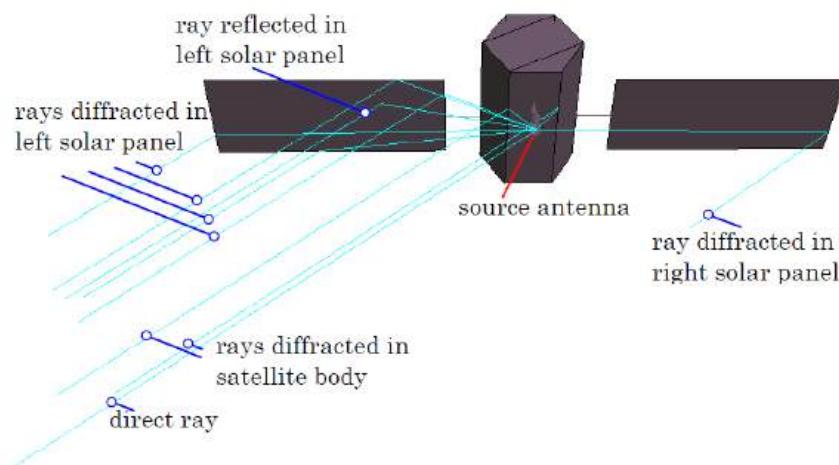


Figure 1

Illustration of rays traced from an antenna on a hexagonal satellite to a far-field point. All ray paths are drawn with the same total length.

Rays diffracted in the satellite body are seen at two vertical edges while the diffraction points in the horizontal edges and the point of reflection all fall outside the extent of the front plate of the satellite body.

GTD SCATTERER

Purpose

The classes defined under *GTD Scatterer* are used to specify scatterings, i.e. reflections or diffractions, to be included in the GO/GTD ray tracing. The way to find the scattering point is also defined.

The complete ray tracing, which may involve several scatterers, is specified in a *Source* object of class *Multi GTD*.

The classes available as *GTD Scatterer* are

GTD Reflector

GTD Plate

GTD Circular Struts

Links

Classes→*Electrical Objects*→*GTD Analysis*→*Multi-GTD*→*GTD Scatterer*

GTD REFLECTOR (gtd_reflector)

Purpose

The class *GTD Reflector* defines the GO/GTD methods to be used for the ray tracing on the specified *Reflector* objects.

This class is only available with the Multi GTD add-on.

Links

Classes→*Electrical Objects*→*GTD Analysis*→*Multi-GTD*→*GTD Scatterer*→*GTD Reflector*

Remarks

Syntax

```
<object name> gtd_reflector
(
    reflectors           : sequence(ref(<n>), ...),
    reflection          : struct(method:<si>, sample_points:<i>,
                                max_reflection_points:<i>),
    diffraction         : struct(method:<si>, edge_index:<i>,
                                sample_points:<i>,
                                max_diffraction_points:<i>,
                                slope:<si>),
    transmission        : <si>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Reflectors (*reflectors*) [sequence of names of other objects].

The search for ray traces is performed in the given sequence of reflectors. The references must all be to objects of the class *Reflector* for which the reflections, diffractions and transmissions shall be computed. The choices of the following attributes apply for all reflectors in the sequence. Other choices for specific rays interacting with the reflectors, or for specific reflectors, must be specified in separate *GTD Reflector* objects.

Reflection (*reflection*) [struct].

Specifies the method determining one or more rays reflected in the reflector surface.

Method (*method*) [item from a list of character strings], default: **automatic**.

The following methods are available:

automatic

Automatic selection of the numerical or the theoretical method.

numerical

The reflection points are found using numerical searches. Each search starts from a sample point, the number of sample points is specified by the struct member *sample_points*. As several reflection points may be found by this method, the number of reflection points (and thus the number of reflected rays) can be limited by the struct member *max_reflection_points*.

theoretical

The reflection point is determined by a simple numerical method. Only one reflection can be found in this way. The values of *sample_points* and *max_reflection_points* are not used.

off

Reflections are not included.

Sample Points (*sample_points*) [integer], default: **120**.

Number of equally distributed sample points on the reflector surface. The points are used as starting points for trial ray traces for the numerical ray tracing process. The total number of trial ray traces for a given ray type is found as the product of the specified values of *sample_points* for all scatterers along the ray trace.

Max Reflection Points (*max_reflection_points*) [integer], default: **-1**.

The maximum number of reflection points on the reflector surface. The search for reflection points is stopped when the number of found reflection points has reached *max_reflection_points*. If negative or zero, the value is disabled and all possible reflection points may be found.

Diffraction (*diffraction*) [struct].

Specifies the method determining the rays diffracted in the reflector edge(s).

Method (*method*) [item from a list of character strings], default: **off**.

The following methods are available:

off

Diffractions are not included.

automatic

Automatic selection of the numerical or the theoretical method.

numerical

The diffraction points are found by numerical searches. Each search starts from a sample point. The number of sample points is specified by the struct member *sample_points*. As several diffraction points may be found by this method, the number of diffraction points (and thus the number of diffracted rays) may be limited by the struct member *max_diffraction_points*.

theoretical

The diffraction points are determined by a simple numerical method. The values of *sample_points* and *max_diffraction_points* are not used.

Edge Index (edge_index) [integer], default: **-1**.

Index of the diffracting edge. The edge numbering follows the numbering of the edges of the reflector rim, as described in the individual *Rim* classes. If the reflector has a central hole, the edge of the hole is given the subsequent edge number. If *edge_index* is specified to -1 all edges, including a possible hole in the reflector, are included.

Sample Points (sample_points) [integer], default: **120**.

Number of equally distributed sample points along the reflector rim. The points are used as starting points for trial ray traces for the numerical ray tracing process. The total number of trial ray traces for a given ray type is found as the product of the specified values of *sample_points* for all scatterers along the ray trace.

Max Diffraction Points (max_diffraction_points) [integer], default: **-1**.

The maximum number of diffraction points along the reflector edge. The search for diffraction points is stopped when the number of found diffraction points has reached *max_diffraction_points*. If negative or zero, the value is disabled and all possible diffraction points may be found.

Slope (slope) [item from a list of character strings], default: **on**.

The slope diffraction can be added in all field calculations.

on

Slope diffraction is included in the diffraction calculations.

off

Slope diffraction is not included.

Transmission (transmission) [item from a list of character strings], default: **off**.

Specifies if transmitted rays should be included. Transmitted rays are rays traces which pass through the reflector at the point of interaction (according to the specified *Electrical Properties* of the reflector).

off

No transmission through the reflector.

on

Rays transmitted through the surface of the reflector are included.

Remarks

When *method* in attribute *reflection* is specified to ‘theoretical’ the reflection point on the reflector surface is found by a simple numerical search starting from the centre of the reflector. No more than one point will be found by this method.

When *method* in attribute *reflection* is specified to ‘numerical’ the reflection points are found by a general numerical search. A reflector may have more than one reflection point for a given ray type. Therefore the numerical search shall be started at various positions, the sample points, equally distributed over the reflector surface. Each sample point is iteratively moved until the ray trace via this point fulfils the reflection law, or until the point moves outside the reflector rim. The number of sample points is given by *sample_points*.

Some sample points may converge to the same reflection point and then only one reflection point is retained. It is the number of retained reflection points which is limited by *max_reflection_points*. Each sample point can give rise to at most one reflection point and the number of reflection points found can therefore never exceed the value of *sample_points*.

In special cases two closely positioned reflection points may exist, but if the value of *sample_points* is specified too low, only one reflection point is found in which case the value of *sample_points* must be increased. In other cases, more reflection points may be found within a small surface area on a slowly varying reflector surface. This indicates that all points within this part of the reflector act as reflection points, i.e. this part of the reflector focuses all its illumination on the field point. This is a caustic and the reflected field cannot be calculated correctly by GO/GTD. The field will be characterized by very high field values within a narrow region.

The diffracted rays are handled like the reflected rays. When the struct member *method* of attribute *diffraction* is specified to ‘theoretical’ the diffraction points are determined theoretically when possible. Otherwise a simple numerical search is applied.

When a *Rectangular Rim* is used in GO/GTD calculations, it should be noted that corner diffractions are not included. The effect of corner diffractions is usually low. The edge-diffracted rays from a rim with straight edges will radiate only in bands (cones) over the far-field sphere. Outside these bands, there will be no diffracted field (of this ray type) and a far-field cut may then show a null field in such regions.

GTD PLATE (gtd_plate)

Purpose

The class *GTD Plate* defines the GO/GTD methods to be used for the ray tracing on the specified *Plate* objects.

This class is only available with the Multi GTD add-on.

Links

Classes→*Electrical Objects*→*GTD Analysis*→*Multi-GTD*→*GTD Scatterer*→*GTD Plate*

Remarks

Syntax

```
<object name> gtd_plate
(
    plates           : sequence(ref(<n>), ...),
    reflection      : <si>,
    diffraction     : struct(method:<si>, edge_index:<i>,
                           sample_points:<i>, slope:<si>),
    transmission    : <si>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plates (*plates*) [sequence of names of other objects].

The search for ray traces will be performed in the given sequence of plates. The references must all be to objects of the class *Plate* for which the reflections or diffractions is to be computed. The choices of the following attributes apply for all plates in the sequence. Other choices for specific rays interacting with the plates, or for specific plates, must be specified in separate *GTD Plate* objects.

Reflection (*reflection*) [item from a list of character strings], default: **on**.

Specifies if reflection shall be included. The reflection points are determined analytically (only one reflection point can exist for each ray type).

on

Rays reflected in the surface of the plate are included.

off

Reflected rays are not included.

Diffraction (*diffraction*) [struct].

Specifies the method for determining the rays diffracted in the edges of the plate.

Method (*method*) [item from a list of character strings], default: **off**.

The following methods are available:

off

Diffractions are not included.

automatic

Automatic selection of the numerical or the theoretical method.

numerical

The diffraction points are found by numerical searches. Each search starts from a sample point. The number of sample points is specified by the struct member *sample_points*.

theoretical

One diffraction point along each of the edges of the plate may exist. It is found by an analytical method. The value of *sample_points* is not used.

Edge Index (*edge_index*) [integer], default: **-1**.

The index of the diffraction edge. The edge numbering follows the numbering of the edges of the plate rim, as described in the individual *Plate* classes. If specified to -1 all edges are included.

Sample Points (*sample_points*) [integer], default: **36**.

Number of equally distributed sample points along the edges of the plate. The points are used as starting points for trial ray traces for the numerical ray tracing process. The total number of trial ray traces for a given ray type is found as the product of the specified values of *sample_points* for all scatterers along the ray trace.

Slope (*slope*) [item from a list of character strings], default: **on**.

The slope diffraction may be added in all field calculations.

on

Slope diffraction is included in the diffraction calculations.

off

Slope diffraction is not included.

Transmission (*transmission*) [item from a list of character strings], default: **off**.

Specifies if transmitted rays shall be included. Transmitted rays are rays traces which pass through the plate at the point of interaction (according to the specified *Electrical Properties* of the plate).

off

No transmission through the plate.

on

Rays transmitted through the surface of the plate are included.

Remarks

Corner diffractions are not included. The effect of corner diffractions is usually small but note that the edge-diffracted rays from the straight edges will radiate only in bands (cones) over the far-field sphere. Outside these bands, there will be no diffracted field and a far-field cut may then show a null field in such regions.

GTD CIRCULAR STRUTS (gtd_circular_struts)

Purpose

The class *GTD Circular Struts* defines the GO/GTD methods to be used for the ray tracing on the specified *Circular Struts* objects.

This class is only available with the Multi GTD add-on.

Links

Classes→*Electrical Objects*→*GTD Analysis*→*Multi-GTD*→*GTD Scatterer*→*GTD Circular Struts*

Remarks

Syntax

```

<object name> gtd_circular_struts
(
    circular_struts          : sequence(
        struct(
            strut_name:ref(<n>),
            index:<i>),
            ...),
        reflection           : struct(method:<si>, sample_points:<i>)
    )
    where
        <i> = integer
        <n> = name of an object
        <si> = item from a list of character strings

```

Attributes

Circular Struts (*circular_struts*) [sequence of structs].

The search for ray traces will be performed in this sequence of circular struts. The struts are, as defined below, specified as struct members in the sequence and each struct member shall consist of a **Strut Name**, which refers to a set of struts defined in an object of class *Circular Struts*, and an **Index**, which refers to the actual strut in the *Circular Struts* object or, when **Index** is specified to -1, to all struts of the object. If not all struts in a *Circular Struts* object are to be included then each strut to be included shall be specified as an individual struct member.

Strut Name (*strut_name*) [name of an object].

Reference to an object of class *Circular Struts*, defining the set of struts in which reflections are determined.

Index (*index*) [integer].

The number of the strut in the set of struts defined in object *Circular Struts*. If specified to -1 (or to 0) all struts in the set are included.

The choices of the following attributes apply for all circular struts in the sequence. Other choices for specific rays interacting with the circular struts, or for specific circular struts, must be specified in separate *GTD Circular Struts* objects.

Reflection (*reflection*) [struct].

Specifies the method for determining the rays reflected in the surface of the struts.

Method (*method*) [item from a list of character strings], default: **automatic**.

The following methods are available:

automatic

Automatic selection of the numerical or the theoretical method.

theoretical

One reflection point may exist in the circular strut. It is found by an analytical method. The value of *sample_points* is not used.

numerical

The reflection points are found using numerical searches. Each search starts from a sample point. The number of sample points is specified by the struct member *sample_points*.

off

Reflections are not included.

Sample Points (*sample_points*) [integer], default: **120**.

Number of equally distributed sample points along and around the strut. The points are used as starting points for trial ray traces for the numerical ray tracing process. The total number of trial ray traces for a given ray type is found as the product of the specified values of *sample_points* for all scatterers along the ray trace.

Remarks

End diffractions are not included. The effect of diffractions in the ends of the strut is usually small as the power is radiated in all directions.

STRUT ANALYSIS

Purpose

The menu *Strut Analysis* contains several classes for the analysis of *Struts*.

The following classes are available:

The analysis of a strut with arbitrary cross section can be defined under

Strut Analysis, Arbitrary Cross Section.

The analysis of a strut with circular cross section can be defined under

Strut Analysis, Circular Cross Section.

The analysis of a strut with polygonal cross section can be defined under

PO, Polygonal Struts (Obsolete).

This class is obsolete and is only included for compatibility reasons.

Links

Classes→*Electrical Objects*→*Strut Analysis*

STRUT ANALYSIS, ARBITRARY CROSS SECTION (strut_analysis_arbitrary_cross)

Purpose

The class *Strut Analysis, Arbitrary Cross Section* is used to specify how the currents on a *Scatterer* of the type *Struts* shall be determined. The currents are stored in the object, and can serve as a source in later calculations.

The class *Strut Analysis, Arbitrary Cross Section* can handle both all perfectly conducting and all dielectric *Struts*.

Links

Classes → *Electrical Objects* → *Strut Analysis* → *Strut Analysis, Arbitrary Cross Section*

Remarks

Syntax

```
<object name> strut_analysis_arbitrary_cross
(
    frequency                  : ref(<n>),
    strut                      : ref(<n>),
    max_mesh_length            : <r>,
    po_points                  : <i>,
    expansion_accuracy         : <si>,
    factor                     : struct(db:<r>, deg:<r>),
    coor_sys                   : ref(<n>),
    file_name                  : <f>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the currents are calculated. The radiated field from the currents can only be calculated at these frequencies. Further, the *Frequency* object must agree with the *Frequency* object to which reference is given in the source object that is used as a source in the *Get Currents* command.

Strut (*strut*) [name of an object].

Reference to a *Struts* scatterer object on which the currents are computed.

Max Mesh Length (*max_mesh_length*) [real number], default: **2**.

Determines the maximum allowed dimension in wavelengths of the segments used to represent the unknown currents around the cross-section of the strut in the analysis. It is recommended to use as large segments as possible, but not larger than 2 wavelengths; a value of Max Mesh Length larger than 3 (wavelengths) is not allowed. If the cross-section is highly curved, a segment size of 2 wavelengths may cause a large geometrical approximation error, and the segment size is therefore automatically decreased to ensure a low geometrical approximation error. It is possible to check if this automatic reduction of the segment size for curved segments is accurate by manually decreasing the value of Max Mesh Length, Remarks belowfor a further discussion of convergence issues.

PO Points (*po_points*) [integer], default: **0**.

Number of current samples in the length direction of the strut (Remarks belowfor details).

Expansion Accuracy (*expansion_accuracy*) [item from a list of character strings], default: **Normal**.

Determines the accuracy of the current expansions, thus also the number of current samples, around the cross-section of the strut. The expansion order is automatically adapted to the electrical size of each segment used to represent the currents. This default level of precision is sufficient for normal accuracy. The expansion order can be increased which may lead to an improved accuracy of the solution but also lead to longer computation time. The normal or the enhanced settings are sufficient for most applications. If large variation of the unknown current occurs on parts of the struts, the high or extreme accuracy setting may improve the solution.

Normal

The default level of precision sufficient for normal accuracy.

Enhanced

Increased polynomial order for enhanced accuracy.

High

Further increased polynomial order for high accuracy.

Extreme

Highest polynomial order for extreme accuracy.

Factor (*factor*) [struct].

The field radiated by the currents will be multiplied by a complex factor,

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system in which the currents are calculated.

File Name (*file_name*) [file name].

Name of the file in which the currents are stored. When a *file_name* is not given, the currents are stored in an internal file, which is deleted at the end of the session. The recommended file extension is *.cur*, cf. the User's Manual.

Command Types

The currents are generated by the command *Get Currents*.

Remarks

The *Strut Analysis, Arbitrary Cross Section* class can handle both all perfectly conducting and all dielectric *Struts*. However, for all perfectly conducting *Circular Struts*, the class *Strut Analysis, Circular Cross Section* is preferred since it is more efficient for this specific case.

Convergence accuracy

The user should always verify that the computed currents are converged to a satisfactory accuracy. This is checked by comparing the result to a more accurate solution.

Three factors contribute to the overall accuracy of the currents calculations:

1. Geometrical discretization error. The currents around the cross-section of the *Struts* scatterers is discretized using curved segments which may be up to 2λ if Max Mesh Length is set to the default value, 2 (wavelengths). The geometrical segments are described by polynomials of up to 3rd order. If the curvature is significant within a single segment this geometrical approximation may be too crude, and the segment size is automatically decreased to reduce this geometrical approximation error. It is possible to check if the automatic decrease of the segment size is accurate by manually decreasing the value of Max Mesh Length. This is done by reducing Max Mesh Length from 2 to, e.g., 1.5 and checking if the result is unchanged. If so, the value of Max Mesh Length of 2 provides sufficiently low geometrical approximation error. If not, Max Mesh Length is gradually reduced until convergent results are obtained.
2. Expansion Accuracy. The current around the cross-section of the strut is expanded in polynomials of up to 10th order. The polynomial order on each segment is automatically selected based on the electrical size of the segment. However, the automatically selected expansion order is intentionally chosen to yield sufficient accuracy when the segment is part of a large smooth segment. If this is not the case, the polynomial

expansion order may need to be increased by changing the settings of the Expansion Accuracy attribute. The 'Normal' and 'Enhanced' settings are adequate for most applications. However, if large variation of the unknown current occurs on parts of the scatterers, 'High' or 'Extreme' settings may be required. For convergence analysis, one should manually change the Expansion Accuracy from 'Normal' to 'Enhanced', from 'Enhanced' to 'High', and from 'High' to 'Extreme', and stop when convergent results are achieved.

3. **Integration Accuracy.** The density of the integration grid must be selected sufficiently high such that the integration will be correct. The density of the integration grid for the currents around the cross-section of the strut is automatically determined such that it is sufficient for the selected Expansion Accuracy attribute. The currents in the length direction of the strut is determined using a PO algorithm. For this current, the density of the integration grid is defined by the PO Points attribute. It is important to perform a convergence test in which PO Points is increased until the field calculation is independent on this parameter. This is done automatically when the auto convergence is switched on in the *Get Currents* command.

Dielectric struts

The currents around the cross-section of the strut are determined using a 2D MoM solver which locally represents the illuminating field as a plane wave. If the strut is illuminated by a source at close proximity this representation may be inaccurate. In such cases inaccurate result may occur for dielectric struts. To obtain an accurate result it is advised to represent the source by a *Plane Wave Expansion*. In principle this applies to a perfectly conducting strut as well, however, the inaccuracy is in this case very small and only occurs for sources positioned at extreme proximity.

STRUT ANALYSIS, CIRCULAR CROSS SECTION (strut_analysis_circ_cross)

Purpose

The class *Strut Analysis, Circular Cross Section* is used to specify how the currents shall be calculated on a *Scatterer* of the type *Circular Struts*. The currents are stored in the object, and can serve as a source in later calculations.

The class *Strut Analysis, Circular Cross Section* can only handle all perfectly conducting *Circular Struts*. For all dielectric *Circular Struts*, one needs to use the *Strut Analysis, Arbitrary Cross Section* class.

Links

Classes→*Electrical Objects*→*Strut Analysis*→*Strut Analysis, Circular Cross Section*

Remarks

Syntax

```
<object name> strut_analysis_circ_cross
(
    frequency           : ref(<n>),
    scatterer          : ref(<n>),
    po_points          : struct(n_length:<i>, n_phi:<i>),
    factor              : struct(db:<r>, deg:<r>),
    spill_over          : <si>,
    ray_output          : <si>,
    coor_sys            : ref(<n>),
    file_name           : <f>,
    obsolete_method     : <si>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the currents are calculated. Further, the *Frequency* class must agree with the *Frequency* class specified in the source object that generates the actual currents.

Scatterer (*scatterer*) [name of an object].

Reference to an object of the *Circular Struts* class defining the struts on which the currents are computed.

PO points (*po_points*) [struct].

The density of the integration grid is defined by the members *n_length* and *n_phi*.

N Length (*n_length*) [integer], default: **0**.

Number of current samples in the length direction of the strut (see the remarks below).

N phi (*n_phi*) [integer], default: **0**.

Number of current samples around the circular cross-section of the strut (see the remarks below).

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Spill Over (*spill_over*) [item from a list of character strings], default: **off**.

Determines if spillover shall be calculated.

off

The spillover is not calculated.

on

The total power W incident on the scatterer is calculated and the spillover is computed as $4\pi/W$. Thus, the result reflects only the true spillover when the feed is power normalised to 4π (see the remarks below). The spillover calculation can double the computation time if the incident field comes from another PO source because both the incident E- and H-field are required in order to compute Poynting's vector.

Ray Output (*ray_output*) [item from a list of character strings], default: **none**.

This attribute defines how the field of the currents shall be calculated if the field is used as input to a GTD or PTD analysis. The attribute is also actual for a subsequent PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case, *ray_output* has the same meaning as stated below for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that the field cannot be used as input to a GTD analysis. In a subsequent PTD analysis the direction of propagation is assumed to be the direction of Poynting's vector at the field point.

all

The field is computed as a ray field with one ray from each current element. A subsequent GTD or PTD analysis is then carried out for each ray separately. This is the most accurate method for GTD/PTD analysis, but it may be time consuming.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system of this source. The currents are calculated in this coordinate system.

File Name (*file_name*) [file name].

Name of file in which the PO and PTD currents are stored. If a *file_name* is not given, the currents are stored in an internal file, which is deleted at the end of the session. The recommended file extension is *.cur*, cf. Section *Standard Currents File*.

Method (Obsolete) (*obsolete_method*) [item from a list of character strings], default: **canonical_po**.

The field scattered from the strut may be calculated by two different methods: a standard PO method (*obsolete_method*: *simple_po*) and a canonical solution for circular cylinders (*obsolete_method*: *canonical_po*). The canonical approach is applicable even for very large strut diameters.

canonical_po

The currents are calculated by using a solution to the canonical problem of a plane wave incident on a circular cylinder.

automatic

If the radius of the strut is less than 2λ , the currents are obtained by using a solution to the canonical problem of a wave incident on a circular cylinder. If the radius is greater than 2λ , ordinary PO currents are calculated on the illuminated part of the strut.

simple_po

The currents on the illuminated part of the struts are calculated as ordinary PO currents.

Command Types

The currents are generated by the command *Get Currents*.

Remarks

The density of the integration grid is defined by the numbers *n_length* giving the number of integration points along the strut axis and *n_phi* giving the number of points around the periphery of the circular cross-section. The number of points necessary for convergence can be estimated by

$$n_length = 3L/\lambda \quad (1)$$

$$n_phi = 8D/\lambda + 4 \quad (2)$$

where *L* is the length of the strut and *D* is the diameter.

It is essential that the number of integration points is chosen so high that the integration will be correct. On the other hand, a too high number may cause an unacceptable long processing time. It is therefore necessary to perform a convergence test in which the parameters *n_length* and *n_phi* are increased until the field calculation is independent on the parameters. This is done automatically when the auto convergence is switched on in the [Get Currents](#) command.

The class [Strut Analysis, Circular Cross Section](#) can only handle all perfectly conducting [Circular Struts](#), whereas [Strut Analysis, Arbitrary Cross Section](#) can handle both all PEC and all dielectric [Circular Struts](#). However, for PEC [Circular Struts](#), the class [Strut Analysis, Circular Cross Section](#) is more efficient.

Spillover

In GRASP the spillover in dB is defined by

$$10 \log(4\pi/W)$$

where *W* is the total power hitting the scatterer. In the GRASP output file, the spillover is printed as well as the relative power hitting the scatterer, *W*/4π. See also the GRAPS9 Technical Description for further information of PO convergence and spillover.

PO, POLYGONAL STRUTS (OBSOLETE) (po_polygonal_struts)

Purpose

The class *PO, Polygonal Struts (Obsolete)* is used to specify how the PO and/or PTD currents shall be calculated on a *Scatterer* of the type *Polygonal Struts*. The currents are stored in the object, and can serve as a source in subsequent field calculations.

The class is included for backwards compatibility only. Instead it is recommended to use the more versatile class *Strut Analysis, Arbitrary Cross Section*.

Struts of circular cross-section may be handled by class *Strut Analysis, Circular Cross Section*.

Links

Classes→*Electrical Objects*→*Strut Analysis*→*PO, Polygonal Struts (Obsolete)*

Remarks

Syntax

```
<object name> po_polygonal_struts
(
    frequency           : ref(<n>),
    scatterer          : ref(<n>),
    method              : <si>,
    po_points           : sequence(
        struct(face:<i>,
                n_length:<i>,
                n_phi:<i>),
        ...),
    ptd_points          : sequence(
        struct(edge:<i>,
                ptd:<i>),
        ...),
    factor              : struct(db:<r>, deg:<r>),
    spill_over          : <si>,
    ray_output          : <si>,
    coor_sys            : ref(<n>),
    file_name           : <f>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the currents are calculated. Further, the *Frequency* class must agree with the *Frequency* class specified in the source object that generates the actual PO or PTD currents.

Scatterer (*scatterer*) [name of an object].

Reference to an object of the *Polygonal Struts* class defining the struts on which the PO currents are computed.

Method (*method*) [item from a list of character strings], default: **po_plus_ptd**.

Determines if the PO and/or PTD contributions are included in the calculations (see the remarks below)

po_plus_ptd

PO as well as PTD contributions are included.

po

Only the PO contribution is included.

ptd

Only the PTD contribution is included.

PO points (*po_points*) [sequence of structs].

A sequence of faces of the strut with specification of the rectangular PO integration grid on the respective faces.

Face (*face*) [integer], default: **-1**.

Number of the actual face of the strut on which the integration grid is specified. The faces of the struts are numbered as described in the *Polygonal Struts* object. The user may specify *face* to -1 in which case all the faces of the struts are included in the analysis.

N Length (*n_length*) [integer], default: **0**.

Number of current samples in the length direction of the strut (see also the remarks below).

N phi (*n_phi*) [integer], default: **0**.

Number of current samples across the face, i.e. perpendicular to the length direction (see the remarks below). When *face* is specified to -1 then *n_phi* specifies the total number of PO points along the circumference of a single strut. The *n_phi* points are distributed on the individual faces according to the width of these faces (see also the remarks below).

Ptd Points (*ptd_points*) [sequence of structs].

A sequence of edges of the strut may be specified for detailing the calculation of the PTD currents along these edges.

Edge (edge) [integer], default: **-1.**

Number of the edge along which the PTD currents shall be calculated. The edges of the struts are numbered as described in the **Polygonal Struts** object. The user may specify *edge* to -1 in which case all edges are included in the analysis.

ptd (ptd) [integer], default: **0.**

Number of samples of the current along the specified *edge* (see also the remarks below). If *edge* is specified to -1, the number applies to each of the edges.

Factor (factor) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (db) [real number], default: **0.**

Amplitude of the factor, in dB.

Phase in degrees (deg) [real number], default: **0.**

Phase of the factor, in degrees.

Spill Over (spill_over) [item from a list of character strings], default: **off.**

Determines if spillover shall be calculated.

off

The spillover is not calculated.

on

The total power W incident on the scatterer is calculated and the spillover is computed as $4\pi/W$. Thus, the result reflects only the true spillover when the feed is power normalised to 4π (see the remarks below). The spillover calculation can significantly increase the computation time if the incident field comes from another PO source because both the incident E- and H-field are required in order to compute Poynting's vector.

Ray Output (ray_output) [item from a list of character strings], default: **none.**

This attribute defines how the field of the currents shall be calculated if the field is used as input to a GTD or PTD analysis. The attribute is also actual for a subsequent PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case, *ray_output* has the same meaning as stated below for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that the field cannot be used as input to a GTD analysis. In a subsequent PTD analysis the direction of propagation is assumed to be the direction of Poynting's vector at the field point.

all

The field is computed as a ray field with one ray from each current element. A subsequent GTD or PTD analysis is then carried out for each ray separately. This is the most accurate method for GTD/PTD analysis, but it may be time consuming.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate Systems* defining the coordinate system of this PO source. The currents are calculated in this coordinate system.

File Name (*file_name*) [file name].

Name of file in which the PO and PTD currents are stored. If a *file_name* is not given, the currents are stored in an internal file, which is deleted at the end of the session. The recommended file extension is *.cur*, cf. Section *Standard Currents File*.

Command Types

The *PO, Polygonal Struts (Obsolete)* is a class under *Strut Analysis*, see under *Strut Analysis* for available commands.

Remarks

The PO and PTD integrations are carried out as specified by *po_points* and *ptd_points*, respectively. Only the faces and edges specified are included. The program detects automatically if a face or an edge is on the shadow side of the strut, in which case the PO currents or the PTD currents, respectively, are set to zero. However, faces or edges that are known to be invisible from the source should be excluded in the calculations in order to save computer time.

This section contains remarks on the following topics:

- Handling of the PO and PTD contributions
- The number of integration points
- Estimates of the parameters *n_length*, *n_phi* and *ptd*
- Spillover

Handling of the PO and PTD contributions

When the PO object is used as the target in a *Get Currents* command, the attribute *method* selects the contributions to be calculated and stored in the currents file:

- If *method* is specified to 'po' or 'po_plus_ptd', then the PO contribution is calculated. If auto-convergence is disabled, then the PO calculation is performed for the faces specified in the attribute *po_points* with the number of PO points specified in the same attribute. If auto-convergence is enabled, then the PO auto-convergence calculation is performed for the faces specified in the attribute *po_points*, independent of the actually specified number of PO points.
- If *method* is specified to 'po_plus_ptd' or 'ptd', then the PTD contribution is calculated. If auto-convergence is disabled, then the PTD calculation is performed for the edges specified in the attribute *ptd_points* with the number of PTD points specified in the same attribute. If auto-convergence is enabled, then the PTD auto-convergence calculation is performed for the edges specified in the attribute *ptd_points*, independent of the actually specified number of PTD points.

When the PO object is used as the source, e.g. in a *Get Field* command, the attribute *method* selects those contributions from the currents file to be included in the numerical integration:

- If *method* is specified to 'po', then only the PO contribution is calculated. The PO calculation is only performed for the faces specified in the attribute *po_points*, even if PO currents for additional faces exist in the file. The PO calculation is performed with the number of PO points available in the file, independent of the actually specified number of PO points in the attribute *po_points*. An error is issued if one of the specified faces is not found in the file. A warning is issued if the file also contains PTD elements.
- If *method* is specified to 'ptd', then only the PTD contribution is calculated. The PTD calculation is only performed for the edges specified in the attribute *ptd_points*, even if PTD elements for additional edges exist in the file. The PTD calculation is performed with the number of PTD points available in the file, independent of the actually specified number of PTD points in the attribute *ptd_points*. An error is issued if one of the specified edges is not found in the file. A warning is issued if the file also contains PO currents.
- If *method* is specified to 'po_plus_ptd', then the PO and the PTD contributions are calculated. The calculation is only performed for the faces and edges specified in the attributes *po_points* and *ptd_points*, even if PO currents for additional faces or PTD elements for additional edges exist in the file. The calculation is performed with the number of PO and PTD points available in the file, independent of the actually specified number of PO and PTD points in the attributes *po_points* and *ptd_points*. An error is issued if one of the specified faces or edges is not found in the file.

The attribute *method* thus yields a simple way of extracting the individual scattering contributions to the total field, as exemplified in the following:

First, calculate the total field by a *Get Currents* command followed by a *Get Field* command. Auto-convergence should be enabled in the *Get Currents*

command, and the attribute *method* in the PO object set to 'po_plus_ptd' to include all scattering effects. Further, the attributes *po_points* and *ptd_points* should have their default settings, selecting all faces and edges of the scatterer. Also, a file name must be specified in the attribute *file_name*, so that the currents calculated in this task is stored in the currents file.

Next, calculate only the PO contribution to the total field. This is achieved by changing the attribute *method* in the PO object to 'po'. The PO contribution could be calculated by re-running the *Get Currents* and *Get Field* commands. However, the same result is obtained by only re-running the *Get Field* command, as the PO currents are simply extracted from the currents file.

Finally, calculate the PTD contribution, by changing *method* to 'ptd', and re-running the *Get Field* command.

The three results (PO+PTD, PO, PTD) were obtained by executing a single and not three *Get Currents* commands, thus significantly reducing the computation time. It is also possible to extract the contribution from a selected number of faces and/or edges by specifying the numbers of these faces and edges in the attributes *po_points* and *ptd_points*. For example to calculate the contribution from edge 2 alone:

Calculate the PTD contribution from edge 2 alone, by selecting *method* to 'ptd', specifying *edge* to 2 in the *ptd_points* attribute, and re-running the *Get Field* command.

The number of integration points

The parameters:

- *n_length* and *n_phi* of the attribute *po_points*, and
- *ptd* in the attribute *ptd_points*

specify numbers of integration points in the PO or PTD integration grids. If one of these parameters is specified to zero then the corresponding contribution is not included in the analysis.

Estimates of the parameters *n_length*, *n_phi* and *ptd*

The number of points in the PO integration grid on the faces of the strut is specified by *n_length* and *n_phi*. It is essential that the number of integration points is so large that the integration will be correct. On the other hand, a too high value may cause an unacceptable long processing time. It is therefore necessary to perform a convergence test in which the parameters *n_length* and *n_phi* are increased until the field calculation is independent on the parameters. This is easiest done by the auto convergence feature of the command *Get Currents* but if the user wants to control the convergence this may be performed by switching the auto converge off in the command *Get Currents*. In that case *n_length*, *n_phi* and *ptd* shall here be specified to non-zero values (in order not to skip the calculations).

The number of integration points across the width of a given face of the strut, *n_phi*, shall be at least 4. If a lower value is specified, *n_phi* will be increased to 4. When all faces are included (*face* is specified to -1), then *n_phi* is the number of integration points for the full periphery of the

strut. In that case *n_phi* will be increased, if necessary, to position at least 4 integration points across each face. If *n_phi* is specified to more than this minimum value, the remaining points will be distributed proportionally to the width of the individually faces. This may increase the actual value of *n_phi*. The chosen distribution of the integration points gives no guarantee for convergence of the PO integration.

The number of PO points necessary for convergence can be estimated by

$$\begin{aligned} n_length &= 3.5L/\lambda \\ n_phi &= 3.5d/\lambda \end{aligned}$$

where L is the strut length, d is the width of the face considered and λ is the wavelength. When the face number is specified to -1, then d shall be the length of the circumference of the strut.

Also for the PTD integration a convergence test shall be carried out. This is automatically included when the auto convergence is switched on in the [Get Currents](#) command.

The number of PTD points necessary for convergence, ptd , can be estimated as *n_length*, i.e.

$$ptd = 3.5L/\lambda$$

The PTD currents are not calculated at the edges perpendicular to the length direction.

Spillover

In GRASP the spillover in dB is defined by

$$10 \log(4\pi/W)$$

where W is the total power hitting the scatterer. In the GRASP output file, the spillover is printed as well as the relative power hitting the scatterer, $W/4\pi$. See also the Technical Description for further information of PO convergence and spillover.

Command Types

The currents on the struts are generated by the command [Get Currents](#). This command is available for all the classes listed above.

OTHER SOURCES

Purpose

The classes of the menu *Other Sources* define special sources from which the radiated field can be calculated.

The members of the menu are:

Plane Wave.

An array of sources:

Array.

The above simple sources may typically be used for illuminating a reflector (or other scatterers). The hereby generated scattered field can also act as a source. These sources are listed below according to the method by which the scattering is calculated.

Some sources may by advantage be converted to a continuous spectrum of plane waves by the

Plane Wave Expansion.

The field radiated from a field distribution defined in a file is given by

Tabulated Planar Source.

Links

Classes→*Electrical Objects*→*Other Sources*

PLANE WAVE (plane_wave)

Purpose

The class **Plane Wave** defines a source, which is a plane wave incident with a specified direction, polarisation, frequency and power density.

Links

[Classes](#)→[Electrical Objects](#)→[Other Sources](#)→[Plane Wave](#)

Remarks

Syntax

```
<object name> plane_wave
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    aperture_radius     : <rl>,
    polarisation        : <si>,
    factor              : struct(db:<r>, deg:<r>),
    frequency_index_for_plot : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a **Frequency** object, defining the frequencies of the plane wave.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the **Coordinate Systems** classes. The phase of the plane wave refers to the origin of this coordinate system and the plane wave propagates along the positive *z*-axis.

Aperture Radius (*aperture_radius*) [real number with unit of length].

Radius of a circular aperture within which the power of the plane wave is normalised to 4π .

Polarisation (*polarisation*) [item from a list of character strings], default: **linear_x**.

The plane wave may be polarised according to several polarisations.

linear_x

The plane wave is linearly polarised along x .

linear_y

The plane wave is linearly polarised along y .

rhc

The plane wave is right hand circularly polarised.

lhc

The plane wave is left hand circularly polarised.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Frequency Index for Plot (*frequency_index_for_plot*) [integer], default: **1**.

Determines the frequency (wavelength) for which the feed shall be plotted. In the attribute Frequency above a sequence of frequencies (wavelengths) is specified. The Frequency Index for Plot points to the number in this sequence to be applied as base for the plot.

Remarks

The *Plane Wave* is normalised by the attribute Aperture Radius so that it radiates a power of 4π within an aperture of that radius. If the attribute Factor is used the power will be increased by the level (in dB) given by this factor.

The radiated field is not limited to within the circular aperture. The *Plane Wave* source can only be used for near-field points.

PLANE WAVE EXPANSION (plane_wave_expansion)

Purpose

The *Plane Wave Expansion* class is used to specify how the radiation from an arbitrary *Source* may be expanded into a spectrum of plane waves. The spectrum is stored in the object, and can serve as a source in later calculations. The primary purpose of this expansion is to improve the accuracy of the analysis of reflectors with special surface materials, e.g. strip grids and dielectric sandwich structures (defined as *Electrical Properties*). It is also useful if the near field immediately in front of a feed or a reflector is needed. See the remarks below for a detailed discussion.

This *Plane Wave Expansion* has certain restrictions on its use. The expansion has no far field and can only be evaluated in the near field. The expansion only includes the visible spectrum i.e. propagating plane waves in a half space $z > 0$. This means that the expansion is only accurate for reasonably directive sources. If e.g. the source has a beam maximum at $\theta = 0^\circ$ the field level must decrease to an insignificant level at $\theta = 90^\circ$.

Links

[Classes](#)→[Electrical Objects](#)→[Other Sources](#)→[Plane Wave Expansion](#)

Remarks

Syntax

```
<object name> plane_wave_expansion
(
    frequency : ref(<n>),
    beam_coor_sys : ref(<n>),
    beam_cone_angle : <r>,
    beam_power : <si>,
    spectral_integration_grid : struct(n_theta:<i>, n_phi:<i>),
    file_name : <f>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the plane-wave expansion is calculated. The radiated field from the expansion can only be calculated at these frequencies. Further, the *Frequency* class must agree with the *Frequency* class specified in the source object that generates the plane-wave expansion.

Beam Coor Sys (*beam_coor_sys*) [name of an object].

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system of the plane-wave expansion. The origin of the coordinate system is not important for the expansion, but the orientation of the *z*-axis must be selected carefully because it defines the central direction of the part of the spectrum which is included in the expansion. Normally, the direction of the *z*-axis should be given by the far-field beam maximum of the source object generating the plane-wave expansion.

Beam Cone Angle (*beam_cone_angle*) [real number].

Angular extent of the spectrum used in the plane-wave expansion. The part of the spectrum included in the expansion is given by a cone in the far field with axis along the *z*-axis of Beam Coor Sys and a half-cone angle given by Beam Cone Angle (in degrees). See also the remarks below. The Beam Cone Angle must be greater than zero and less than or equal to 90° degrees.

Beam Power (*beam_power*) [item from a list of character strings], default: **on**.

If specified to 'on' the total radiated power in the plane-wave expansion is calculated. This is useful for checking that the expansion is valid and has captured a sufficiently large part of the power radiated by the source.

on

the total power contained in the plane-wave expansion is calculated and displayed.

off

the power in the plane-wave expansion is not calculated.

Spectral Integration Grid (*spectral_integration_grid*) [struct].

The density of the plane waves in the spectral integral is defined by the members *n_theta* and *n_phi*. Normally the user does not have to care about these numbers due to the automatic convergence test in the *Get Plane Wave Expansion* command.

N theta (*n_theta*) [integer], default: **10**.

Number of samples along θ in the spectral integral, $\theta = 0^\circ$ and $\theta =$ Beam Cone Angle included.

N phi (*n_phi*) [integer], default: **10**.

Number of samples along ϕ , $0^\circ \leq \phi < 360^\circ$, in the spectral integral.

File Name (*file_name*) [file name].

Name of file in which the plane-wave expansion is stored. When a File Name is not given, the waves are stored in an internal file, which is deleted at the end of the session. The recommended file extension is .pwe.

Command Types

The expansion into plane waves is activated by the command

Get Plane Wave Expansion.

Remarks

This section contains guidelines for the choice of the attribute Beam Cone Angle followed by some examples of the use of the *Plane Wave Expansion*.

The examples are further elaborated in the Tutorial Examples.

The headings of the sections are:

- Determination of the Beam Cone Angle
- Example: Near field from a feed
- Example: Polarisation grid
- Example: Near field from a reflector
- How the reflector near field was determined.

Determination of the Beam Cone Angle

The field of a source may be expanded into plane waves propagating into a forward hemisphere from a source. The centre of the hemisphere is here given by the z -axis of the Beam Coor Sys. The plane-wave expansion is only accurate if the far field is reasonably directive and has decreased to a low level at $\theta = 90^\circ$.

A further truncation is possible by means of the attribute Beam Cone Angle which limits the spectrum to the far-field cone $\theta \leq \theta_0$, θ_0 being the Beam Cone Angle. Such a truncation will improve the computational speed and is convenient if it is known that nearly all power is contained within $\theta \leq \theta_0$. The angle θ_0 is measured from the z -axis of the coordinate system specified in Beam Coor Sys. For that reason the orientation of this coordinate system is important and the z -axis should point towards the beam maximum of the far-field pattern to capture as much power as possible within the cone $\theta \leq \theta_0$.

The half-cone angle (Beam Cone Angle) shall be sufficiently large to include the near-field grid upon which the field later shall be determined from the plane-wave expansion. The criterion is that rays drawn from the rim of the antenna aperture to the outer edge of the cone shall fully encompass the near-field grid, as illustrated in the following figure.

The convergence properties of a plane-wave expansion is rather different from that of induced or equivalent currents (see *PO Analysis*). When a field is represented by currents, the field can be calculated in the far field as well as in the near field except on the surface on which the currents are located. If the observation point approaches this surface very many current elements will be needed and the convergence becomes very slow. As opposed to the currents, the plane-wave expansion is most rapidly converging close to the source, and it has no far field. An increasing number of samples is needed in the numerical integration as the observation point moves away from the source. The plane-wave expansion is thus a useful tool if the near field is needed very close to a source. The following examples illustrate the most

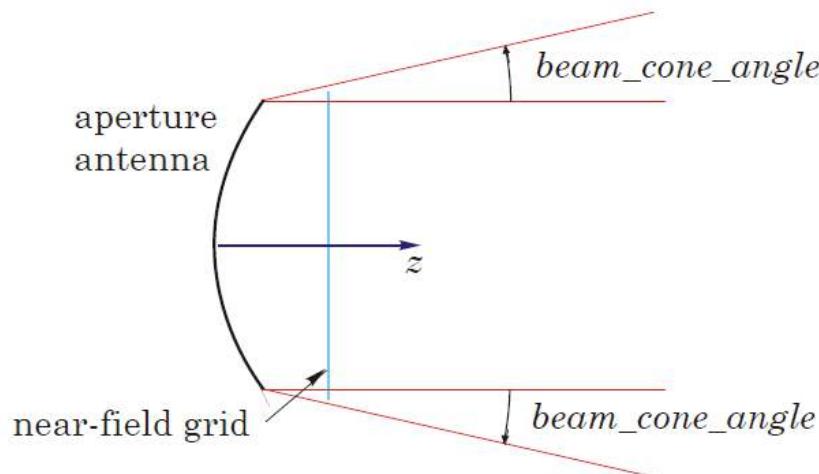


Figure 1 The (Beam Cone Angle shall be large enough to enclose the complete near-field grid upon which the field shall be evaluated by the plane-wave expansion.

important properties of the plane-wave expansion. The examples are further elaborated in the Tutorial Examples.

Example: Near field from a feed

This example illustrates some basic properties of the plane-wave expansion. A feed radiating an ideal Gaussian beam is shown below in Figure 2. The beam has a far-field taper of -12 dB at $\theta = 30^\circ$ corresponding to a waist radius of $w_0 = 0.704\lambda$.

The near field is now calculated from the far field by means of the plane-wave expansion. Since the feed pattern is a simple Gaussian beam the exact near field is also known and can be compared to the result of the plane-wave expansion.

The near field is calculated on a line 1λ in front of the feed aperture (shown as a light blue line in Figure 2) and the results are compared in Figure 3 with different truncation angles θ_0 of the plane-wave expansion. It is seen that a truncation of the far field (spectrum) at $\theta_0 = 40^\circ$ gives an accurate reconstruction of the near field down to approximately 35 dB below peak and that larger truncation angles rapidly improves the accuracy.

The power of the plane-wave spectrum is output when Beam Power is set to 'on'. This power is given relative to the full power (4π watt) of the horn in the table 1. The power value is a useful tool in the evaluation of the accuracy of the spectrum and thus of the determined near-field.

θ_0	40°	50°	60°
Relative power	0.9921	0.9994	0.99997

Table 1 The power in the plane-wave expansion for different truncation angles relative to the total power of the feed.

Example: Polarisation grid

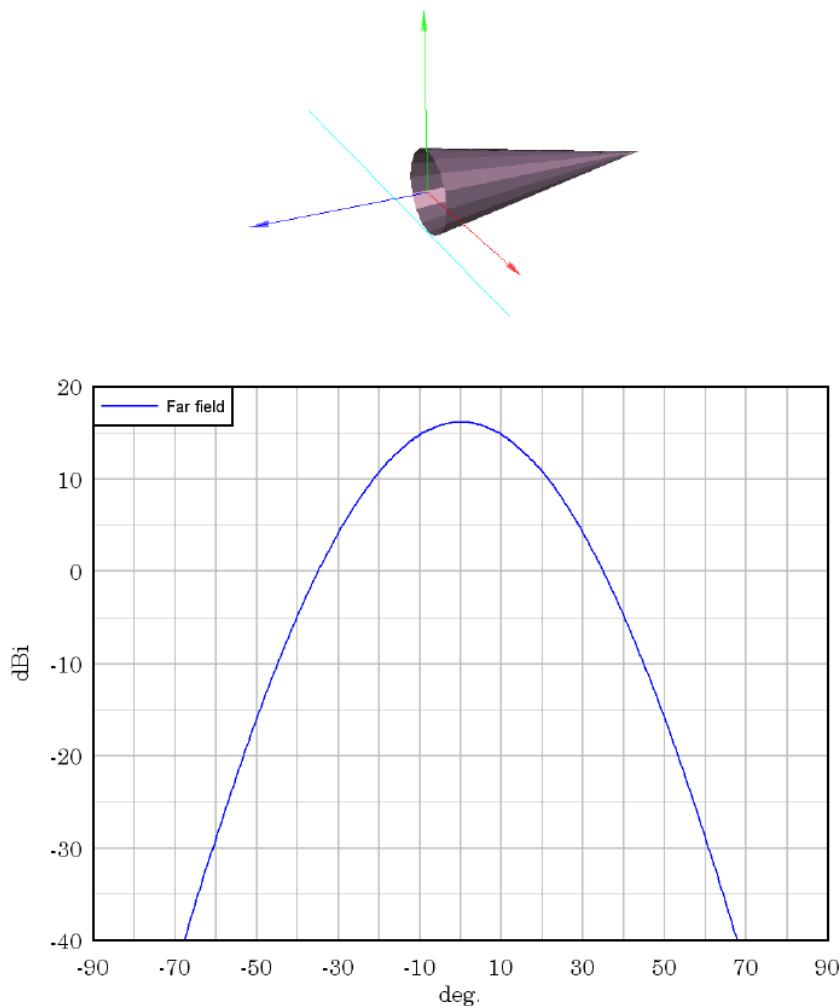


Figure 2 Feed horn and the near-field cut shown as a line in light blue (top) and far-field pattern (bottom).

In this example the plane-wave expansion is used to improve the accuracy of the PO-analysis of non-perfectly conducting reflector surfaces. The analysis by PO of scatterers with special surface materials (e.g. polarisation grids and dielectric layers, see the classes *Electrical Properties*) is based on reflection and transmission coefficients given for an infinite planar surface. The approximations introduced are that the surface can be considered to be locally planar and that the incident field locally is a plane wave. The first approximation is normally satisfied because the radius of curvature of the surface is normally much larger than the wavelength. The second approximation is more problematic, because the incident field may have a complicated behavior as e.g. close to a beam waist. In this case it is important to expand the incident field in plane waves which are then treated separately in the PO analysis of the surface.

An example is shown below in Figure 4. A Gaussian beam is radiated by the feed and reflected by a solid ellipsoidal mirror. Hereafter the beam reaches an ideal polarisation grid where most of the power is reflected, since the polarisation of the Gaussian beam and the direction of the wires in the grid are both orthogonal to the plane of symmetry of the system (i.e. the plane of the beam path).

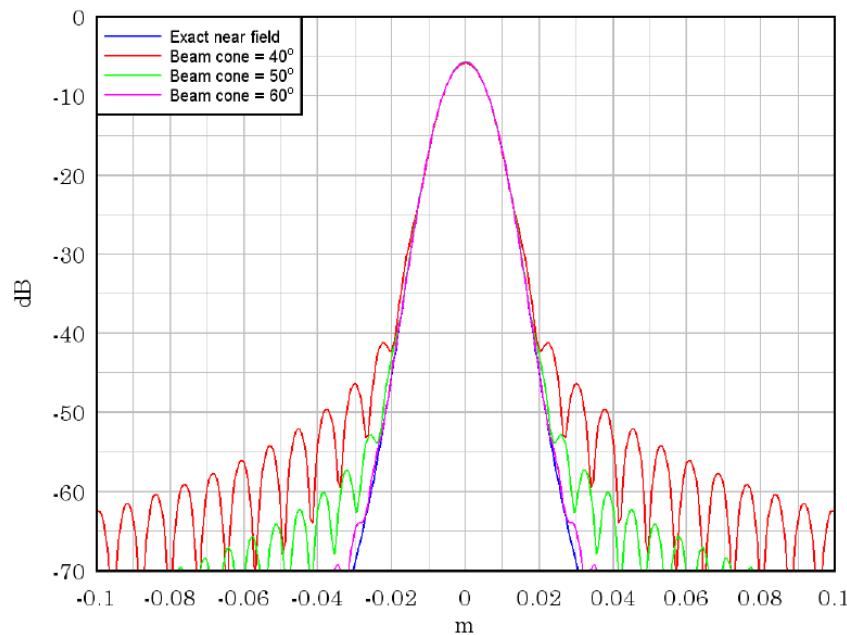


Figure 3 Near field from a Gaussian beam feed calculated by means of plane-wave expansion and compared to the exact Gaussian beam near field. The wavelength is 1 cm.

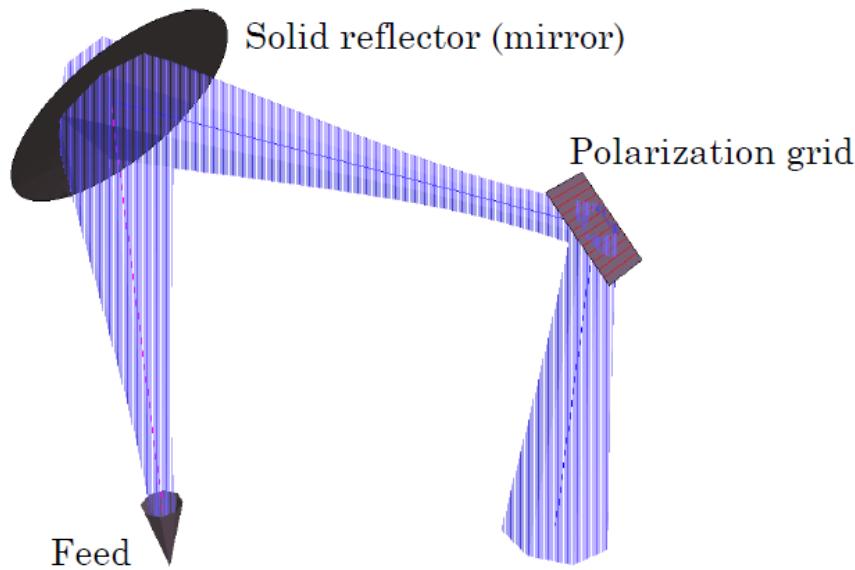


Figure 4 An ideal polarisation grid located close to a beam waist

A minor part of the field is, however, transmitted through the grid. Ideally, this field should be zero, but since the incident field is not a single perfect plane wave and since the solid reflector generates some crosspolarisation the grid is not fully able to block for transmission. The results for the transmitted field are shown in Figure 5, computed with the standard PO method and with an expansion of the incident field in plane waves. The polarisation component shown is parallel to the wires of the grid and is thus not caused by the cross-polarisation of the solid reflector. It is seen that the correct result obtained by the plane-wave expansion is about 30 dB lower than by

the standard PO method. If the grid is placed so far away from the waist that the incident field can be considered as a spherical wave, the agreement between the two methods becomes much better. In general we recommend using a plane-wave expansion of a source when the source illuminates a reflector with non perfectly conducting surface materials. The standard PO method is, however, adequate when the reflector is clearly in the far field of the source.

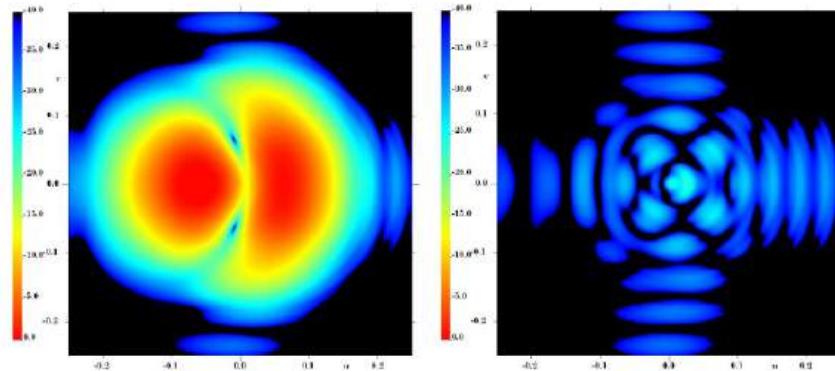


Figure 5

Transmitted field computed by standard PO (left) and by plane-wave expansion (right).

The beam in Figure 4 is reflected at a right angle at the ellipsoidal mirror and the feed and polarisation grid are located in the two foci of the ellipsoid. These foci are both located at a distance of 200 cm from the centre of the mirror. The feed is an ideal Gaussian beam with waist radius of $w_o = 4.17$ cm radiating with a wavelength of 1 cm. For the plane-wave expansion the Beam Cone Angle is selected as 15° which captures the fraction 0.999997 of the radiated power from the reflector.

Example: Near field from a reflector

This example illustrates that the near field very close to a large reflector can be computed efficiently by a plane-wave expansion. A rotationally symmetric parabolic reflector is shown below in Figure 6. The diameter of the reflector and the focal length are both 2 m and the wavelength is 0.005 m, corresponding to a frequency close to 60 GHz. The feed is a Gaussian beam with an edge taper of -12 dB. The near field is calculated on a line (shown in light blue in Figure 6) located 0.15 m in front of the centre of the reflector corresponding to 0.025 m in front of the plane of the rim.

The advantage of using the plane-wave expansion in the close near field is that it is much faster (in the present example a factor of 40) than PO. This may be important for e.g. calculating the field incident on struts. When the integration of the PO current shall converge at field points close to some of the current elements then a dense integration grid is required, and the integration will be time consuming. By means of the plane-wave expansion only a sparse integration grid is needed and the integration is much faster. That is because the plane-wave expansion is based on the farfield within the specified Beam Cone Angle and the integration grid of the PO-currents may be an open grid when the field shall be determined in these quasi-focused

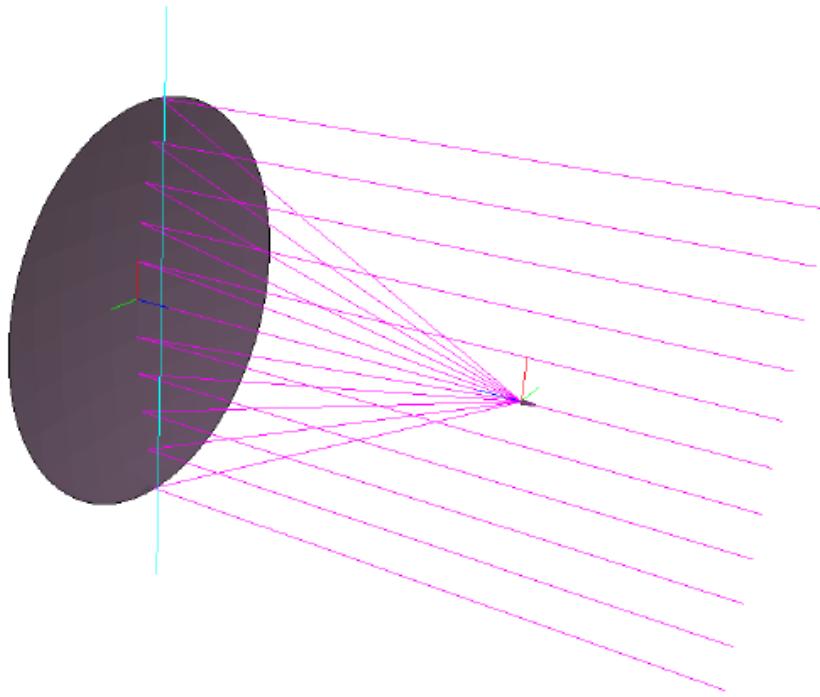


Figure 6 Near-field from a parabolic reflector.

far-field directions. In next turn the near field shall be determined from the plane-wave expansion which is a fast procedure.

The results are shown in Figure 7. It is seen that the agreement between the two methods is very good inside the aperture, but that the details of the diffraction lobes outside the aperture differ. Further, a small dip at $\theta = 0^\circ$ due to a caustic from the edge diffracted rays is correctly described by PO but missing in the plane-wave expansion. The reason for these small inaccuracies is that they are caused by diffracted rays with an angle from boresight much larger than the Beam Cone Angle, here selected to 5° . The fraction of power inside this cone is 0.9990 compared to total radiated power of the reflector.

How the reflector near field was determined

The following short exposition is explained in more details in the Tutorial Examples.

The standard way to determine a near field from a scatterer is to apply **PO, Single-Face Scatterer**. It is by this method the pattern denoted 'PO near field' in Figure 7 is generated. The method runs as follows: The **Feed** of the reflector is defined as the **Source** illuminating the reflector on which currents are induced. The reflector is referenced as the scatterer in the **PO, Single-Face Scatterer** object and it is in this object that the currents are stored after being generated by a **Get Currents** command. In the **Get Currents** command **convergence_on_output_grid** shall be specified to the **Planar Cut** in which the near field is to be determined. The near field is, finally, determined by a **Get Field** command with the currents in the **PO, Single-Face Scatterer** object as source.

The **Plane Wave Expansion** adds a step in this computation of the near field

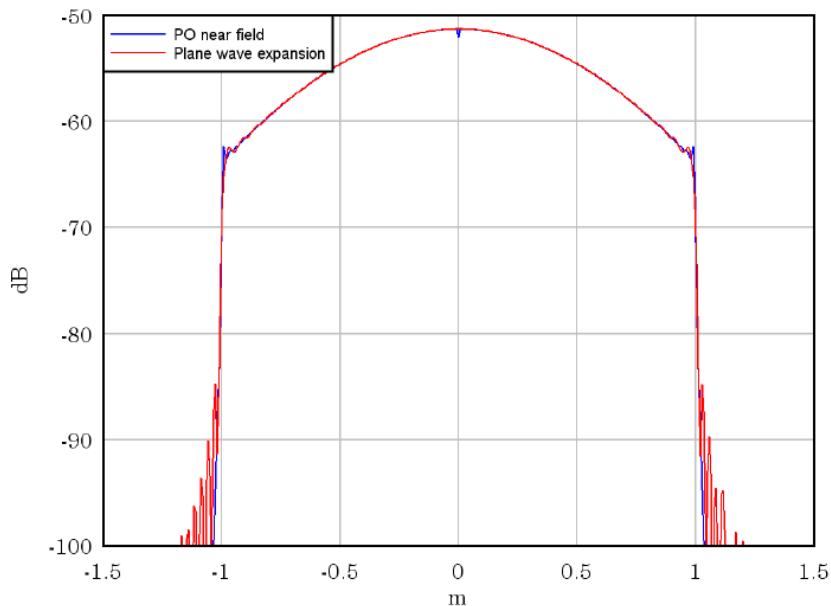


Figure 7

Comparison of PO and plane-wave expansion in the near field of a parabolic reflector.

of a reflector. Again currents induced by the *Feed* are determined and stored in a *PO, Single-Face Scatterer* object by a *Get Currents* command. Now the currents shall be applied to generate a plane-wave expansion and in the *Get Currents* command it is *convergence_on_expansion_grid* which shall be specified to a *Plane Wave Expansion* object defining the region of validity for the expansion (by beam direction and cone angle). When the currents have been generated and stored in the *PO, Single-Face Scatterer* object then the plane-wave expansion is generated by a *Get Plane Wave Expansion* command in which the *PO, Single-Face Scatterer* object is referenced as the source and *convergence_on_near_field_grid* is referencing to the *Planar Cut* in which the plane-wave expansion shall be applied. The determined plane-wave spectrum is stored in the *Plane Wave Expansion* object and by a final *Get Field* command with this *Plane Wave Expansion* object as source the nearfield cut is determined.

ARRAY**Purpose**

In the group *Array* it is possible to describe arrays of sources. The following classes are available:

General Array

Tabulated General Array

Planar Grid Array

Tabulated Planar Grid Array

Links

Classes→*Electrical Objects*→*Other Sources*→*Array*

GENERAL ARRAY (general_array)

Purpose

The class *General Array* defines an array of source elements where the position, orientation and radiation properties of each element may be individually specified. Data are entered directly through the attributes.

If the specification data are available on a file, the class *Tabulated General Array* may be used instead.

When the elements are arranged in a regular grid the class *Planar Grid Array* or the class *Tabulated Planar Grid Array* may be applied.

Links

Classes→*Electrical Objects*→*Other Sources*→*Array*→*General Array*

Remarks

Syntax

```

<object name> general_array
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    elements             : sequence(
                                struct(id:<s>,
                                       x:<rl>,
                                       y:<rl>,
                                       z:<rl>,
                                       theta:<r>,
                                       phi:<r>,
                                       psi:<r>,
                                       source:ref(<n>),
                                       beam_id:<s>),
                                ...),
    element_composite   : <si>,
    exc                  : sequence(
                                struct(id:<s>,
                                       db:<r>,
                                       deg:<r>),
                                ...),
    factor               : struct(db:<r>, deg:<r>),
    ray_output            : <si>
)
where
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<si> = item from a list of character strings

```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the array should radiate.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate System* defining the coordinate system (the array coordinate system) in which the array and its elements are defined.

Elements (*elements*) [sequence of structs].

Each struct in this sequence defines a source element in the array (see the remarks for positioning of the elements).

Id (*id*) [character string].

Alphanumeric identification string of the source element. Each element in the array must be given an identification, which is unique among the elements of this array.

x (*x*) [real number with unit of length].

x-coordinate of the position of the source element in the array coordinate system.

y (*y*) [real number with unit of length].

y-coordinate of the position of the source element in the array coordinate system.

z (*z*) [real number with unit of length].

z-coordinate of the position of the source element in the array coordinate system.

theta (*theta*) [real number].

θ -orientation (in degrees) of the source element. See the remarks below.

phi (*phi*) [real number].

ϕ -orientation (in degrees) of the source element. See the remarks below.

psi (*psi*) [real number].

ψ -orientation (in degrees) of the source element. See the remarks below.

Source (*source*) [name of an object].

Reference to a source object (an object of class *Source*) specifying the source used as the array element. The source is inserted together with its own coordinate system such that this coincides with the element position and orientation. Note that the syntax allows for recursive definitions, i.e. a sub-array can easily be defined.

Beam Id (*beam_id*) [character string], default: `.`.

This struct member is only used if the above source refers to another array for which the attribute *element_composite* has been set to 'element'. In this case, the string must contain the beam identifier for that beam which is used here as array element. In all other cases, the *beam_id* must be an empty string (' ').

Element Composite (*element_composite*) [item from a list of character strings], default: **element**.

Determines if the array field shall be calculated on an element-by-element basis or as a composite array.

element

The array radiation is calculated element by element without using the excitation coefficients. The computed field will contain a number of beams equal to the number of elements in the array.

composite

The array radiation is computed as a composite field. Each beam is multiplied by the excitation coefficient of the element and added up to a total field.

Exc (*exc*) [sequence of structs].

Excitation coefficients of the array elements. This attribute is only required if *element_composite* is set to composite. Each struct in the sequence defines an excitation coefficient for one of the array elements. This sequence must not contain excitation coefficients for elements, which are not present in the *elements* attribute. If, on the other hand, an excitation coefficient is not specified for some of the elements, the excitation coefficients of these elements are set to zero (the elements are switched off).

Id (*id*) [character string].

Element identifier which associates the specified excitation to a particular array element.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the excitation, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the excitation, in degrees.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Ray Output (*ray_output*) [item from a list of character strings], default: **none**.

This attribute defines how the array field shall be calculated if the field is used as input to a GTD or PTD analysis. The attribute is also actual for a subsequent PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis).

of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case, *ray_output* shall be specified as stated for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that the field cannot be used as input to a GTD analysis. In a subsequent PTD analysis the direction of propagation is assumed to be the direction of Poynting's vector at the field point.

all

The field is computed as a ray field with one ray from each array element (including sub-array elements). A subsequent GTD or PTD analysis is then carried out for each ray separately. This is the most accurate method for GTD/PTD analysis, but it may be time consuming.

spherical

The field from the elements will be treated as a single ray which emanates from the origin of the array coordinate system (specified by *coor_sys*). This procedure is only accurate if the scatterer in the following GTD/PTD analysis is in the far field from the array or the array elements are phased such that they radiate a field at the scatterer with a spherical phase front centred at the origin.

plane

The field from the elements will be treated as a single ray which is parallel to the z -axis of the array coordinate system (specified by *coor_sys*). This procedure is only accurate if the elements radiate a field in the direction of the z -axis of the array coordinate system and if this field has a nearly plane phase front at the scatterer to be analysed by GTD/PTD.

Remarks

Each element has a position (x, y, z) and an orientation (θ, ϕ, ψ) given by the members x, y, z and *theta*, *phi*, *psi*, respectively, of the attribute *elements*.

The position (x, y, z) defines the origin of the element coordinate system in the array coordinate system.

The angles (θ, ϕ, ψ) define the orientation of the element coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ through

$$\hat{x}_\ell = \hat{\theta} \cos(\phi - \psi) - \hat{\phi} \sin(\phi - \psi)$$

$$\hat{y}_\ell = \hat{\theta} \sin(\phi - \psi) + \hat{\phi} \cos(\phi - \psi)$$

$$\hat{z}_\ell = \hat{r}$$

where $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ are the base vectors of the axes of the element coordinate system and $\hat{r}, \hat{\theta}, \hat{\phi}$ are the standard spherical vectors in the array coordinate system,

$$\begin{aligned}\hat{r} &= (\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta) \\ \hat{\theta} &= (\cos \theta \cos \phi, \cos \theta \sin \phi, -\sin \theta) \\ \hat{\phi} &= (-\sin \phi, \cos \phi, 0)\end{aligned}$$

The coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ may be obtained by tilting the z -axis (and the full array coordinate system) the angle θ towards ϕ , and then rotate the system the angle ψ around the z -axis. More details are given in the Technical Description in the section on Coordinate Systems.

Each source (element) in the array is defined in its own coordinate system such that this coordinate system coincides with the element coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ defined here.

The field of the array is determined element by element as near field or far field as described under the respective *Source* elements.

TABULATED GENERAL ARRAY (tabulated_general_array)

Purpose

The class *Tabulated General Array* defines an array of source elements where the position, orientation and radiation properties of each element may be individually specified. The element data are read from a data file.

If it is preferred to specify the data directly through the attributes, the class *General Array* may be used instead.

When the elements are arranged in a regular grid the class *Planar Grid Array* of the class *Tabulated Planar Grid Array* may be applied.

Links

Classes→*Electrical Objects*→*Other Sources*→*Array*→*Tabulated General Array*

Remarks

Syntax

```
<object name> tabulated_general_array
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    elements_file       : <f>,
    element_composite  : <si>,
    exc_file            : <f>,
    factor              : struct(db:<r>, deg:<r>),
    ray_output          : <si>
)
where
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the array should radiate.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate Systems* defining the coordinate system (the array coordinate system) in which the array and its elements are defined.

Elements File (*elements_file*) [file name].

Name of file in which the element position, orientation and definition are specified (see the remarks for positioning of the elements). See *Irregularly Spaced Feed Data* for a description of the format of this file.

Element or Composite (*element_composite*) [item from a list of character strings], default: **element**.

Determines if the array field shall be calculated on an element-by-element basis or as a composite array.

element

The array radiation is calculated element by element without using the excitation coefficients. The computed field will contain a number of beams equal to the number of elements in the array.

composite

The array radiation is computed as a composite field. Each beam is multiplied by the excitation coefficient of the element and added up to a total field.

Excitation Coef. File (*exc_file*) [file name].

Name of the file from which the excitation coefficients of the elements are read. This file is only required if Element or Composite is set to 'composite'. The format of the file is described in Section *Array Element Excitation Coefficients*.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Ray Output (*ray_output*) [item from a list of character strings], default: **none**.

This attribute defines how the array field shall be calculated if the field is used as input to a GTD or PTD analysis. The attribute is also actual for a subsequent PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case, *ray_output* has the same meaning as stated for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that the field cannot be used as input to a GTD analysis. In a subsequent PTD analysis the direction of propagation is assumed to be the direction of Poynting's vector at the field point.

all

The field is computed as a ray field with one ray from each array element (including sub-array elements). A subsequent GTD or PTD analysis is then carried out for each ray separately. This is the most accurate method for GTD/PTD analysis, but it may be time consuming.

spherical

The field from the elements will be treated as a single ray which emanates from the origin of the array coordinate system (specified by Coordinate System). This procedure is only accurate if the scatterer in the following GTD/PTD analysis is in the far field from the array or the array elements are phased such that they radiate a field at the scatterer with a spherical phase front centred at the origin.

plane

The field from the elements will be treated as a single ray which is parallel to the z -axis of the array coordinate system (specified by Coordinate System). This procedure is only accurate if the elements radiate a field in the direction of the z -axis of the array coordinate system and if this field has a nearly plane phase front at the scatterer to be analysed by GTD/PTD.

Remarks

Each element has a position (x, y, z) and an orientation (θ, ϕ, ψ) given by the data x, y, z and *theta, phi, psi*, respectively, in Elements File.

The position (x, y, z) defines the origin of the element coordinate system in the array coordinate system.

The angles (θ, ϕ, ψ) define the orientation of the element coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ through

$$\begin{aligned}\hat{x}_\ell &= \hat{\theta} \cos(\phi - \psi) - \hat{\phi} \sin(\phi - \psi) \\ \hat{y}_\ell &= \hat{\theta} \sin(\phi - \psi) + \hat{\phi} \cos(\phi - \psi) \\ \hat{z}_\ell &= \hat{r},\end{aligned}$$

where $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ are the base vectors of the axes of the element coordinate system and $\hat{r}, \hat{\theta}, \hat{\phi}$ are the standard spherical vectors in the array coordinate

system,

$$\begin{aligned}\hat{r} &= (\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta) \\ \hat{\theta} &= (\cos \theta \cos \phi, \cos \theta \sin \phi, -\sin \theta) \\ \hat{\phi} &= (-\sin \phi, \cos \phi, 0).\end{aligned}$$

The coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ may be obtained by tilting the z -axis (and the full array coordinate system) the angle θ towards ϕ , and then rotate the system the angle ψ around the z -axis. More details are given in the Technical Description in the section on Coordinate Systems.

Each source (element) in the array is defined in its own coordinate system such that this coordinate system coincides with the element coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ defined here.

The field of the array is determined element by element as near field or far field as described under the respective *Source* elements.

PLANAR GRID ARRAY (planar_grid_array)

Purpose

The class *Planar Grid Array* defines an array of source elements where the elements can be arranged in planar hexagonal or rectangular grids. Data are entered directly through the attributes.

If the specification data are available on a file, the class *Tabulated Planar Grid Array* may be used instead.

When the elements are not arranged in a regular grid the class *General Array* or the class *Tabulated General Array* may be applied.

Links

Classes→*Electrical Objects*→*Other Sources*→*Array*→*Planar Grid Array*

Remarks

Syntax

```

<object name> planar_grid_array
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    grid_topology       : struct(grid_type:<si>, spac_ratio:<r>),
    grid_position       : struct(grid_cenx:<rl>, grid_ceny:<rl>,
                                grid_rot:<r>, grid_spac:<rl>),
    elements            : sequence(
        struct(id:<s>,
               mx:<i>,
               my:<i>,
               theta:<r>,
               phi:<r>,
               psi:<r>,
               source:ref(<n>),
               beam_id:<s>),
        ...),
    element_composite  : <si>,
    exc                 : sequence(
        struct(id:<s>,
               db:<r>,
               deg:<r>),
        ...),
    factor              : struct(db:<r>, deg:<r>),
    ray_output          : <si>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length

```

`<s>` = character string

`<si>` = item from a list of character strings

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the array should radiate.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* defining the coordinate system (the array coordinate system) in which the array and its elements are defined.

Grid Topology (*grid_topology*) [struct].

Defines the topology of the array grid.

Grid Type (*grid_type*) [item from a list of character strings], default: **hexagonal**.

The array elements may be organised in a regular hexagonal or rectangular grid and *grid_type* may have one of two values.

hexagonal

When the grid is hexagonal, the angle between the first and second grid direction is 60°.

rectangular

In a rectangular grid the angle between the first and second grid direction is 90°.

Spac Ratio (*spac_ratio*) [real number], default: **1**.

The element spacing in the second grid direction relative to the element spacing in the first grid direction (*grid_spac*, see the attribute *grid_position*).

Grid Position (*grid_position*) [struct].

Determines the position and orientation of the grid in which the array element positions are defined.

Grid Cenx (*grid_cenx*) [real number with unit of length].

The position of the reference point in the grid is at (x,y) = (*grid_cenx*, *grid_ceny*) in the *xy*-plane of the array coordinate system. The reference point must be a node in the grid (see also the figure in the remarks below).

Grid Ceny (*grid_ceny*) [real number with unit of length].

The position of the reference point in the grid is at (x,y) = (*grid_cenx*, *grid_ceny*) in the *xy*-plane of the array coordinate system. The reference point must be a node in the grid (see also the figure in the remarks below).

Grid Rot (*grid_rot*) [real number].

Rotation angle of the grid in the *xy*-plane of the array coordinate system around the grid reference point, measured positive in degrees from the *x*-axis towards the *y*-axis. See also the figure in the remarks below.

Grid Spac (*grid_spac*) [real number with unit of length].

Spacing between elements along first direction in the grid (parallel to the *x*-axis for *grid_rot* specified to zero).

Elements (*elements*) [sequence of structs].

Each struct in this sequence defines a source element in the array (see the remarks for positioning of the elements).

Id (*id*) [character string].

Alphanumeric identification string of the source element. Each element in the array must be given an identification, which is unique among the elements of this array.

mx (*mx*) [integer].

Position of the source element in the grid defined elsewhere, given as indices of the element along the two grid directions.

my (*my*) [integer].

Position of the source element in the grid defined elsewhere, given as indices of the element along the two grid directions.

theta (*theta*) [real number].

Orientation of the source element, in degrees. See the remarks below.

phi (*phi*) [real number].

Orientation of the source element, in degrees. See the remarks below.

psi (*psi*) [real number].

Orientation of the source element, in degrees. See the remarks below.

Source (*source*) [name of an object].

Reference to an object of class *Source* specifying the source used as the array element. The source is inserted together with its own coordinate system such that this coincides with the element position and orientation. Note that the syntax allows for recursive definitions, i.e. a sub-array can easily be defined.

Beam Id (*beam_id*) [character string], default: .

This struct member is only used if the above source refers to another array for which the attribute *element_composite* has been set to 'element'. In this case, the string must contain the beam identifier for that beam which is used here as array element. In all other cases, the *beam_id* must be an empty string ('').

Element Composite (*element_composite*) [item from a list of character strings], default: **element**.

Determines if the array field shall be calculated on an element-by-element basis or as a composite array.

element

The array radiation is calculated element by element without using the excitation coefficients. The computed field will contain a number of beams equal to the number of elements in the array.

composite

The array radiation is computed as a composite field. Each beam is multiplied by the excitation coefficient of the element and added up to a total field.

Exc (exc) [sequence of structs].

Excitation coefficients of the array elements. This attribute is only required if *element_composite* is set to composite. Each struct in the sequence defines an excitation coefficient for one of the array elements. This sequence must not contain excitation coefficients for elements, which are not present in the *elements* attribute. If, on the other hand, an excitation coefficient is not specified for some of the elements, the excitation coefficients of these elements are set to zero (the elements are switched off).

Id (id) [character string].

Element identifier which associates the specified excitation to a particular array element.

Amplitude in dB (db) [real number], default: **0.**

Amplitude of the excitation, in dB.

Phase in degrees (deg) [real number], default: **0.**

Phase of the excitation, in degrees.

Factor (factor) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (db) [real number], default: **0.**

Amplitude of the factor, in dB.

Phase in degrees (deg) [real number], default: **0.**

Phase of the factor, in degrees.

Ray Output (ray_output) [item from a list of character strings], default: **none.**

This attribute defines how the array field shall be calculated if the field is used as input to a GTD or PTD analysis. The attribute is also actual for a subsequent PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case, *ray_output* has the same meaning as stated below for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that the field cannot be used as input to a GTD analysis. In a subsequent PTD analysis the direction of propagation is assumed to be the direction of Poynting's vector at the field point.

all

The field is computed as a ray field with one ray from each array element (including sub-array elements). A subsequent GTD or PTD analysis is then carried out for each ray separately. This is the most accurate method for GTD/PTD analysis, but it may be time consuming.

spherical

The field from the elements will be treated as a single ray which emanates from the origin of the array coordinate system (specified by *coor_sys*). This procedure is only accurate if the scatterer in the following GTD/PTD analysis is in the far field from the array or the array elements are phased such that they radiate a field at the scatterer with a spherical phase front centred at the origin.

plane

The field from the elements will be treated as a single ray which is parallel to the *z*-axis of the array coordinate system (specified by *coor_sys*). This procedure is only accurate if the elements radiate a field in the direction of the *z*-axis of the array coordinate system and if this field has a nearly plane phase front at the scatterer to be analysed by GTD/PTD.

Remarks

The array elements are positioned in a plane, regular grid that may be either hexagonal or rectangular. The origin and the rotation of the array grid are given by the attribute *grid_position* of which the elements *grid_cenx*, *grid_ceny* and *grid_rot* are shown in the figure below as the vector $\bar{r}_{cen} = (grid_cenx, grid_ceny)$ to a reference node in the grid and the angle $\theta_{grid} = grid_rot$ for the rotation of the grid around this node. The nodes of the grid are defined as an integer combination of the two base vectors \bar{M}_x and \bar{M}_y . Array elements may be specified at any node.

The length of the first base vector, \bar{M}_x , is *grid_spac* (also an element of attribute *grid_position*) and the length of the second base vector, \bar{M}_y , is *grid_spac* multiplied by *spac_ratio* (an element of attribute *grid_topology*). The angle between the two base vectors is 60° or 90° for the hexagonal or the rectangular grid, respectively, according to the value of *grid_type*.

Each element has a position (x, y, z) and an orientation (θ, ϕ, ψ) given by the members *mx*, *my* and *theta*, *phi*, *psi*, respectively, of the attribute *elements*.

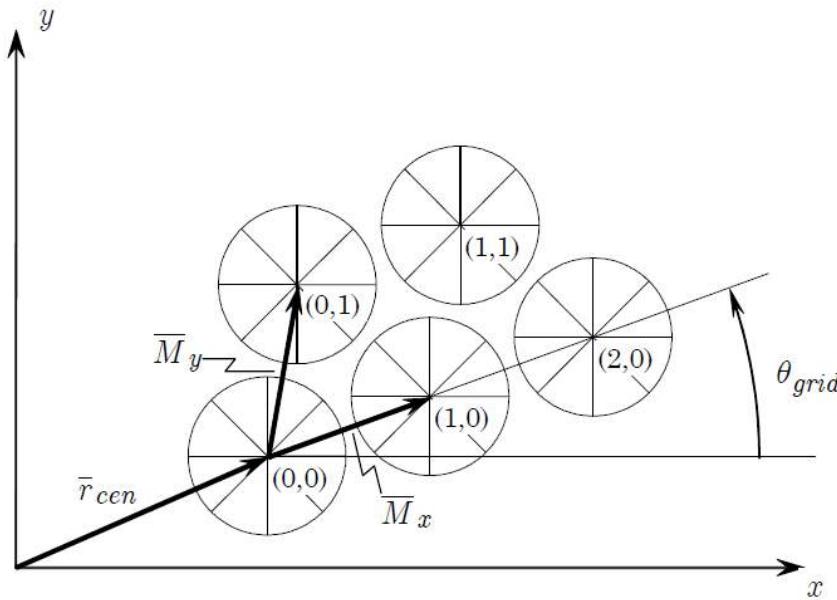


Figure 1

Illustration of five feed elements in a hexagonal grid. The element positions are given by the indices $(mx, my) = (0,0), (1,0), (2,0), (0,1)$ and $(1,1)$.

The position (x, y, z) is given in the array coordinate system by

$$(x, y, z) = \vec{r}_{cen} + mx\bar{M}_x + my\bar{M}_y$$

z will always have the value zero.

The angles (θ, ϕ, ψ) define the orientation of the coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ of each element through

$$\begin{aligned}\hat{x}_\ell &= \hat{\theta} \cos(\phi - \psi) - \hat{\phi} \sin(\phi - \psi) \\ \hat{y}_\ell &= \hat{\theta} \sin(\phi - \psi) + \hat{\phi} \cos(\phi - \psi) \\ \hat{z}_\ell &= \hat{r}\end{aligned}$$

where $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ are the base vectors of the element coordinate system and $\hat{r}, \hat{\theta}, \hat{\phi}$ are the standard spherical vectors in the array coordinate system,

$$\begin{aligned}\hat{r} &= (\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta) \\ \hat{\theta} &= (\cos \theta \cos \phi, \cos \theta \sin \phi, -\sin \theta) \\ \hat{\phi} &= (-\sin \phi, \cos \phi, 0)\end{aligned}$$

The coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ may be obtained by tilting the z -axis (and the full array coordinate system) the angle θ towards ϕ , and then rotate the system the angle ψ around the tilted z -axis. More details are given in the GRASP Technical Description in the section on Coordinate Systems.

Each source (element) in the array is defined in its own coordinate system such that this coordinate system coincides with the element coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ defined here.

The field of the array is determined element by element as near field or far field as described under the respective **Source** elements.

TABULATED PLANAR GRID ARRAY (tabulated_planar_grid_array)

Purpose

The class *Tabulated Planar Grid Array* defines an array of source elements where the elements can be arranged in planar hexagonal or rectangular grids. The element data are read from a data file.

If it is preferred to specify the data directly through the attributes, the class *Planar Grid Array* may be used instead.

When the elements are not arranged in a regular grid the class *General Array* or the class *Tabulated General Array* may be applied.

Links

Classes→*Electrical Objects*→*Other Sources*→*Array*→*Tabulated Planar Grid Array*

Remarks

Syntax

```
<object name> tabulated_planar_grid_array
(
    frequency                      : ref(<n>),
    coor_sys                        : ref(<n>),
    grid_topology                   : struct(grid_type:<si>, spac_ratio:<r>),
    grid_position                   : struct(grid_cenx:<rl>, grid_ceny:<rl>,
                                                grid_rot:<r>, grid_spac:<rl>),
    elements_file                   : <f>,
    element_composite              : <si>,
    exc_file                        : <f>,
    factor                          : struct(db:<r>, deg:<r>),
    ray_output                      : <si>
)
where
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the array should radiate.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate Systems* defining the coordinate system (the array coordinate system) in which the array and its elements are defined.

Grid Topology (*grid_topology*) [struct].

Defines the topology of the array grid.

Grid Type (*grid_type*) [item from a list of character strings], default: **hexagonal**.

The array elements may be organised in a regular hexagonal or rectangular grid and *grid_type* may have one of the two values. hexagonal

When the grid is hexagonal, the angle between the first and second grid direction is 60° .

rectangular

In a rectangular grid the angle between the first and second grid direction is 90° .

Spacing Ratio (*spac_ratio*) [real number], default: **1**.

The element spacing in the second grid direction relative to the element spacing in the first grid direction (*grid_spac*, see the attribute Grid Position).

Grid Position (*grid_position*) [struct].

Determines the position and orientation of the grid in which the array element positions are defined.

Grid Centre in X (*grid_cenx*) [real number with unit of length].

The position of the reference point in the grid is at $(x,y) = (\text{grid_cenx}, \text{grid_ceny})$ in the *xy*-plane of the array coordinate system. The reference point must be a node in the grid (see also the figure in the remarks below).

Grid Centre in Y (*grid_ceny*) [real number with unit of length].

The position of the reference point in the grid is at $(x,y) = (\text{grid_cenx}, \text{grid_ceny})$ in the *xy*-plane of the array coordinate system. The reference point must be a node in the grid (see also the figure in the remarks below).

Grid Rotation (*grid_rot*) [real number].

Rotation angle of the grid in the *xy*-plane of the array coordinate system around the grid point, measured positive in degrees from the *x*-axis towards the *y*-axis. See also the figure in the remarks below.

Grid Spacing (*grid_spac*) [real number with unit of length].

Spacing between elements along first direction in the grid.

Elements File (*elements_file*) [file name].

Name of file in which the element position, orientation and definition are specified (see the remarks for positioning of the elements). See Section *Feed Data in Regular Grid* for a description of the format of this file.

Element or Composite (*element_composite*) [item from a list of character strings], default: **element**.

Determines if the array field shall be calculated on an element-by-element basis or as a composite array.

element

The array radiation is calculated element by element without using the excitation coefficients. The computed field will contain a number of beams equal to the number of elements in the array.

composite

The array radiation is computed as a composite field. Each beam is multiplied by the excitation coefficient of the element and added up to a total field.

Excitation Coef. File (*exc_file*) [file name].

Name of the file from which the excitation coefficients of the elements are read. This file is only required if Element or Composite is set to composite. The format of the file is described in Section *Array Element Excitation Coefficients*.

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Ray Output (*ray_output*) [item from a list of character strings], default: **none**.

This attribute defines how the array field shall be calculated if the field is used as input to a GTD or PTD analysis. The attribute is also actual for a subsequent PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case, *ray_output* has the same meaning as stated below for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that the field cannot be used as input to a GTD analysis. In a subsequent PTD analysis the direction of propagation is assumed to be the direction of Poynting's vector at the field point.

all

The field is computed as a ray field with one ray from each array element (including sub-array elements). A subsequent GTD or PTD analysis is then carried out for each ray separately. This is the most accurate method for GTD/PTD analysis, but it may be time consuming.

spherical

The field from the elements will be treated as a single ray which emanates from the origin of the array coordinate system (specified by *coor_sys*). This procedure is only accurate if the scatterer in the following GTD/PTD analysis is in the far field from the array or the array elements are phased such that they radiate a field at the scatterer with a spherical phase front centred at the origin.

plane

The field from the elements will be treated as a single ray which is parallel to the *z*-axis of the array coordinate system (specified by *coor_sys*). This procedure is only accurate if the elements radiate a field in the direction of the *z*-axis of the array coordinate system and if this field has a nearly plane phase front at the scatterer to be analysed by GTD/PTD.

Remarks

The array elements are positioned in a plane, regular grid that may be either hexagonal or rectangular. The origin and the rotation of the array grid are given by the attribute *grid_position* of which the elements *grid_cenx*, *grid_ceny* and *grid_rot* are shown in Figure 1 below as the vector $\vec{r}_{cen} = (\text{grid_cenx}, \text{grid_ceny})$ to a reference node in the grid and the angle $\theta_{grid} = \text{grid_rot}$ for the rotation of the grid around this node. The nodes of the grid are defined as an integer combination of the two base vectors \bar{M}_x and \bar{M}_y . Array elements may be specified at any node.

The length of the first base vector, \bar{M}_x , is *grid_spac* (also an element of attribute Grid Position) and the length of the second base vector, \bar{M}_y , is *grid_spac* multiplied by *spac_ratio* (an element of attribute Grid Topology). The angle between the two base vectors is 60° or 90° for the hexagonal or the rectangular grid, respectively, according to the value of *grid_type*.

Each element has a position (x, y, z) and an orientation (θ, ϕ, ψ) given by the data *mx*, *my* and *theta*, *phi*, *psi*, respectively, in Elements File.

The position (x, y, z) is given in the in the array coordinate system by

$$(x, y, z) = \vec{r}_{cen} + mx\bar{M}_x + my\bar{M}_y, \quad (1)$$

z will always have the value zero.

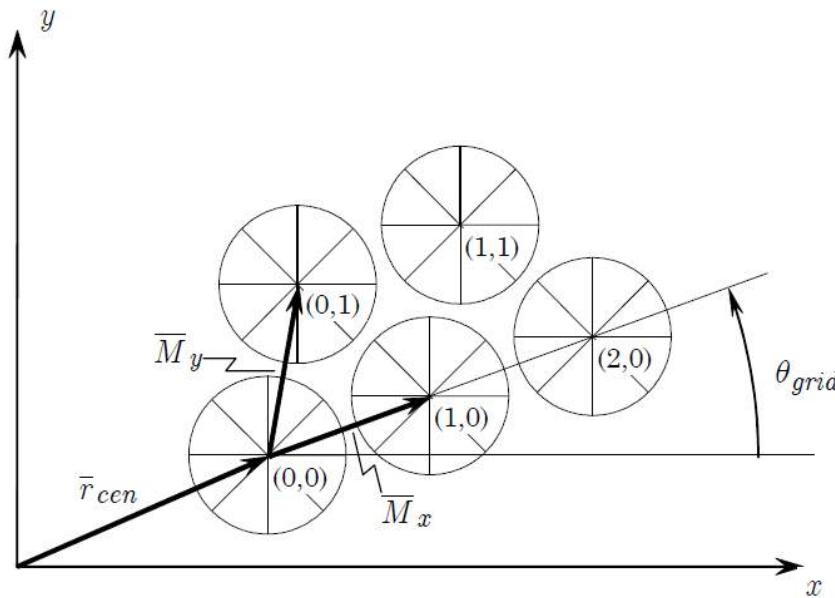


Figure 1

Illustration of five feed elements in a hexagonal grid. The element positions are given by the indices $(mx, my) = (0,0), (1,0), (2,0), (0,1)$ and $(1,1)$.

The angles (θ, ϕ, ψ) define the orientation of the coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ of each element through

$$\begin{aligned}\hat{x}_\ell &= \hat{\theta} \cos(\phi - \psi) - \hat{\phi} \sin(\phi - \psi) \\ \hat{y}_\ell &= \hat{\theta} \sin(\phi - \psi) + \hat{\phi} \cos(\phi - \psi) \\ \hat{z}_\ell &= \hat{r}\end{aligned}$$

where $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ are the base vectors of the element coordinate system and $\hat{r}, \hat{\theta}, \hat{\phi}$ are the standard spherical vectors in the array coordinate system,

$$\begin{aligned}\hat{r} &= (\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta) \\ \hat{\theta} &= (\cos \theta \cos \phi, \cos \theta \sin \phi, -\sin \theta) \\ \hat{\phi} &= (-\sin \phi, \cos \phi, 0)\end{aligned}$$

The coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ may be obtained by tilting the z -axis (and the full array coordinate system) the angle θ towards ϕ , and then rotate the system the angle ψ around the z -axis. More details are given in the Technical Description in the section on Coordinate Systems.

Each source (element) in the array is defined in its own coordinate system such that this coordinate system coincides with the element coordinate system $\hat{x}_\ell, \hat{y}_\ell, \hat{z}_\ell$ defined here.

The field of the array is determined element by element as near field or far field as described under the respective *Source* elements.

TABULATED PLANAR SOURCE (tabulated_planar_source)

Purpose

This class is used to transform a field specified in a plane to equivalent currents in the same plane. These currents are then integrated when the radiated field shall be determined.

A typical application is the radiation from a plane aperture. The field in the aperture can be determined by objects of the *Field Storage* class and stored on a file.

Links

[Classes](#)→[Electrical Objects](#)→[Other Sources](#)→[Tabulated Planar Source](#)

Remarks

Syntax

```
<object name> tabulated_planar_source
(
    frequency           : ref(<n>),
    coor_sys            : ref(<n>),
    e_file               : struct(status:<si>, file_name:<f>),
    h_file               : struct(status:<si>, file_name:<f>),
    file_attributes     : struct(file_type:<si>, unit:<si>,
                                  obsolete_file_form:<si>),
    max_m_mode_index    : <i>,
    power_norm           : struct(status:<si>, method:<si>),
    po_points            : struct(po1:<i>, po2:<i>),
    factor               : struct(db:<r>, deg:<r>),
    ray_output            : <si>,
    far_field_convergence : struct(status:<si>, cone_angle:<r>),
    near_field_convergence : struct(status:<si>, near_dist:<rl>,
                                      rho:<rl>),
    scatterer_convergence : struct(status:<si>, scatterer:ref(<n>))
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object. See the remarks below.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The *Tabulated Planar Source* is located in the *xy*-plane and with phase centre at the origin of this coordinate system. A standard polar $\rho \phi$ coordinate system is defined in the *xy*-plane.

E File (*e_file*) [struct].

Specifies input in the form of E-fields.

Status (*status*) [item from a list of character strings], default: **off**.

Determines if the E-fields are used as input:

off

No E-fields will be read.

on

A file with E-fields will be read (see the remarks below).

File Name (*file_name*) [file name].

For '*status*: on' the name of the file containing the tabulated E-field shall be specified. The file format shall be in accordance with the *file_type* (see file attributes).

H File (*h_file*) [struct].

Specifies input in the form of H-fields.

Status (*status*) [item from a list of character strings], default: **off**.

Determines if the H-fields are used as input:

off

No H-fields will be read.

on

A file with H-fields will be read (see the remarks below).

File Name (*file_name*) [file name].

For '*status*: on' the name of the file containing the tabulated H-field shall be specified. The file format shall be in accordance with the *file_type* (see file attributes).

File Attributes (*file_attributes*) [struct].

When both an E-field and an H-field are read, the file attributes are the same for the two files.

File Type (*file_type*) [item from a list of character strings], default: **grid**.

The field data may be organised.

grid

The file format of a field grid is described in Section *Field Data in Rectangular Grid*.

cut

The file format of cuts is described in Section *Field Data in Cuts*.

Unit (*unit*) [item from a list of character strings], default: **m**.

Defines the unit of length by which the field positions are given of the file(s). The unit must be a valid unit of length (mm, cm, m, km, in, ft).

File Form (Obsolete) (*obsolete_file_form*) [item from a list of character strings], default: **formatted**.

Determines the file format (obsolete):

formatted

The files are written as an formatted file.

unformatted_single

The files are written as an unformatted file.

Max M Mode Index (*max_m_mode_index*) [integer], default: **-1**.

Only applicable for cut-file(s), *file_type*:cut. Maximum number of azimuthal *m*-modes to be used in the azimuthal ϕ -interpolation of tabulated cut data. If -1, all azimuthal modes are retained (default).(This interpolation in ϕ is a Fourier interpolation while the interpolation in ρ applies cubic interpolation).

Power Norm (*power_norm*) [struct].

The total radiated power may be normalised to 4π watts whereby the pattern is normalised to dBi.

Status (*status*) [item from a list of character strings], default: **off**.

Determines if the pattern is normalised:

off

No normalisation is carried out.

on

The field is power normalised to 4π .

Method (*method*) [item from a list of character strings], default: **approximate**.

The power normalisation may be carried out by one of the two methods.

approximate

The field is normalised by integration of Poynting's vector over the plane. This method is adequate when the *Tabulated Planar Source* is large in wavelengths and the tabulated fields are in phase. If only the E-field or the H-field is input, Poynting's vector is determined assuming locally a plane wave propagating orthogonally to the input plane. The approximate method becomes more correct if both the E-field and H-field are input. The approximate method is faster than the exact method.

exact

The field is normalised by integration over the far field sphere (half sphere if only an E-field or an H-field is read and full sphere if both fields are read). The exact method is preferable if the *Tabulated Planar Source* is small in wavelengths. The exact method is time consuming if the source is large (e.g. $>20\lambda$).

Po Points (*po_points*) [struct].

The integration grid is specified by *po1* and *po2*. The current elements in this grid are obtained by interpolation in the input field data. A description of the necessary number of points can be found in the remarks to the class *PO, Single-Face Scatterer*.

po1 (*po1*) [integer], default: **0**.

Number of current samples in first grid coordinate (i.e. ρ for cuts and x for grids).

po2 (*po2*) [integer], default: **0**.

Number of current samples in second grid coordinate (i.e. ϕ for cuts and y for grids).

Factor (*factor*) [struct].

The radiated field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Ray Output (*ray_output*) [item from a list of character strings], default: **none**.

This attribute defines how the field of the *Tabulated Planar Source* shall be calculated if it is used as source in a GTD or PTD analysis. The attribute is also actual for a subsequent PO-analysis of a scatterer with electrical properties specified (the direction of propagation is not used in a PO analysis of a scatterer which is perfectly conducting, i.e. without electrical properties specified). For this particular PO case, Ray Output has the same meaning as stated below for PTD.

none

The direction of propagation of the radiated field will not be computed. This means that the field cannot be used as input to a GTD analysis. In a subsequent PTD analysis, the direction of propagation is assumed to be the direction of Poynting's vector at the field point.

all

The field is computed as a ray field with one ray from each current element. A subsequent GTD or PTD analysis is then carried out for each ray separately. This is the most accurate method for GTD/PTD analysis, but it may be time consuming.

spherical

The field from the *Tabulated Planar Source* will be treated as a single ray which emanates from the origin of the coordinate system specified by Coordinate System. This procedure is only accurate if the scatterer in the following GTD/PTD analysis is in the far field from the source or the array elements are phased such that they radiate a field at the scatterer with a spherical phase front centred at the origin.

plane

The field from the elements will be treated as a single ray which is parallel to the z -axis of the coordinate system specified by *coor_sys*. This procedure is only accurate if the elements radiate a field in the direction of the z -axis of the coordinate system and if this field has a nearly plane phase front at the scatterer to be analysed by GTD/PTD.

Far Field Convergence (*far_field_convergence*) [struct].

When the far field of the source shall be determined, *po1* and *po2* of the attribute *po_points* may be determined automatically, see also the remarks below.

Status (*status*) [item from a list of character strings], default: **off**.

Determines if the parameters are found automatically:

off

po1 and *po2* are not determined.

on

po1 and *po2* are determined automatically.

Cone Angle (*cone_angle*) [real number], default: **90**.

The convergence of the PO integral is secured for a far field within a cone where the half cone angle is *cone_angle* (in degrees) and the z -axis of *coor_sys* is the cone axis.

Near Field Convergence (*near_field_convergence*) [struct].

When the near field of the source shall be determined on a plane, *po1* and *po2* of the attribute *po_points* may be determined automatically, see also the remarks below.

Status (*status*) [item from a list of character strings], default: **off**.

Determines if the parameters are found automatically:

off

po1 and *po2* are not determined.

on

po1 and *po2* are determined automatically.

Near Dist (*near_dist*) [real number with unit of length], default: **0**.

The near field shall be determined on a plane parallel to the *xy*-plane of *coor_sys* and *z*-value equal to *near_dist*.

rho (*rho*) [real number with unit of length], default: **0**.

The convergence of the PO integral is secured for the near field within a circle of radius *rho* on the above plane. The centre of the circle is on the *z*-axis of *coor_sys*

Scatterer Convergence (*scatterer_convergence*) [struct].

When the near field of the source shall be determined on a scatterer, *po1* and *po2* of the attribute *po_points* may be determined automatically, see also the remarks below.

Status (*status*) [item from a list of character strings], default: **off**.

Determines if the parameters are found automatically:

off

po1 and *po2* are not determined.

on

po1 and *po2* are determined automatically.

Scatterer (*scatterer*) [name of an object], default: **blank**.

Reference to an object of the class *Scatterer* on which the field shall be determined.

Remarks

If the cut or grid file(s) contain more than one frequency, the number of frequencies shall agree with the number of frequencies in the frequency object.

When the *file_type* is specified to 'cut', the field in the file(s) may be given as asymmetrical cuts (the field is given in the interval $0 \leq \rho \leq \rho_{\max}$ in cuts for $0 \leq \phi < 360^\circ$) or symmetrical cuts (the field is given in the interval $-\rho_{\max} \leq \rho \leq \rho_{\max}$ in cuts for $0 \leq \phi < 180^\circ$). In both cases, the field at $\rho = 0$ shall be included which means that for a symmetrical cut an uneven number of points must be read. Furthermore, the file must begin with the cut along $\phi = 0$.

The number of asymmetrical cuts N_a must be even and the maximum azimuthal mode index M becomes $M = N_a/2 - 1$. For N_s symmetrical cuts M becomes $M = N_s - 1$.

The field is interpolated by a Fourier method along ϕ and by a third order interpolation along ρ .

When the *file_type* is specified to 'grid', the interpolation is a third order interpolation along the two grid coordinates.

If any of the three convergence-attributes are set, the number of PO points specified by the user (*po1* and *po2*) will be used in the PO integral only if

they turn out to be larger than the numbers obtained from the convergence tests. If more than one convergence attribute is set the one that gives the largest values of *po1* and *po2* will be used. The convergence tests are performed by evaluating the field values at a number of test points on the desired surface (far field, near-field plane or scatterer) with an accuracy of approximately 80 dB below peak.

FIELD STORAGE

Purpose

The classes of the menu *Field Storage* define the output points at which the field is calculated.

The output points may be grouped in a one-dimensional

Cut

or they may be grouped in a two-dimensional

Grid

Finally, field grids, irregular in two dimensions, may be defined in the class

Tabulated uv-Points

Command Types

The calculation of the specified field is activated by one of the commands:

Get Field

Add Field

Subtract Field

Further, the following commands may be applied to interpolate, add and subtract already calculated fields.

Get Pattern

Add Pattern

Subtract Pattern

Replace Pattern

Links

Classes→*Electrical Objects*→*Field Storage*

CUT

Purpose

Cut is a menu which contains classes that define regular field cuts or pattern cuts.

Field upon a surface in space:

Spherical Cut (includes far-field cuts),

Planar Cut

Cylindrical Cut

Field at the surface of a scatterer:

Surface Cut

Links

Classes→*Electrical Objects*→*Field Storage*→*Cut*

SPHERICAL CUT (spherical_cut)

Purpose

The class *Spherical Cut* defines points in cuts on a sphere at which the field is to be calculated. Polar as well as conical cuts can be specified, both in the near-field and in the far-field region.

Links

Classes→*Electrical Objects*→*Field Storage*→*Cut*→*Spherical Cut*

Remarks

Syntax

```
<object name> spherical_cut
(
    coor_sys : ref(<n>),
    cut_type : <si>,
    theta_range : struct(start:<r>, end:<r>, np:<i>),
    phi_range : struct(start:<r>, end:<r>, np:<i>),
    e_h : <si>,
    polarisation : <si>,
    polarisation_modification : struct(status:<si>, coor_sys:ref(<n>)),
    near_far : <si>,
    near_dist : <rl>,
    file_name : <f>,
    file_format : <si>,
    comment : <s>,
    frequency : ref(<n>)
)
```

where

<i> = integer

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

<s> = character string

<f> = file name

<si> = item from a list of character strings

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes in *Coordinate System* defining the coordinate system (the output coordinate system) in which the field points will be calculated. Further, the origin serves as the phase reference for far-fields. The field polarisation components will also be expressed in this coordinate system unless otherwise specified in the attribute *polarisation_modification*.

Cut Type (*cut_type*) [item from a list of character strings], default: **polar**.

Defines the direction of the cuts over the sphere. See also the remarks below.

polar

Polar cuts are cuts for which ϕ is constant and θ is varying.
Polar cuts will pass through the pole of the sphere when $\theta = 0^\circ$ is within the θ -range.

conical

Conical cuts are cuts for which θ is constant and ϕ is varying.

theta-Range (*theta_range*) [struct].

Defines the range of the polar angle θ (see the remarks below).

Start (*start*) [real number].

Start value of the polar coordinate θ , in degrees.

End (*end*) [real number].

End value of the polar coordinate θ , in degrees.

Np (*np*) [integer].

Number of θ -values. When the *cut_type* is specified to 'polar' *np* is the number of θ -values in each polar cut. When the *cut_type* is specified to 'conical' *np* is the number of conical cuts.

phi-Range (*phi_range*) [struct].

Defines the range of the azimuthal angle ϕ (see the remarks below).

Start (*start*) [real number].

Start value of the azimuthal coordinate ϕ , in degrees.

End (*end*) [real number].

End value of the azimuthal coordinate ϕ , in degrees.

Np (*np*) [integer].

Number of ϕ -values. When the *cut_type* is specified to 'polar' *np* is the number of polar cuts. When the *cut_type* is specified to 'conical' *np* is the number of ϕ -values in each conical cut.

E/H-Field (*e_h*) [item from a list of character strings], default: **e_field**.

Specifies whether the complex *E*-field or *H*-field shall be calculated.

e_field

The complex *E*-field is calculated.

h_field

The complex *H*-field is calculated.

Polarisation (*polarisation*) [item from a list of character strings], default: **linear**.

Defines how the calculated field shall be decomposed. All components refer to the polarisation coordinate system as defined under the attribute Polarisation Modification. In the near field the r -component of the field is calculated as a third component.

linear

Linear components are calculated according to Ludwig's 3rd definition, with the first component (E_{co}) along x and the second component (E_{cx}) along y (at $\theta = 0^\circ$). The notation implies that for a field, which is mainly y -polarised then the second component (E_{cx}) represents the co-polar field component.

circular

Circular components are calculated based on the linear components defined above. The first component is the right hand circular (E_{rhc}) and the second is left hand circular component (E_{lhc}).

theta_phi

The field is decomposed along the θ - and ϕ -unit vectors with the θ -component (E_θ) being the first component and the ϕ -component being the second (E_ϕ).

major_minor

The field is de-composed along the major and minor axes of the polarisation ellipse. The first field component is parallel to the major axis (E_{maj}) and the second to the minor axis (E_{min}).

linear_xpd

The ratios E_{co}/E_{cx} and E_{cx}/E_{co} , where E_{co} and E_{cx} are the first and second components as defined for the '*polarisation: linear*' above. This is the linear cross-polar discrimination ratio.

circular_xpd

The ratios E_{rhc}/E_{lhc} and E_{lhc}/E_{rhc} , where E_{rhc} and E_{lhc} are the first and second components as defined for the '*polarisation: circular*' above. This is the circular cross-polar discrimination ratio.

theta_phi_xpd

The ratios E_θ/E_ϕ and E_ϕ/E_θ , where E_θ and E_ϕ are the first and second components as defined for the '*polarisation: theta_phi*' above.

major_minor_xpd

The ratios E_{maj}/E_{min} and E_{min}/E_{maj} , where E_{maj} and E_{min} are the first and second components as defined for the '*polarisation: major_minor*' above.

power

The first component is the amplitude of the field, $|\vec{E}|$, (i.e. the square root of the power) and the second component is the complex square root $\sqrt{E_{rhc}/E_{lhc}}$. The phase of the latter component is the rotation angle of the polarisation ellipse.

In the far field the square root of the power is determined from

$$|\vec{E}| = \sqrt{|E_{co}|^2 + |E_{cx}|^2}$$

and in the near field it is determined from all three field components: $|\vec{E}| = \sqrt{|E_{co}|^2 + |E_{cx}|^2 + |E_r|^2}$, E_r being the r -component of the field.

Polarisation Modification (*polarisation_modification*) [struct].

Defines a coordinate system in which the field polarisation components are determined (if different from the output coordinate system defined).

Status (*status*) [item from a list of character strings], default: **off**.

Determines if the polarisation modification shall be performed:

off

No polarisation modification, the polarisation is defined in the above defined output coordinate system and the polarisation coordinate system is identical to the output coordinate system.

on

The polarisation is defined in the coordinate system defined next.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes in *Coordinate System* defining the coordinate system (the polarisation coordinate system) in which the polarisation components of the calculated field vectors will be expressed. Shall only be specified for *status*: on.

Near Far (*near_far*) [item from a list of character strings], default: **far**.

The value specifies if a near or a far field is to be calculated.

far

A far field is calculated. In this case only two field components are calculated according to the specified *polarisation* as described in the remarks below.

near

The three field components of a near field is calculated. These are the two components defined under the attribute *polarisation* below and the third component is the radial component.

Near Dist (*near_dist*) [real number with unit of length], default: **0**.

Defines the radius of a near-field sphere. For a far field this attribute has no effect and needs not to be specified.

File Name (*file_name*) [file name].

Name of a file, to which the calculated field values shall be written.

File Format (*file_format*) [item from a list of character strings], default: **TICRA**.

The file format used to store the field data:

TICRA

The field is written in GRASP units and stored as an ASCII-file according to the TICRA-format described in *Field Data in Cuts*. The recommended file extension is *.cut*.

EDX

The field is written in SI units in a format according to the Electromagnetic Data Exchange (EDX) standard. The recommended file extension is *.cut*.

Files in this format cannot be read by the PostProcessor.

EDI

Obsolete file format name. The same as EDX.

Comment (*comment*) [character string], default: **Field data in cuts**.

A line of text which will be written as a header in the file specified by *file_name* above.

Frequency (*frequency*) [name of an object], default: **blank**.

Reference to a *Frequency* object, defining the frequencies for which the field is calculated. The reference must be to the same object as specified in the *frequency* attribute of the source generating the field. A *frequency* needs not to be specified. In that case, the frequencies in the *frequency* object of the source generating the field will be applied.

Command Types

The *Spherical Cut* class is derived from the class *Field Storage*. See this class for available commands.

Remarks

The section introduces the position of the field points and the definition of a polarisation coordinate system.

Field Points

Field points in a *Spherical Cut* are defined in usual spherical (θ, ϕ) -coordinates given in the output coordinate system. For far fields, (θ, ϕ) defines a direction

$$\hat{r} = \hat{x} \sin \theta \cos \phi + \hat{y} \sin \theta \sin \phi + \hat{z} \cos \theta$$

and for near fields, (θ, ϕ) defines a point

$$\bar{R} = R(\hat{x} \sin \theta \cos \phi + \hat{y} \sin \theta \sin \phi + \hat{z} \cos \theta)$$

where $R = |\bar{R}|$ is the radius (given by Near Dist) of the near-field sphere.

In a polar cut, ϕ is fixed and θ takes on the values

$$\theta_i = \theta_{start} + \Delta\theta \cdot (i - 1), \quad i = 1, 2, \dots, n_\theta \quad (1)$$

with

$$\Delta\theta = (\theta_{end} - \theta_{start}) / (n_\theta - 1)$$

where θ_{start} and θ_{end} are the members *start* and *end*, respectively, of the attribute *theta_range*, and n_θ is the member *np* of the same attribute.

The ϕ -angle is increased equidistantly from one cut to the next by the values

$$\phi_j = \phi_{start} + \Delta\phi \cdot (j - 1), \quad j = 1, 2, \dots, n_\phi \quad (2)$$

with

$$\Delta\phi = (\phi_{end} - \phi_{start}) / (n_\phi - 1)$$

where ϕ_{start} and ϕ_{end} are the members *start* and *end*, respectively, of the attribute *phi_range*, and n_ϕ is the member *np* of the same attribute.

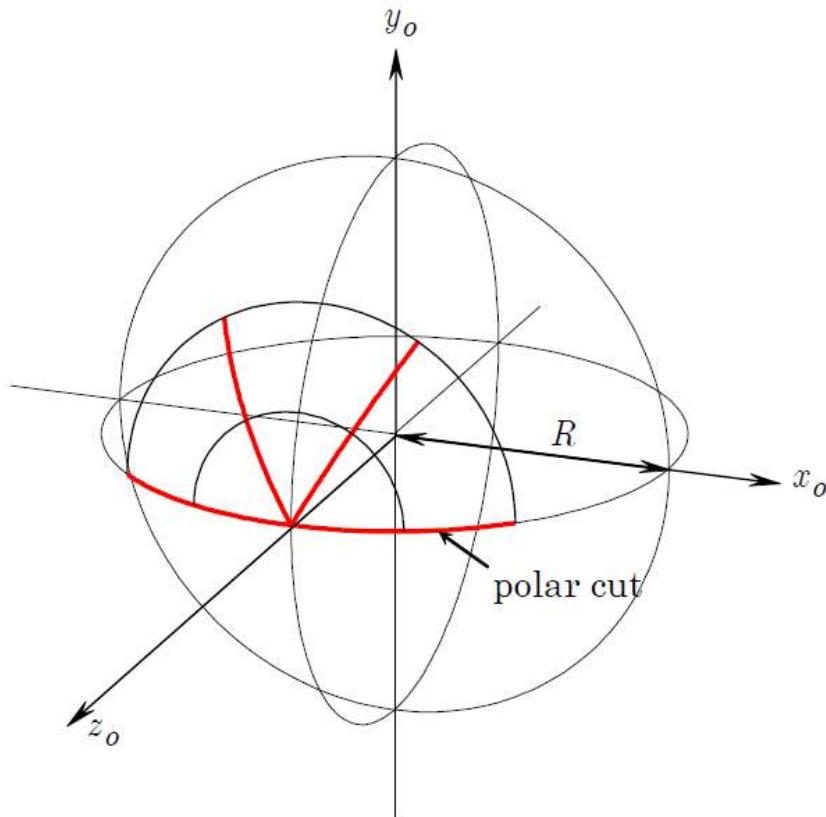


Figure 1 4 polar cuts for $0^\circ \leq \theta \leq 45^\circ$, and $\phi = 0^\circ, 60^\circ, 120^\circ$ and 180° .

In a conical cut θ is fixed and ϕ runs through the values given by Eq. (2). From one conical cut to the next, the θ -angle is increased according to Eq. (1).

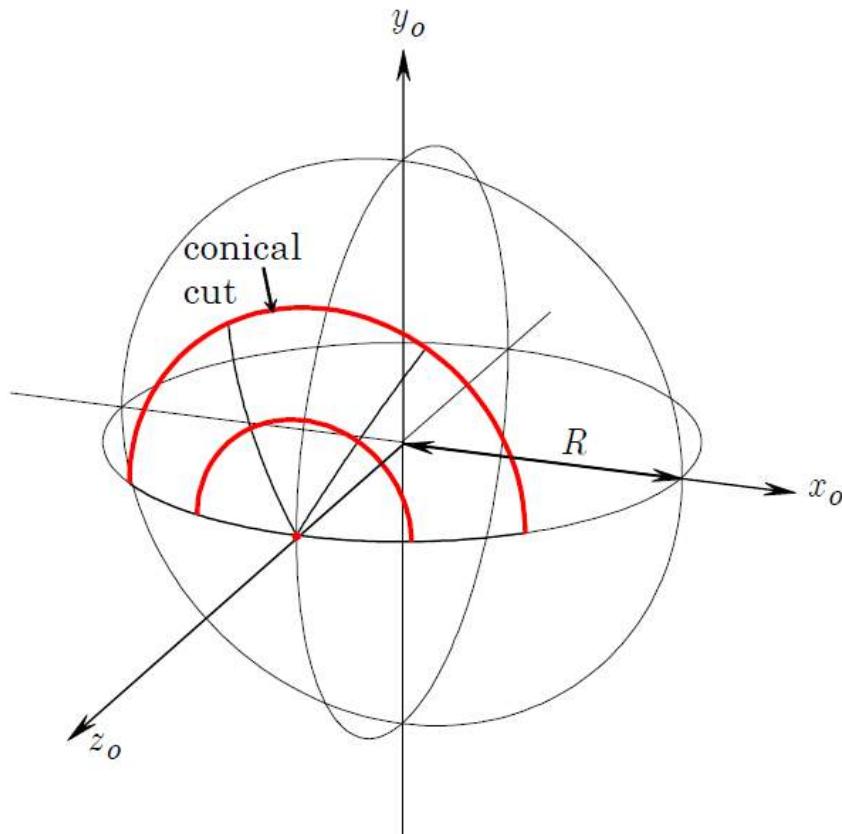


Figure 2 3 conical cuts for $\theta = 0^\circ, 22.5^\circ$ and 45° , and $0^\circ \leq \phi \leq 180^\circ$.

The Polarisation Coordinate System

When the field polarisation is requested in another coordinate system than the output coordinate system then the attribute *polarisation_modification* shall be applied with ‘status: on’ followed by a reference to the polarisation coordinate system.

The attribute *polarisation* may be specified as ‘linear’ (E_{co} - and E_{cx} -components), ‘circular’ (E_{rhc} - and E_{lhc} -components) or ‘theta_phi’ (E_θ - and E_ϕ -components). In the near field also an E_r -component will be present. In case of ‘theta_phi’ polarisation the electric field is given by

$$\bar{E} = E_r \hat{r} + E_\theta \hat{\theta} + E_\phi \hat{\phi}$$

where \hat{r} , $\hat{\theta}$ and $\hat{\phi}$ are the polarisation vectors in the polarisation coordinate system. The spherical cut in which the field is determined is always given in the output coordinate system.

An example is shown in Figure 3, the cut is the polar cut shown in red and the field shall be determined at one of the output points, P . The output coordinate system is denoted $x_0y_0z_0$ and the polarisation coordinate system is $x_py_pz_p$. In the figure the latter is a simple rotation of the former around the y -axis but any coordinate system may be chosen as polarisation coordinate system.

Internally in GRASP the field is calculated in Cartesian components in the output coordinate system

$$\bar{E} = E_{ox} \hat{x}_0 + E_{oy} \hat{y}_0 + E_{oz} \hat{z}_0$$

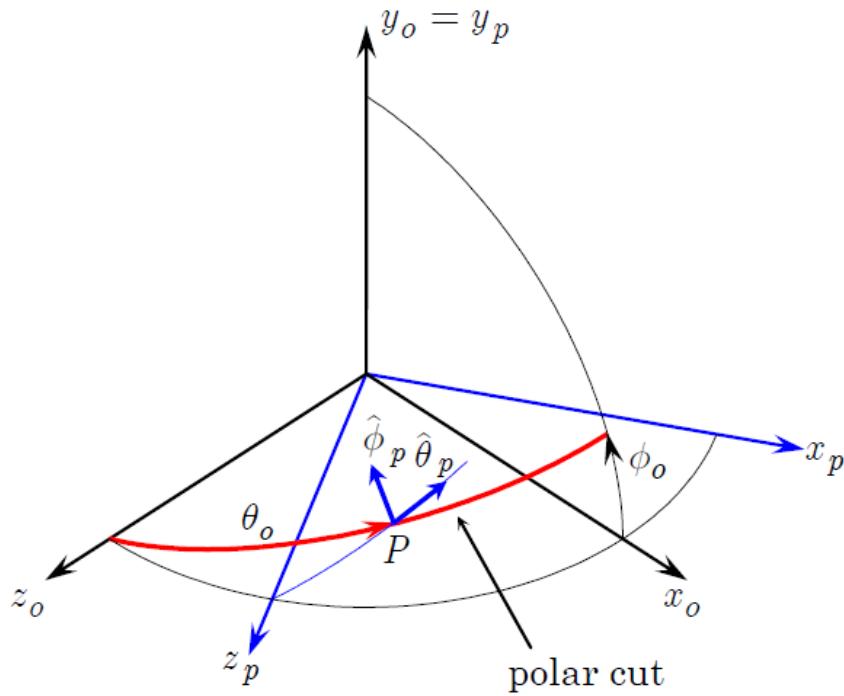


Figure 3

In red is shown a single polar cut in the output coordinate system given by $x_0y_0z_0$. A field point P at the direction (θ_0, ϕ_0) is illustrated. The polarisation is expressed in $\theta\phi$ -components (*polarisation*: theta_phi) along $\hat{\theta}_p$ and $\hat{\phi}_p$ in the polarisation coordinate system $x_py_pz_p$.

The field is then converted to the new components

$$\bar{E} = E_{px}\hat{x}_p + E_{py}\hat{y}_p + E_{pz}\hat{z}_p$$

before the Cartesian components are converted to (in this case) the polar components in the usual way.

The resulting θ_p -component (shown in blue) is pointing away from the z_p -axis in the same way as the standard θ_0 -component will point away from the z_0 -axis (along θ_0).

PLANAR CUT (planar_cut)

Purpose

The class *Planar Cut* defines field points in cuts on a plane, at a finite distance from the origin of the reference coordinate system, i.e. points that are generally in the near field of the source. Two types of cuts can be specified, radial and circular. In a radial cut, the points are located along "spokes" for constant values of the plane-polar coordinate ϕ . In a circular cut, the points are located on rings for constant values of the plane-polar coordinate ρ . Only near fields can be computed.

Links

Classes→*Electrical Objects*→*Field Storage*→*Cut*→*Planar Cut*

Remarks

Syntax

```
<object name> planar_cut
(
    coor_sys           : ref(<n>),
    near_dist          : <rl>,
    cut_type           : <si>,
    rho_range          : struct(start:<r>, end:<r>, np:<i>,
                                rho_unit:<si>),
    phi_range          : struct(start:<r>, end:<r>, np:<i>),
    e_h                : <si>,
    polarisation       : <si>,
    file_name          : <f>,
    file_format        : <si>,
    comment            : <s>,
    frequency          : ref(<n>)
)
```

where

<i> = integer

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

<s> = character string

<f> = file name

<si> = item from a list of character strings

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system (the output coordinate system) in which the field points and the calculated field vectors will be expressed.

Near Dist (*near_dist*) [real number with unit of length], default: **0**.

Defines the distance along the *z*-axis of the output coordinate system (defined by the attribute Coordinate System), from the origin to the plane in which the cuts are placed.

Cut Type (*cut_type*) [item from a list of character strings], default: **radial**.

Defines the direction of the cuts in the plane. See also the remarks below.

radial

Radial cuts are made along spokes for constant values of the azimuth angle ϕ , while the radius ρ is varying.

circular

Circular cuts are cuts where ρ is constant and ϕ is varying.

rho-Range (*rho_range*) [struct].

Defines the range of the radial coordinate ρ (see the remarks below).

Start (*start*) [real number].

Start value of the radial coordinate ρ .

End (*end*) [real number].

End value of the radial coordinate ρ .

Np (*np*) [integer].

Number of ρ -values. When Cut Type is specified to 'radial', np is the number of ρ -values in each radial cut. When Cut Type is specified to 'circular' np is the number of circular cuts.

rho-Unit (*rho_unit*) [item from a list of character strings], default: **m**.

Defines the unit of length in which the rho-Range is given (mm, cm, m, km, in, ft).

phi-Range (*phi_range*) [struct].

Defines the range of the azimuthal angle ϕ (see the remarks below).

Start (*start*) [real number].

Start value of the azimuthal coordinate ϕ , in degrees.

End (*end*) [real number].

End value of the azimuthal coordinate ϕ , in degrees.

Np (*np*) [integer].

Number of ϕ -values. When Cut Type is specified to 'radial', np is the number of radial cuts. When Cut Type is specified to 'circular', np is the number of ϕ -values in each circular cut.

E/H-Field (*e_h*) [item from a list of character strings], default: **e_field**.

Specifies whether the complex *E*-field or *H*-field shall be calculated.

e_field

The complex E-field is calculated.

h_field

The complex H-field is calculated.

Polarisation (*polarisation*) [item from a list of character strings], default: **linear**.

Defines how the calculated field shall be decomposed in two field components parallel to the plane. Additionally, the z -component of the field is calculated as a third component. All coordinates refer to the output coordinates defined by *coor_sys*.

linear

The linear components are the field components with the first component along x and the second component along y . These components are E_{co} and E_{cx} , respectively, in the determination of the circular components.

circular

Circular components are calculated based on the linear components, E_{co} and E_{cx} , defined above. The first component is right hand circular (E_{rhc}) and the second is left hand circular component (E_{lhc}).

rho_phi

The field is decomposed along the ρ - and ϕ -unit vectors with the ρ -component (E_ρ) being the first component and the ϕ -component being the second (E_ϕ).

major_minor

The field is decomposed along the major and minor axes of the polarisation ellipse. The first field component is parallel to the major axis (E_{maj}) and the second to the minor axis (E_{min}).

linear_xpd

The ratios E_{co}/E_{cx} and E_{cx}/E_{co} , where E_{co} and E_{cx} are the first and second components as defined for the '*polarisation: linear*'. This is the linear cross-polar discrimination ratio.

circular_xpd

The ratios E_{rhc}/E_{lhc} and E_{lhc}/E_{rhc} , where E_{rhc} and E_{lhc} are the first and second components as defined for the '*polarisation: circular*'. This is the circular cross-polar discrimination ratio.

rho_phi_xpd

The ratios E_ρ/E_ϕ and E_ϕ/E_ρ , where E_ρ and E_ϕ are the first and second components as defined for the '*polarisation: rho_phi*'.

major_minor_xpd

The ratios E_{maj}/E_{min} and E_{min}/E_{maj} , where E_{maj} and E_{min} are the first and second components as defined for the '*polarisation: major_minor*' above.

power

The first component is the amplitude of the field,

$$|\vec{E}| = \sqrt{|E_{co}|^2 + |E_{cx}|^2 + |E_z|^2} \quad (1)$$

E_z being the z -component of the field (i.e. the square root of the power of all three field components) and the second component is the complex square root $\sqrt{E_{rhc}/E_{lhc}}$. The phase of the latter component is the rotation angle of the polarisation ellipse.

File Name (*file_name*) [file name].

Name of a file, to which the calculated field values shall be written.

File Format (*file_format*) [item from a list of character strings], default: **TICRA**.

The file format used to store the field data:

TICRA

The field is written in GRASP units and stored as an ASCII-file according to the TICRA-format described in *Field Data in Cuts*. The recommended file extension is *.cut*.

EDX

The field is written in SI units in a format according to the Electromagnetic Data Exchange (EDX) standard. The recommended file extension is *.cut*.

Files in this format cannot be read by the PostProcessor.

EDI

Obsolete file format name. The same as EDX.

Comment (*comment*) [character string], default: **Field data in cuts**.

A line of text which will be written as a header in the file specified by *file_name* above.

Frequency (*frequency*) [name of an object], default: **blank**.

Reference to a *Frequency* object, defining the frequencies for which the field is calculated. The reference must be to the same object as specified in the *frequency* attribute of the source generating the field. A *frequency* needs not to be specified. In that case, the frequencies in the *frequency* object of the source generating the field will be applied.

Command Types

The *Planar Cut* class is derived from the class *Field Storage*. See this class for available commands.

Remarks

A planar cut is defined by the points

$$\bar{r}(\rho, \phi) = \hat{x}\rho \cos \phi + \hat{y}\rho \sin \phi + \hat{z}z_d$$

where z_d is the distance (given by *near_dist*) to the near-field plane defined by the attribute *near_dist*. The attributes *rho_range* and *phi_range* define the ρ - and ϕ -values.

In a radial cut, ϕ is fixed and ρ runs through the values

$$\rho_i = \rho_{start} + \Delta\rho \cdot (i - 1), \quad i = 1, 2, \dots, n_\rho \quad (2)$$

with

$$\Delta\rho = (\rho_{end} - \rho_{start})/(n_\rho - 1)$$

where ρ_{start} and ρ_{end} are the members *start* and *end*, respectively, of the attribute *rho_range*, and n_ρ is the member *np* of the same attribute.

The ϕ -angle is increased equidistantly from one cut to the next by the values

$$\phi_j = \phi_{start} + \Delta\phi \cdot (j - 1), \quad j = 1, 2, \dots, n_\phi \quad (3)$$

with

$$\Delta\phi = (\phi_{end} - \phi_{start})/(n_\phi - 1)$$

where ϕ_{start} and ϕ_{end} are the members *start* and *end*, respectively, of the attribute *phi_range*, and n_ϕ is the member *np* of the same attribute.

A circular cut means that ρ is fixed in each cut and ϕ runs through the values given by Eq. (3). From one circular cut to the next the ρ -angle is increased according to Eq. (2).

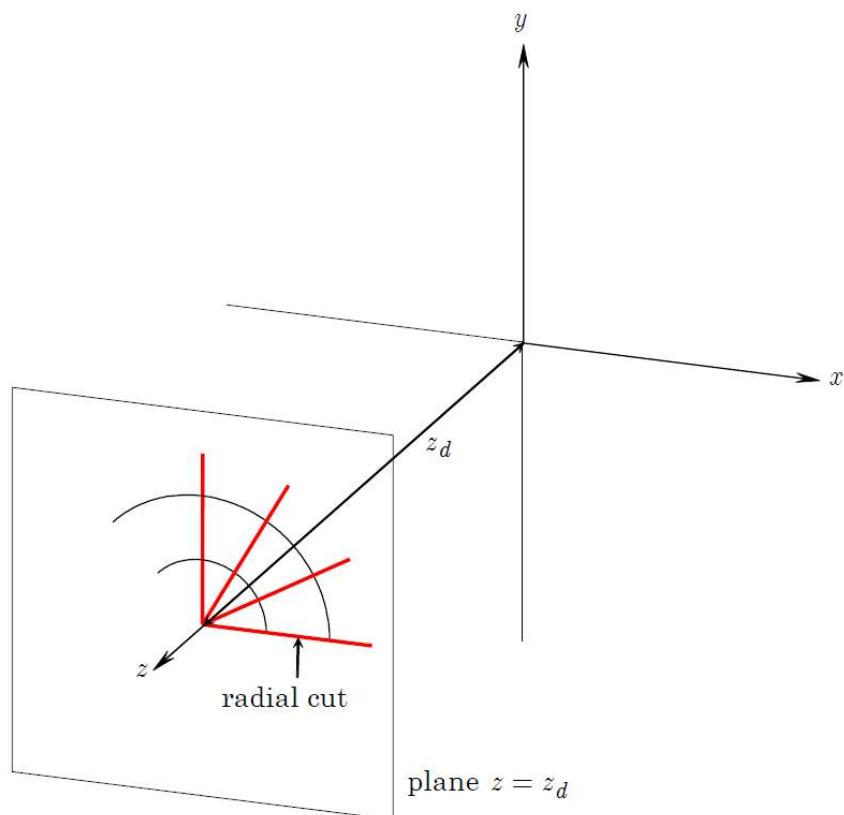


Figure 1 4 radial cuts in a plane at $z_d = 8$ m for $0 \leq \rho \leq 2.6$ m; and $\phi = 0^\circ, 30^\circ, 60^\circ$ and 90° .

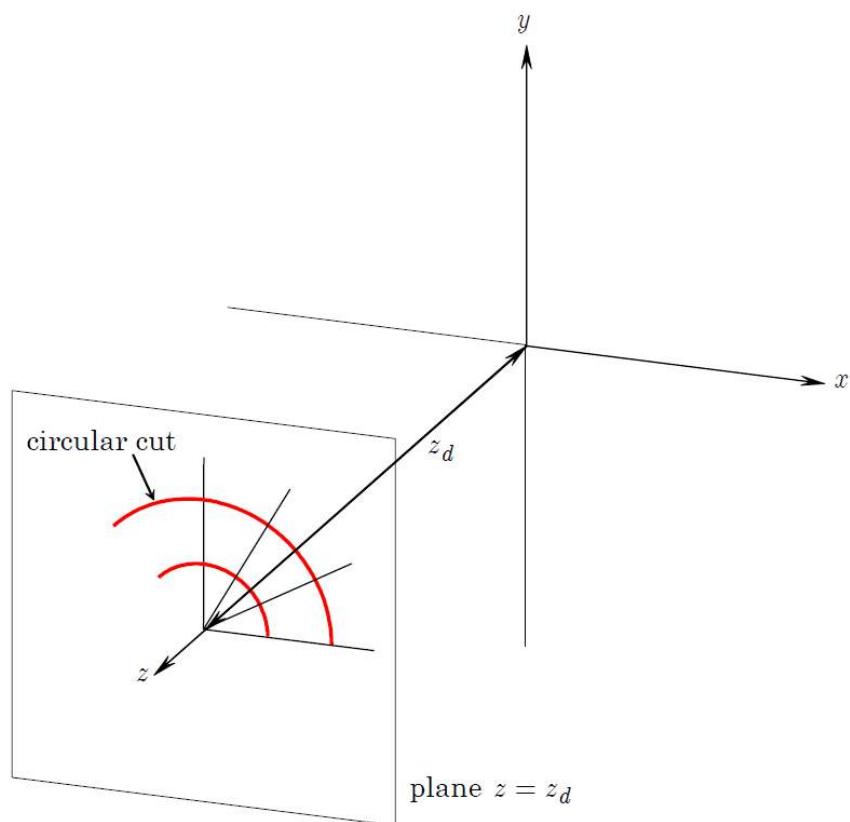


Figure 2

2 circular cuts in a plane at $z_d = 8$ m for $\rho = 1$ and 2 m;
and $0^\circ \leq \phi \leq 135^\circ$.

CYLINDRICAL CUT (cylindrical_cut)

Purpose

The class *Cylindrical Cut* defines field points in cuts on a circular cylinder. Two types of cuts can be specified, axial and circular. The axis of the cylinder coincides with the z -axis of the coordinate system to which the output is referred. The cylinder surface is described by the parameters ϕ and z for constant value of ρ in a conventional circular cylindrical coordinate system. Thus, ϕ is a conventional azimuth angle measured in the xy -plane with the x -axis defining $\phi = 0$; ρ is the radius of the cylinder. Only near fields can be computed.

Links

[Classes](#)→[Electrical Objects](#)→[Field Storage](#)→[Cut](#)→[Cylindrical Cut](#)

Remarks

Syntax

```
<object name> cylindrical_cut
(
    coor_sys           : ref(<n>),
    radius             : <rl>,
    cut_type           : <si>,
    z_range            : struct(start:<r>, end:<r>, np:<i>,
                                z_unit:<si>),
    phi_range          : struct(start:<r>, end:<r>, np:<i>),
    e_h                : <si>,
    polarisation       : <si>,
    file_name          : <f>,
    file_format        : <si>,
    comment            : <s>,
    frequency          : ref(<n>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<f> = file name
<si> = item from a list of character strings
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system (the output coordinate system) in which the field points and the calculated field vectors will be expressed.

Radius (*radius*) [real number with unit of length].

Defines the radius of the cylinder.

Cut Type (*cut_type*) [item from a list of character strings], default: **axial**.

Defines the direction of the cuts on the cylinder. See also the remarks below.

axial

Axial cuts are parallel to the axis of the cylinder for constant values of the azimuth ϕ , z is varying.

circular

Circular cuts are cuts where z is constant and ϕ is varying, i.e. the cuts are perpendicular to the axis of the cylinder.

z-Range (*z_range*) [struct].

Defines the range of the axial coordinate z (see the remarks below).

Start (*start*) [real number].

Start value of the axial coordinate z .

End (*end*) [real number].

End value of the axial coordinate z .

Np (*np*) [integer].

Number of z -values. When Cut Type is specified to 'axial', np is the number of z -values in each axial cut. When Cut Type is specified to 'circular' np is the number of circular cuts.

z-Unit (*z_unit*) [item from a list of character strings], default: **m**.

Defines the unit of length in which the z-Range is given (mm, cm, m, km, in, ft).

phi-Range (*phi_range*) [struct].

Defines the range of the azimuthal angle ϕ (see the remarks below).

Start (*start*) [real number].

Start value of the azimuthal coordinate ϕ , in degrees.

End (*end*) [real number].

End value of the azimuthal coordinate ϕ , in degrees.

Np (*np*) [integer].

Number of ϕ -values. When Cut Type is specified to 'axial', np is the number of axial cuts. When Cut Type is specified to 'circular', np is the number of ϕ -values in each circular cut.

E/H-Field (*e_h*) [item from a list of character strings], default: **e_field**.

Specifies whether the complex *E*-field or *H*-field shall be calculated.

e_field

The complex E-field is calculated.

`h_field`

The complex H-field is calculated.

Polarisation (*polarisation*) [item from a list of character strings], default: **linear**.

Defines how the calculated field shall be decomposed in two field components parallel to the surface of the cylinder. Additionally, the ρ -component of the field is calculated as a third component. All coordinates refer to the output coordinates defined by *coor_sys*.

`linear`

The linear components are the field components with the first component along ϕ and the second component along z . These components are E_{co} and E_{cx} , respectively, in the determination of the circular components.

`circular`

Circular components are calculated based on the linear components, E_{co} and E_{cx} , defined above. The first component is right hand circular (E_{rhc}) and the second is left hand circular component (E_{lhc}).

`major_minor`

The field is de-composed along the major and minor axes of the polarisation ellipse. The first field component is parallel to the major axis (E_{maj}) and the second to the minor axis (E_{min}).

`linear_xpd`

The ratios E_{co}/E_{cx} and E_{cx}/E_{co} , where E_{co} and E_{cx} are the first and second components as defined for the '*polarisation: linear*' above. This is the linear cross-polar discrimination ratio.

`circular_xpd`

The ratios E_{rhc}/E_{lhc} and E_{lhc}/E_{rhc} , where E_{rhc} and E_{lhc} are the first and second components as defined for the '*polarisation: circular*' above. This is the circular cross-polar discrimination ratio.

`major_minor_xpd`

The ratios E_{maj}/E_{min} and E_{min}/E_{maj} , where E_{maj} and E_{min} are the first and second components as defined for the '*polarisation: major_minor*' above.

power

The first component is the amplitude of the field,

$$|\vec{E}| = \sqrt{|E_{co}|^2 + |E_{cx}|^2 + |E_\rho|^2} \quad (1)$$

E_ρ being the ρ -component of the field (i.e. the square root of the power of all three field components) and the second component is the complex square root $\sqrt{E_{rhc}/E_{lhc}}$. The phase of the latter component is the rotation angle of the polarisation ellipse.

File Name (*file_name*) [file name].

Name of a file, to which the calculated field values shall be written.

File Format (*file_format*) [item from a list of character strings], default: **TICRA**.

The file format used to store the field data:

TICRA

The field is written in GRASP units and stored as an ASCII-file according to the TICRA-format described in *Field Data in Cuts*. The recommended file extension is *.cut*.

EDX

The field is written in SI units in a format according to the Electromagnetic Data Exchange (EDX) standard. The recommended file extension is *.cut*.

Files in this format cannot be read by the PostProcessor.

EDI

Obsolete file format name. The same as EDX.

Comment (*comment*) [character string], default: **Field data in cuts**.

A line of text which will be written as a header in the file specified by *file_name* above.

Frequency (*frequency*) [name of an object], default: **blank**.

Reference to a *Frequency* object, defining the frequencies for which the field is calculated. The reference must be to the same object as specified in the *frequency* attribute of the source generating the field. A *frequency* needs not to be specified. In that case, the frequencies in the *frequency* object of the source generating the field will be applied.

Command Types

The *Cylindrical Cut* class is derived from the class *Field Storage*. See this class for available commands.

Remarks

A cylindrical surface is defined in the usual cylindrical coordinates (ρ, ϕ, z) by the points

$$\vec{r}(\phi, z) = \hat{x}R \cos \phi + \hat{y}R \sin \phi + \hat{z}z$$

where $\rho = R$ is the fixed *radius* of the cylinder.

The attributes *phi_range* and *z_range* define the ranges for the ϕ - and z -values.

In an axial cut, ϕ is fixed and z runs through the values

$$z_i = z_{start} + \Delta z \cdot (i - 1), \quad i = 1, 2, \dots, n_z \quad (2)$$

with

$$\Delta z = (z_{end} - z_{start}) / (n_z - 1)$$

where z_{start} and z_{end} are the members *start* and *end*, respectively, of the attribute *z_range*, and n_z is the member *np* of the same attribute.

The ϕ -angle is increased equidistantly from one cut to the next by the values

$$\phi_j = \phi_{start} + \Delta \phi \cdot (j - 1), \quad j = 1, 2, \dots, n_\phi \quad (3)$$

with

$$\Delta \phi = (\phi_{end} - \phi_{start}) / (n_\phi - 1)$$

where ϕ_{start} and ϕ_{end} are the members *start* and *end*, respectively, of the attribute *phi_range*, and n_ϕ is the member *np* of the same attribute.

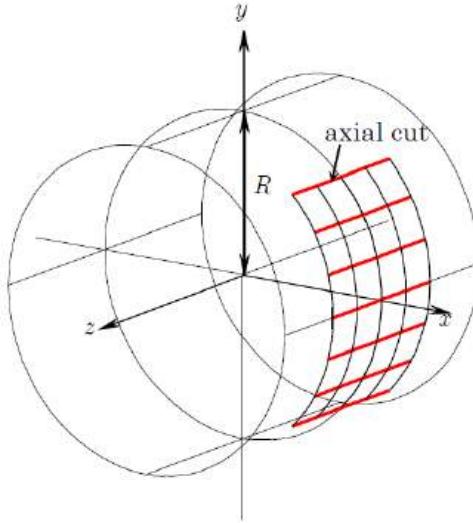


Figure 1 7 axial cuts on a cylinder with radius $R = 2$ m for $-1 \leq z \leq 1$ m; and $\phi = -45^\circ, -30^\circ, -15^\circ, 0^\circ, 15^\circ, 30^\circ$ and 45° .

A circular cut means that z is fixed in each cut and ϕ runs through the values given by Eq. (3). From one circular cut to the next, the z -value is increased according to Eq. (2), as shown in Figure 2.

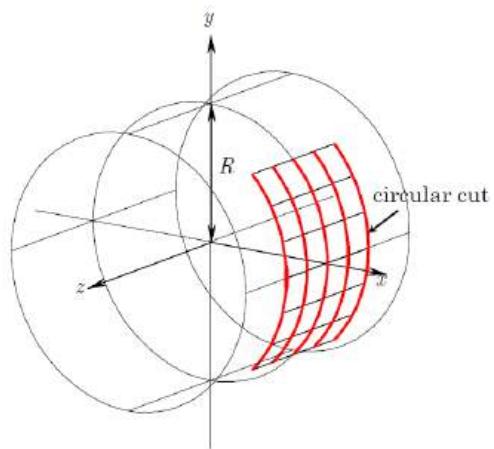


Figure 2

5 circular cuts on a cylinder with radius $R = 2$ m for $z = -1, -0.5, 0, 0.5$ and 1 m; and $-45^\circ \leq \phi \leq 45^\circ$.

SURFACE CUT (surface_cut)

Purpose

The class *Surface Cut* defines field points in cuts on a *Reflector* or a *Plate* (*Triangular Plate*, *Parallelogram* and *Rectangular Plate*) where the field shall be calculated. Radial as well as circular cuts can be specified. This class is useful for calculating an aperture field or displaying the PO current distribution on a reflector or a plate, see the remarks below.

Links

[Classes](#)→[Electrical Objects](#)→[Field Storage](#)→[Cut](#)→[Surface Cut](#)

Remarks

Syntax

```
<object name> surface_cut
(
    scatterer           : ref(<n>),
    cut_type            : <si>,
    rho_range           : struct(start:<r>, end:<r>, np:<i>,
                                  rho_unit:<si>),
    phi_range           : struct(start:<r>, end:<r>, np:<i>),
    field_type          : <si>,
    polarisation        : <si>,
    polarisation_and_phase_coor_sys: ref(<n>),
    phase_adjustment   : <si>,
    centre_offset       : struct(x:<rl>, y:<rl>),
    file_name           : <f>,
    comment             : <s>,
    frequency           : ref(<n>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<f> = file name
<si> = item from a list of character strings
```

Attributes

Scatterer (*scatterer*) [name of an object].

Reference to a *Reflector* object or a *Plate* object (*Triangular Plate*, *Parallelogram* and *Rectangular Plate*) on which the surface field is calculated.

Cut Type (*cut_type*) [item from a list of character strings], default: **radial**.

Defines the type of cuts. The cuts are for a reflector defined in the coordinate system of the reflector and for a plane in the local coordinate system in the plane of the plate (see also the remarks below).

radial

Radial cuts are made for constant values of the azimuth angle ϕ , while the radius ρ is varying.

circular

Circular cuts are cuts where ρ is constant and ϕ is varying.

rho-Range (*rho_range*) [struct].

Defines the range of the radial coordinate ρ , see the remarks below.

Start (*start*) [real number].

Start value of the radial coordinate ρ .

End (*end*) [real number].

End value of the radial coordinate ρ .

Np (*np*) [integer].

Number of ρ -values. When Cut Type is specified to 'radial', np is the number of ρ -values in each radial cut. When Cut Type is specified to 'circular', np is the number of circular cuts.

rho-Unit (*rho_unit*) [item from a list of character strings], default: **m.**

Defines the unit of length in which the rho-Range is given (mm, cm, m, km, in, ft).

phi-Range (*phi_range*) [struct].

Defines the range of the azimuthal angle ϕ , see the remarks below.

Start (*start*) [real number].

Start value of the azimuthal coordinate ϕ , in degrees.

End (*end*) [real number].

End value of the azimuthal coordinate ϕ , in degrees.

Np (*np*) [integer].

Number of ϕ -values. When Cut Type is specified to 'radial', np is the number of radial cuts. When Cut Type is specified to 'circular', np is the number of ϕ -values in each circular cut.

Field Type (*field_type*) [item from a list of character strings], default: **incident_e_field.**

Different field types may be determined. The field vector is the normal vector to the front surface of the scatterer.

incident_e_field

The incident E-field on the surface is determined.

incident_h_field

The incident H-field on the surface is determined.

reflected_e_field

The reflected E-field, \bar{E}_r , on the surface is determined according to the formula $\bar{E}_r = 2\hat{n}(\hat{n} \cdot \bar{E}_i) - \bar{E}_i$, where \bar{E}_i is the incident E-field.

reflected_h_field

The reflected H-field, \bar{H}_r , on the surface is determined according to the formula $\bar{H}_r = \bar{H}_i - 2\hat{n}(\hat{n} \cdot \bar{H}_i)$, where \bar{H}_i is the incident H-field.

currents

The PO currents are determined as $2\hat{n} \times \bar{H}_i$ where \bar{H}_i is the incident H-field.

Polarisation (polarisation) [item from a list of character strings], default: **linear**.

Defines how the calculated field shall be decomposed in two field components parallel to the plane. Additionally, the z -component of the field is calculated.

linear

The linear components E_x and E_y are the field components with the first component along x and the second component along y of the *polarisation_and_phase_coor_sys*.

circular

Circular components are calculated based on the linear components, E_x and E_y , defined above. The first component is right hand circular (E_{rhc}) and the second is left hand circular component (E_{lhc}).

rho_phi

The field is decomposed along the ρ - and ϕ -unit vectors of the *polarisation_and_phase_coor_sys*. The ρ -component (E_ρ) is the first field component and the ϕ -component is the second (E_ϕ).

major_minor

The field is decomposed along the major and minor axes of the polarisation ellipse. The first field component is parallel to the major axis (E_{maj}) and the second to the minor axis (E_{min}).

linear_xpd

The ratios E_x/E_y and E_y/E_x , where E_x and E_y are the first and second components as defined for the 'polarisation: linear'. This is the linear cross-polar discrimination ratio.

circular_xpd

The ratios E_{rhc}/E_{lhc} and E_{lhc}/E_{rhc} , where E_{rhc} and E_{lhc} are the first and second components as defined for the '*polarisation: circular*'. This is the circular cross-polar discrimination ratio.

rho_phi_xpd

The ratios E_ρ/E_ϕ and E_ϕ/E_ρ , where E_ρ and E_ϕ are the first and second components as defined for the '*polarisation: rho_phi*'.

major_minor_xpd

The ratios E_{maj}/E_{min} and E_{min}/E_{maj} , where E_{maj} and E_{min} are the first and second components as defined for the '*polarisation: major_minor*'.

power

The first component is the amplitude of the field,

$$|\vec{E}| = \sqrt{|E_x|^2 + |E_y|^2 + |E_z|^2} \quad (1)$$

E_z being the z -component of the field in the *polarisation_and_phase_coor_sys* (i.e. it is the square root of the power of all three field components) and the second component is the complex square root $\sqrt{E_{rhc}/E_{lhc}}$. The phase of the latter component is the rotation angle of the polarisation ellipse.

Polarisation and Phase Coor Sys (*polarisation_and_phase_coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate System*. The calculated field vectors on the surface will be resolved in components according to the axes of this coordinate system. The x - and y -components of the field may be converted according to the specified polarisation as described below whereas the z -component is unchanged. If a phase adjustment is required the phase of the field will be modified as described in the remarks below.

Phase Adjustment (*phase_adjustment*) [item from a list of character strings], default: **off**.

A phase adjustment of the calculated field is possible.

off

No phase adjustment is performed.

on

The phase of the field on the surface is adjusted as explained in the remarks below.

Centre Offset (*centre_offset*) [struct].

The origin of the polar grid may be translated according to this attribute. For a reflector the default origin is the centre of the reflector rim, as defined in the *Rim* object; for a plate the default origin is that of the local coordinate system in the plane of the plate (see also the remarks below).

x (*x*) [real number with unit of length], default: **0**.

x-coordinate of the origin.

y (*y*) [real number with unit of length], default: **0**.

y-coordinate of the origin.

File Name (*file_name*) [file name].

Name of a file, to which the calculated field values shall be written. The content of the file is described in *Field Data in Cuts*. The recommended file extension is *.cut*.

Comment (*comment*) [character string], default: **Field data in cuts**.

A line of text which will be written as a header in the file specified by *file_name*.

Frequency (*frequency*) [name of an object], default: **blank**.

Reference to a *Frequency* object, defining the frequencies for which the field is calculated. The reference must be to the same object as specified in the *frequency* attribute of the source generating the field. A *frequency* needs not to be specified. In that case, the frequencies in the *frequency* object of the source generating the field will be applied.

Command Types

The *Surface Cut* class is derived from the class *Field Storage*. See this class for available commands.

Remarks

Definition of $\rho\phi$ -cuts

For a *Reflector* a surface cut is defined in polar coordinates (ρ, ϕ) in the *xy*-plane of the coordinate system of the *Reflector*. For a *Plate*, the surface cut is defined correspondingly in the *xy*-plane in the local coordinate system in the plane of the *Plate*, see the individual sub-classes of *Plate*.

The reflector surface may be described by means of ρ and ϕ as

$$\bar{r}(\rho, \phi) = \hat{x}\rho \cos \phi + \hat{y}\rho \sin \phi + \hat{z}z(x, y)$$

where $z(x, y)$ defines the *Surface* of the reflector for

$$x = \rho \cos \phi + x_c + x_0, y = \rho \sin \phi + y_c + y_0$$

(x_c, y_c) are the coordinates of the centre of the *Rim* of the reflector and (x_0, y_0) are the offset coordinates as defined in the attribute *centre_offset*.

The coordinates x and y are given in the above mentioned coordinate system.

In a radial cut, ϕ is fixed and ρ runs through the values

$$\rho_i = \rho_{start} + \Delta\rho \cdot (i - 1), \quad i = 1, 2, \dots, n_\rho \quad (2)$$

with

$$\Delta\rho = (\rho_{end} - \rho_{start})/(n_\rho - 1)$$

where ρ_{start} and ρ_{end} are the members *start* and *end*, respectively, of the attribute *rho_range*, and n_ρ is the member *np* of the same attribute.

The ϕ -angle is increased equidistantly from one cut to the next by the values

$$\phi_j = \phi_{start} + \Delta\phi \cdot (j - 1), \quad j = 1, 2, \dots, n_\phi \quad (3)$$

with

$$\Delta\phi = (\phi_{end} - \phi_{start})/(n_\phi - 1)$$

where ϕ_{start} and ϕ_{end} are the members *start* and *end*, respectively, of the attribute *phi_range*, and n_ϕ is the member *np* of the same attribute.

A circular cut means that ρ is fixed in each cut and ϕ runs through the values given by Eq. (3). From one circular cut to the next, the ρ -angle is increased according to Eq. (2).

Phase adjustment

If *phase_adjustment* is set to 'off', the field is computed at the surface point R , cf. the figure below.

If *phase_adjustment* is set to 'on', the computed field on the surface at point R is multiplied by e^{-jkd} , where k is the wavenumber and d is the distance from R to the xy -plane of the *polarisation_and_phase_coor_sys*, which is denoted by the subscript pp in the figure.

Setting *phase_adjustment* to 'on' means that the phase of the calculated field is the one it would have on the xy -plane of the *polarisation_and_phase_coor_sys* if the field propagated along parallel rays in direction z_{pp} .

The distance d becomes negative if the plane is located behind the reflector.

In Figure 1, an offset paraboloidal reflector with focus at F is illuminated by a feed, which is positioned below this focus. Rays from the feed, reflected in the reflector, are shown.

When the *phase_adjustment* is 'off', the field is determined at points on the reflector, the field components being parallel to x_{pp} and y_{pp} , which are x and y of the *polarisation_and_phase_coor_sys*. The field amplitudes will illustrate the field intensity, but the phase will vary drastically as the ray path length for the different rays to the reflector varies.

When the *phase_adjustment* is 'on', the field is determined at the same points but phase-corrected by the distance d to the xy -plane of the *polarisation_and_phase_coor_sys*, the coordinates of which are denoted x_{pp} , y_{pp} and z_{pp} . When this coordinate system is tilted to having the z_{pp} -axis parallel to the reflected rays, the corrected phase is nearly constant as all ray

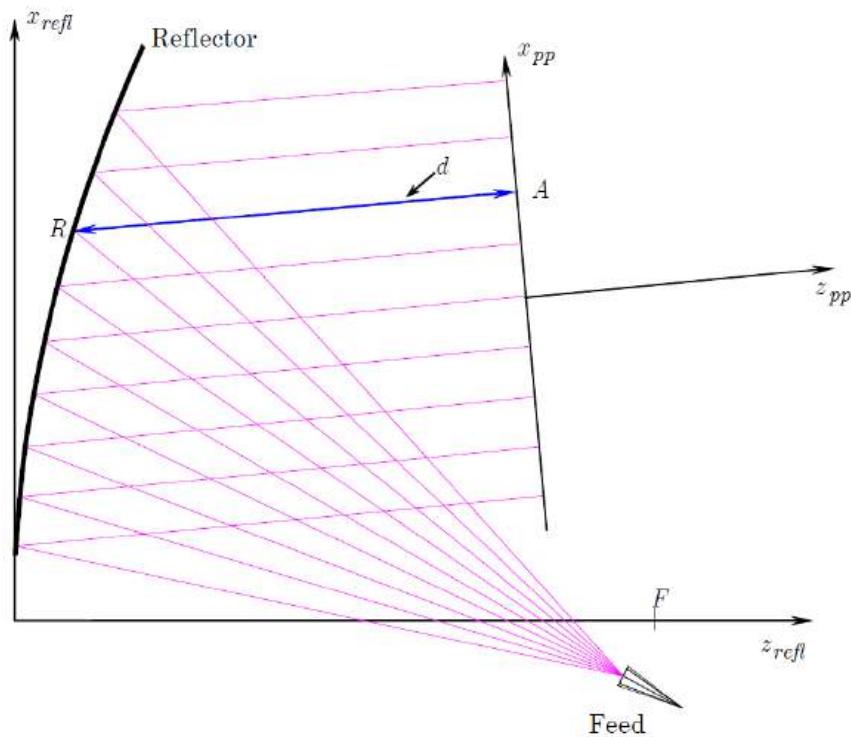


Figure 1 Example illustrating the *phase_adjustment*.

paths from the feed via the reflector to the x_{pp}, y_{pp} -plane are nearly of the same lengths. As the reflector is defocused, the phase will not be precisely uniform, but the deviations may easily be evaluated in this way.

The field components are again parallel to x_{pp} and y_{pp} , x and y of the *polarisation_and_phase_coor_sys*. The third component in the file is the component parallel to z_{pp} .

GRID

Purpose

The menu *Grid* contains classes that define regular output grids for field storage.

Field upon a surface in space:

Spherical Grid (includes far-field grids and uv-grids)

Planar Grid

Cylindrical Grid

Field at the surface of a scatterer:

Surface Grid

Links

Classes→*Electrical Objects*→*Field Storage*→*Grid*

SPHERICAL GRID (spherical_grid)

Purpose

The class *Spherical Grid* defines field points in a 2D grid on a sphere where the field shall be calculated. Both near fields and far fields may be calculated.

Links

Classes→*Electrical Objects*→*Field Storage*→*Grid*→*Spherical Grid*

Remarks

Syntax

```

<object name> spherical_grid
(
    coor_sys           : ref(<n>),
    grid_type          : <si>,
    x_range            : struct(start:<r>, end:<r>, np:<i>),
    y_range            : struct(start:<r>, end:<r>, np:<i>),
    truncation         : <si>,
    e_h                : <si>,
    polarisation       : <si>,
    polarisation_modification : struct(status:<si>, coor_sys:ref(<n>)),
    near_far           : <si>,
    near_dist          : <rl>,
    file_name          : <f>,
    file_format        : <si>,
    comment             : <s>,
    beam_grid_file     : <f>,
    frequency          : ref(<n>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<f> = file name
<si> = item from a list of character strings

```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system (the output coordinate system) in which the field points will be calculated. Further, the origin serves as the phase reference for far fields. The field polarisation components will also be

expressed in this coordinate system unless otherwise specified in the attribute Polarisation Modification.

Grid Type (*grid_type*) [item from a list of character strings], default: **uv**.

The field points are positioned in a 2D grid over the spherical surface. The 2D grid is defined by the variables *X* and *Y* (see X-Range and Y-Range) according to the following definitions (see also the figures in the remarks below).

uv

$(X, Y) = (u, v)$ where *u* and *v* are the two first coordinates of the unit vector to the field point. Hence,

$$\hat{r} = (u, v, \sqrt{1 - u^2 - v^2}) \quad (1)$$

u and *v* are related to the spherical angles by $u = \sin \theta \cos \phi, v = \sin \theta \sin \phi$. *u* and *v* have no units.

elevation_over_azimuth

$(X, Y) = (Az, El)$, where *Az* and *El* defines the direction to the field point by $\hat{r} = (-\sin Az \cos El, \sin El, \cos Az \cos El)$. *Az* and *El* are angles in degrees. Note that an antenna will be measured in such a grid (with respect to the antenna) applying an azimuth-over-elevation set-up.

Can only be used when File Format is specified to 'TICRA'.

elevation_and_azimuth

$(X, Y) = (Az, El)$, where *Az* and *El* defines the direction to the field point through the relations $Az = -\theta \cos \phi, El = \theta \sin \phi$ to the spherical angles θ and ϕ . *Az* and *El* are, as θ , angles in degrees.

Can only be used when File Format is specified to 'TICRA'.

azimuth_over_elevation

$(X, Y) = (Az, El)$, where *Az* and *El* defines the direction to the field point by $\hat{r} = (-\sin Az, \cos Az \sin El, \cos Az \cos El)$. *Az* and *El* are angles in degrees. Note that an antenna will be measured in such a grid (with respect to the antenna) applying an elevation-over-azimuth set-up.

Can only be used when File Format is specified to 'TICRA'.

theta_phi

$(X, Y) = (\phi, \theta)$, where θ and ϕ are the usual spherical angles of the direction to the field point. θ and ϕ are angles in degrees.

elevation_over_azimuth_EDX

$(X, Y) = (Az, El)$, where Az and El defines the direction to the field point by $\hat{r} = (\sin Az, \cos Az \sin El, \cos Az \cos El)$. Az and El are angles in degrees. Note that an antenna will be measured in such a standard grid (with respect to the antenna) applying a real elevation-over-azimuth set-up.

Can only be used when File Format is specified to 'EDX'.

azimuth_over_elevation_EDX

$(X, Y) = (Az, El)$, where Az and El defines the direction to the field point by $\hat{r} = (\sin Az \cos El, \sin El, \cos Az \cos El)$. Az and El are angles in degrees. Note that an antenna will be measured in such a standard grid (with respect to the antenna) applying a real azimuth-over-elevation set-up.

Can only be used when File Format is specified to 'EDX'.

X-Range (*x_range*) [struct].

Defines the range and number of points along the first grid coordinate, X , as specified by the attribute Grid Type above.

Start (*start*) [real number].

Start value of the first grid coordinate X .

End (*end*) [real number].

End value of the first grid coordinate X .

Np (*np*) [integer].

Number of field points along first grid coordinate X .

Y-Range (*y_range*) [struct].

Defines the range and number of points along the second grid coordinate, Y , as specified by the attribute Grid Type above.

Start (*start*) [real number].

Start value of the second grid coordinate Y .

End (*end*) [real number].

End value of the second grid coordinate Y .

Np (*np*) [integer].

Number of field points along second grid coordinate Y .

Truncation (*truncation*) [item from a list of character strings], default: **rect-angular**.

Specifies the area in which the field is calculated.

rectangular

The field is calculated in all of the grid points within the rectangular area defined by X-Range and Y-Range.

elliptical

The field is only calculated at the grid points inside the elliptical area with the axes of the ellipse defined by X-Range and Y-Range (in a plane rectangular XY-coordinate system).

E/H-Field (e_h) [item from a list of character strings], default: **e_field.**

Specifies the field type to be calculated.

e_field

The complex *E*-field is calculated.

h_field

The complex *H*-field is calculated.

Polarisation (polarisation) [item from a list of character strings], default: **linear.**

Defines how the calculated field shall be decomposed. All components refer to the output coordinate system given in Coordinate System unless a polarisation coordinate system as defined under the attribute Polarisation Modification. In the near field the *r*-component of the field is calculated as a third component.

linear

Linear components are calculated according to Ludwig's 3rd definition, with the first component (E_{co}) along *x* and the second component (E_{cx}) along *y* (at $\theta = 0^\circ$). The notation implies that for a field, which is mainly *y*-polarised then the second component (E_{cx}) represents the co-polar field component.

circular

Circular components are calculated based on the linear components defined above. The first component is the right hand circular (E_{rhc}) and the second is left hand circular component (E_{lhc}).

theta_phi

The field is decomposed along the θ - and ϕ -unit vectors with the θ -component (E_θ) being the first component and the ϕ -component being the second (E_ϕ).

major_minor

The field is decomposed along the major and minor axes of the polarisation ellipse. The first field component is parallel to the major axis (E_{maj}) and the second to the minor axis (E_{min}).

linear_xpd

The ratios E_{co}/E_{cx} and E_{cx}/E_{co} , where E_{co} and E_{cx} are the first and second components as defined for the '*polarisation*: linear'. This is the linear cross-polar discrimination ratio.

circular_xpd

The ratios E_{rhc}/E_{lhc} and E_{lhc}/E_{rhc} , where E_{rhc} and E_{lhc} are the first and second components as defined for the '*polarisation*: circular' above. This is the circular cross-polar discrimination ratio.

theta_phi_xpd

The ratios E_θ/E_ϕ and E_ϕ/E_θ , where E_θ and E_ϕ are the first and second components as defined for the '*polarisation*: theta_phi' above.

major_minor_xpd

The ratios E_{maj}/E_{min} and E_{min}/E_{maj} , where E_{maj} and E_{min} are the first and second components as defined for the '*polarisation*: major_minor' above.

power

The first component is the amplitude of the field, $|\vec{E}|$, (i.e. the square root of the power) and the second component is the complex square root $\sqrt{E_{rhc}/E_{lhc}}$. The phase of the latter component is the rotation angle of the polarisation ellipse.

In the far field the square root of the power is determined from

$$|\vec{E}| = \sqrt{|E_{co}|^2 + |E_{cx}|^2}$$

and in the near field it is determined from all three field components: $|\vec{E}| = \sqrt{|E_{co}|^2 + |E_{cx}|^2 + |E_r|^2}$, E_r being the r -component of the field.

Polarisation Modification (*polarisation_modification*) [struct].

Defines a coordinate system in which the field polarisation components are determined (if different from the output coordinate system defined in attribute Coordinate System above).

Status (*status*) [item from a list of character strings], default: **off**.

Determines if the polarisation modification shall be performed:

off

No polarisation modification, the polarisation is defined in the above defined output coordinate system and the polarisation coordinate system is identical to the output coordinate system.

on

The polarisation is defined in the coordinate system defined next.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes in *Coordinate System* defining the coordinate system (the polarisation coordinate system) in which the polarisation components of the calculated field vectors will be expressed. Shall only be specified for *status*: on.

Near/Far (*near_far*) [item from a list of character strings], default: **far**.

The value specifies if a near or a far field is to be calculated.

far

A far field is calculated. In this case only two field components are calculated according to the specified Polarisation.

near

The three field components of a near field is calculated. These are the two components defined under the attribute Polarisation and the radial component.

Near-Field Distance (*near_dist*) [real number with unit of length], default: **0**.

Defines the radius of a near-field sphere. For a far field this attribute has no effect and needs not to be specified.

File Name (*file_name*) [file name].

Name of a file, to which the calculated field values shall be written.

File Format (*file_format*) [item from a list of character strings], default: **TICRA**.

The file format used to store the field data:

TICRA

The field is written in GRASP units and stored as an ASCII-file according to the TICRA-format described in *Field Data in Rectangular Grid*. The recommended file extension is *.grd*.

EDX

The field is written in SI units in a format according to the Electromagnetic Data Exchange (EDX) standard. The recommended file extension is *.grd*.

Files in this format cannot be read by the PostProcessor.

EDI

Obsolete file format name. The same as EDX.

Comment (*comment*) [character string].

A line of text which will be written as a header in the file specified by File Name above.

Beam Grid File (*beam_grid_file*) [file name].

Name of a file containing beam directions. The file is only used when the source generating the field is one of the array sources (see the listing of these under the menu point *Array*). When a file name is not specified, the field is determined at the *XY*-range given above without modifications. When the file is applied, each beam direction defines a translation of the above-specified grid such that the grid will become positioned relative to the beam direction specified in the file. The file shall for each direction contain an identification, which agrees with the element identification given in the relevant *Array* class. The beam directions in the file must be generated by the user. The format of the file is described in the section *Beam Grid Directions*.

Frequency (*frequency*) [name of an object], default: **blank**.

Reference to a *Frequency* object, defining the frequencies for which the field is calculated. The reference must be to the same object as specified in the *frequency* attribute of the source generating the field. A *frequency* needs not to be specified. In that case, the frequencies in the *frequency* object of the source generating the field will be applied.

Command Types

The *Spherical Grid* class is derived from the class *Field Storage*. See this class for available commands.

Remarks

The section introduces the grids specifying the positions of the field points and the definition of a polarisation coordinate system.

Field Points

The field points in a *Spherical Grid* are defined in usual spherical (θ, ϕ) -coordinates. For far fields, (θ, ϕ) defines a direction

$$\hat{r} = \hat{x} \sin \theta \cos \phi + \hat{y} \sin \theta \sin \phi + \hat{z} \cos \theta$$

and for near fields, (θ, ϕ) defines a point

$$\bar{R} = R(\hat{x} \sin \theta \cos \phi + \hat{y} \sin \theta \sin \phi + \hat{z} \cos \theta)$$

where $R = |\bar{R}|$ is the radius of the near-field sphere (as given by the attribute Near-Field Distance).

The unit vectors \hat{x} , \hat{y} and \hat{z} are the unit vectors along the axes of the output coordinate system as defined by the attribute *coor_sys*. In the following figures the axes are denoted x_o , y_o and z_o .

The field grid is defined by the ranges X-Range and Y-Range in a general *XY*-coordinate system where (X, Y) may take one of the definitions given by the attribute Grid Type. The field grid then consists of the points

$$X = X_s + \Delta X(i - 1) + X_1$$

$$Y = Y_s + \Delta Y(j - 1) + Y_1$$

where i and j takes on the values

$$i = 1, 2, \dots, N_x$$

$$j = 1, 2, \dots, N_y.$$

Moreover, X_s and Y_s are the Start values and N_x and N_y are the number of values, N_p , of the attributes X-Range and Y-Range, respectively, and ΔX and ΔY are the spacings in the grid

$$\Delta X = (X_e - X_s)/(N_x - 1)$$

$$\Delta Y = (Y_e - Y_s)/(N_y - 1)$$

where X_e and Y_e are the End values of the attributes X-Range and Y-Range, respectively.

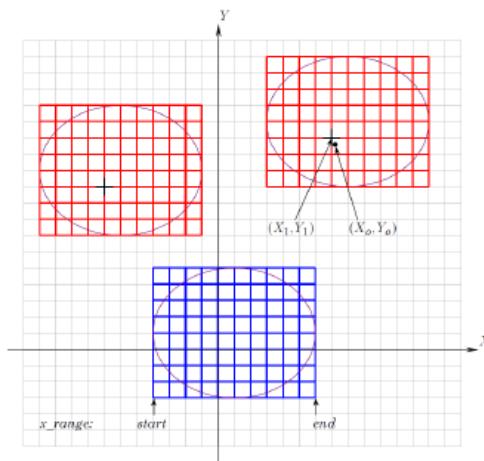


Figure 1

The XY coordinate system. A grid defined by X-Range and Y-Range is shown in blue. In the Beam Grid File two beams are defined, their grids, copies of the blue, are shown in red. For the right grid, the beam direction (X_0, Y_0) read from the file is shown along with the nearest grid point (X_1, Y_1) . The ranges of the grid are measured from this point. When the Truncation is 'rectangular' the full grids in red are included, and when the Truncation is 'elliptical' only the grid points within the shown ellipses are included.

The numbers X_1 and Y_1 are zero if a Beam Grid File is not defined. If Beam Grid File is defined, beam directions (X_0, Y_0) are read from the file and X_1 and Y_1 are defined by

$$X_1 = \text{NINT}(X_0/\Delta X) \cdot \Delta X,$$

$$Y_1 = \text{NINT}(Y_0/\Delta Y) \cdot \Delta Y.$$

Herein, the function $\text{NINT}(x)$ gives the integer number nearest to x . It is seen that a beam direction different from $(0,0)$ has the effect that the field grid is translated by an integer number of grid spacings so that the new grid is centred as close as possible around (X_0, Y_0) .

The Grids

The X and Y variables of the grid are related to the direction to the field points according to the selected value of the attribute Grid Type. The different possibilities are illustrated in the following figures. $x_0y_0z_0$ is the output coordinate system as defined by the attribute Coordinate System. Some grids are related to a scanning procedure, but for all grids the $x_0y_0z_0$ -coordinate system is fixed with respect to the antenna and follows this during the scanning.

For all the grids, the central direction given by $(X, Y) = (0, 0)$ is along the z_0 -axis, $\theta = 0$.

Note that many grid definitions involve the rotational angles azimuth and elevation. These grids are alike but have important differences which are explained in the following sections.

Grid Type: uv

The uv-grid constitutes a regular grid when projected to the x_0y_0 -plane. The far-field directions of this grid are obtained by projecting the grid to a unit-sphere, see the following Figure 2. This grid is the only grid not given in angles.

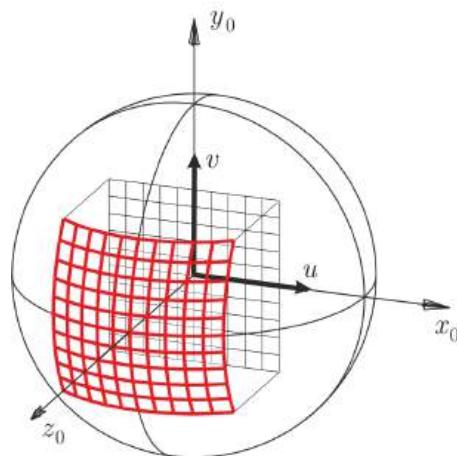


Figure 2

Grid Type: *uv*.

The *uv*-grid is that drawn in the x_0y_0 -plane, while the red grid upon the sphere (obtained by parallel projection) shows the far-field directions of the irregular angles corresponding to the regular *uv*-grid.

The grid is drawn for $-0.5 \leq u \leq 0.5$ and $-0.5 \leq v \leq 0.5$ with a spacing of 0.1 in both u and v .

Grid Type: *elevation_over_azimuth*

For the elevation-over-azimuth grid the direction to a field point can be visualized by a telescope mounted in an elevation-over-azimuth set-up at the origin of the $x_0y_0z_0$ -coordinate system. When rotated the angles (Az, El), the telescope will point at the field point given by the angles (Az, El). The grid has poles on the y_0 -axis, see Figure 3.

Apart from the positive direction of the azimuth rotation, the grid is the same as the *azimuth_over_elevation_EDX-grid* described below.

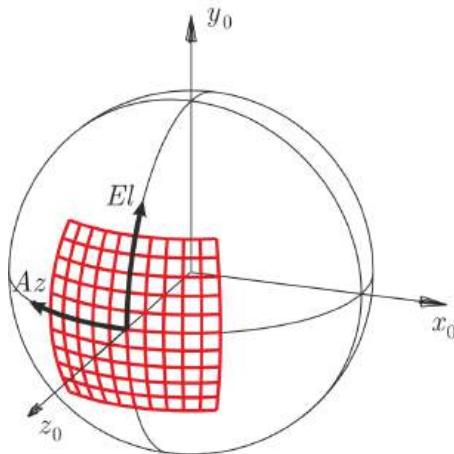


Figure 3

Grid Type: *elevation_over_azimuth*.

The grid is drawn for $-30^\circ \leq Az \leq 30^\circ$ and $-30^\circ \leq El \leq 30^\circ$ with a spacing of 6° in both Az (azimuth) and El (elevation).

Grid Type: *elevation_and_azimuth*

This grid treats azimuth and elevation symmetrically but is not related to physical rotations. The grid is shown in Figure 4.

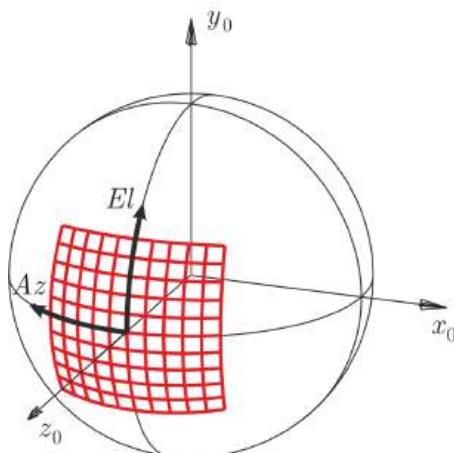


Figure 4

Grid Type: *elevation_and_azimuth*.

The grid is drawn for $-30^\circ \leq Az \leq 30^\circ$ and $0^\circ \leq El \leq 30^\circ$ with a spacing of 6° in both Az (azimuth) and El (elevation).

Grid Type: *azimuth_over_elevation*

Also for this grid the direction to a field point can be visualized by a telescope, now mounted in an azimuth-over-elevation set-up, placed at the origin of the $x_0y_0z_0$ -coordinate system. When rotated the angles (Az, El) , the

telescope will point at the field point given by the angles (Az, El). The grid has poles on the x_0 -axis, see Figure 5.

Apart from the positive direction of the azimuth rotation, the grid is the same as the *elevation_over_azimuth_EDX*-grid described below.

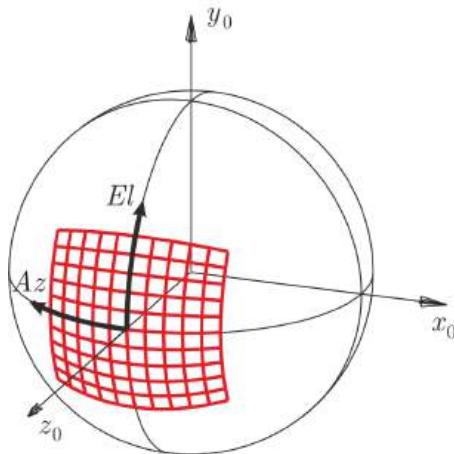


Figure 5

Grid Type: *azimuth_over_elevation*.

The grid is drawn for $-30^\circ \leq Az \leq 30^\circ$ and $-30^\circ \leq El \leq 30^\circ$ with a spacing of 6° in both Az (azimuth) and El (elevation).

Grid Type: *theta_phi*

This grid is a conventional grid in the spherical coordinates θ and ϕ with poles on the z_0 -axis (for $\theta = 0^\circ$ and $\theta = 180^\circ$), Figure 6.

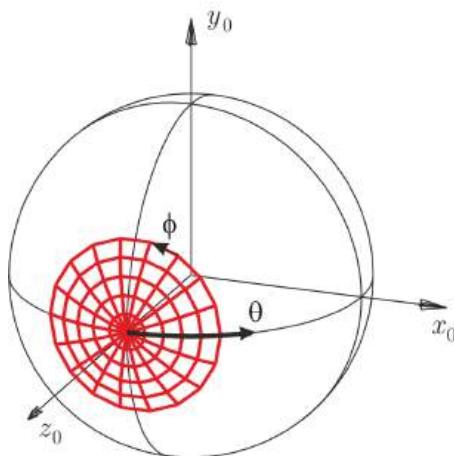


Figure 6

Grid Type: *theta_phi*.

The grid is drawn for $0^\circ \leq \theta \leq 30^\circ$ and $0^\circ \leq \phi \leq 360^\circ$ with a spacing of 6° in θ and 20° in ϕ .

Grid Type: *elevation_over_azimuth_EDX*

This grid is the grid in which an antenna will be sampled when it is mounted in a traditional elevation-over-azimuth (El/Az) scanner while the direction to the field point is kept constant as in a far-field range.

In order to illustrate the elevation-over-azimuth scanning, a reflector antenna is shown in the following figures. The scanner itself is not shown.

The grid type is one of the standard grid types in EDX¹. The directions in the grid are given by azimuth, Az , and elevation, El , alternatively denoted α and ε , respectively, in order to distinguish from the azimuth and elevation angles in the azimuth-over-elevation grid (see following grid type). As for all other grids, the grid follows the antenna during rotations.

The grid has poles on the x_0 -axis and is shown in Figure 7.

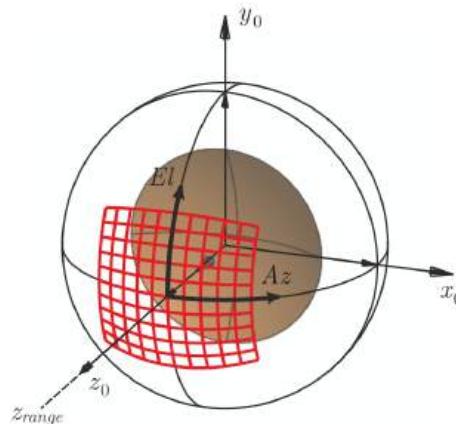


Figure 7

Grid Type: *elevation_over_azimuth_EDX* (El/Az or ε/α).

To illustrate that this grid corresponds to rotation of the antenna in an elevation-over-azimuth positioner, this and the following figures shows the antenna rotations, here before rotation, i.e. $Az = \alpha = 0^\circ$ and $El = \varepsilon = 0^\circ$. z_{range} indicates the direction to the field probe (the far-field direction).

The grid is drawn for $-30^\circ \leq Az (\alpha) \leq 30^\circ$ and $-30^\circ \leq El (\varepsilon) \leq 30^\circ$ with a spacing of 6° in both Az (or α , azimuth) and El (or ε , elevation).

The rotation of an antenna mounted in an elevation-over-azimuth set-up is in the following figures illustrated step by step. The antenna is a front-fed reflector antenna which initially (i.e. for $Az = 0^\circ$ and $El = 0^\circ$) is pointing in the far-field direction of the measurement range which is along the z -axis of the range, z_{range} . The antenna is given in its output coordinate system (attribute Coordinate System) $x_0y_0z_0$ with the main beam along the z_0 -axis. This initial situation is shown in Figure 7.

When this antenna is rotated in azimuth we get Figure 8. The antenna beam is rotated to the left in the figure as given by the angle Az which - in the grid fixed to the antenna - is positive to the right.

¹EDX is a definition of how electromagnetic data may be exchanged between various software tools, see "Electromagnetic Data Exchange Field Data Dictionary definition", European Space Agency and Satimo S.A., 2008

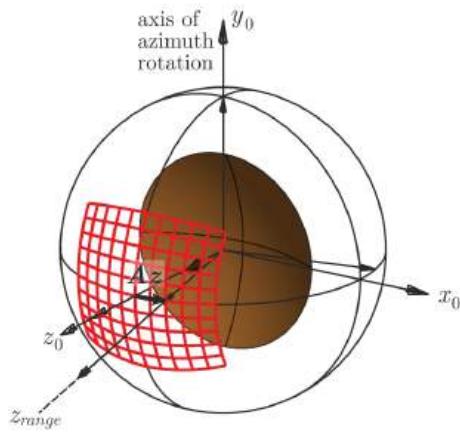


Figure 8 The reflector antenna is first rotated in azimuth. The grid spacing is 6° , thus $Az = \alpha = 12^\circ$.

The antenna is then tilted in elevation (upon the rotated azimuth platform), Figure 9. The antenna beam is hereby tilted down the angle El whereby the fixed far-field direction is moved the same angle El up with respect to the antenna.

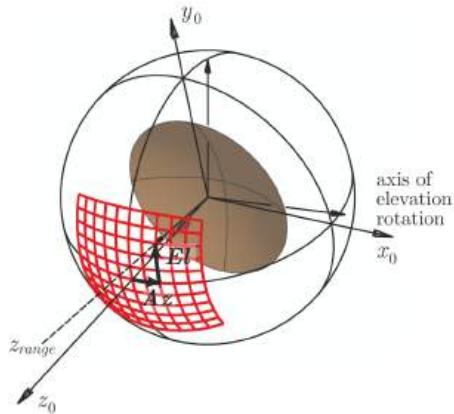


Figure 9 The reflector antenna is finally tilted in elevation. The grid spacing is 6° thus $El = \varepsilon = 18^\circ$ (and $Az = \alpha = 12^\circ$, unchanged).

In all the above figures, the range coordinate system is kept fixed (z_{range} points in the same direction) while the antenna is rotating as in a (imagined) physical scanner. If we instead show the antenna and the grid in a fixed antenna coordinate system, the $x_0y_0z_0$ -coordinate system, and move the field direction (represented by z_{range}) to the grid point we obtain Figure 10. This figure is thus shown in the same antenna perspective as Figure 7 but with a field direction corresponding to the grid point at $(Az, El) = (18^\circ, 12^\circ)$.

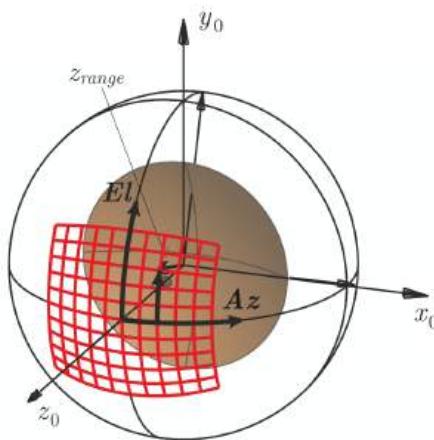


Figure 10

The grid is here shown with the field direction moved to $(Az, El) = (\alpha, \varepsilon) = (18^\circ, 12^\circ)$ but the antenna kept fixed, while in Figure 9 the antenna was moved. The antenna and the grid is therefore oriented in the same way as in Figure 7. The field direction is given by the z -axis of the range system, z_{range} , which here points near to directly out of the paper.

The *elevation_over_azimuth*-grid has poles on the x_0 -axis and is thus the same as the *azimuth_over_elevation*-grid illustrated in Figure 5 apart from the positive direction of the azimuth rotation (Az or α) which for the EDX grid increases to the right.

Grid Type: *azimuth_over_elevation_EDX*

This grid is the grid in which an antenna will be sampled when it is mounted in a traditional azimuth-over-elevation (Az/EI) scanner while the direction to the field point (Az, El) is kept constant as in a far-field range.

In order to illustrate the azimuth-over-elevation scanning, a reflector antenna is shown in the following figures. The scanner itself is not shown.

The grid type is one of the standard grid types in EDX². As for all other grids, the grid follows the antenna when it is rotated. The grid has poles on the y_0 -axis and is shown in Figure 11.

²EDX is a definition of how electromagnetic data may be exchanged between various software tools, see "Electromagnetic Data Exchange Field Data Dictionary definition", European Space Agency and Satimo S.A., 2008

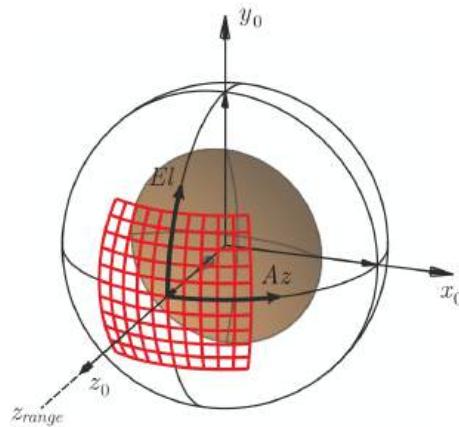


Figure 11

Grid Type: *azimuth_over_elevation_EDX* (Az/El).

To illustrate that this grid corresponds to rotation of the antenna in an azimuth-over-elevation positioner, this and the following figures shows the antenna rotations, here before rotation, i.e. $Az = 0^\circ$ and $El = 0^\circ$. z_{range} indicates the direction to the field probe.

The grid is drawn for $-30^\circ \leq Az \leq 30^\circ$ and $-30^\circ \leq El \leq 30^\circ$ with a spacing of 6° in both Az (azimuth) and El (elevation).

The rotation of an antenna mounted in an azimuth-over-elevation set-up is in the following figures illustrated step by step. The antenna is a front-fed reflector antenna which initially (i.e. for $Az = 0^\circ$ and $El = 0^\circ$) is pointing in the far-field direction of the measurement range which is along the z -axis of the range, z_{range} . The antenna is given in its output coordinate system (attribute Coordinate System) $x_0y_0z_0$ with the main beam along the z_0 -axis. This initial situation is shown in Figure 11.

If this antenna is tilted in elevation we get Figure 12. The antenna beam is here tilted down the angle El whereby the fixed far-field direction is moved the same angle El up with respect to the antenna.

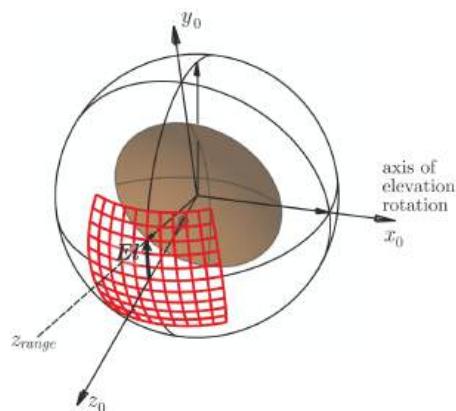


Figure 12

The reflector antenna is now tilted in elevation. The grid spacing is 6° thus $El = 18^\circ$ (and $Az = 0^\circ$).

Finally the antenna is rotated in azimuth upon the tilted elevation platform, Figure 13. The antenna beam is now rotated to the left as given by the angle Az which in the grid fixed to the antenna is positive to the right.

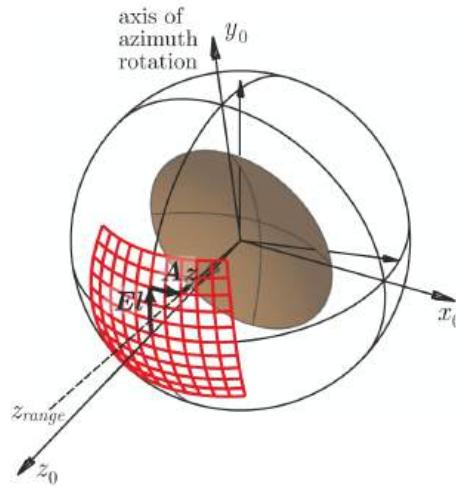


Figure 13

The reflector antenna is finally rotated in azimuth. The grid spacing is 6° thus $Az = 12^\circ$ (and $El = 18^\circ$, unchanged).

In all these figures, the range coordinate system is kept fixed (z_{range} points in the same direction) while the antenna is rotating as in a (imagined) physical scanner. If we instead show the antenna and the grid in a fixed antenna coordinate system, the $x_0y_0z_0$ -coordinate system, and move the field direction (represented by z_{range}) to the grid point we obtain Figure 14. This figure is thus shown in the same antenna perspective as Figure 11 but with the grid point at $(Az, El) = (18^\circ, 12^\circ)$.

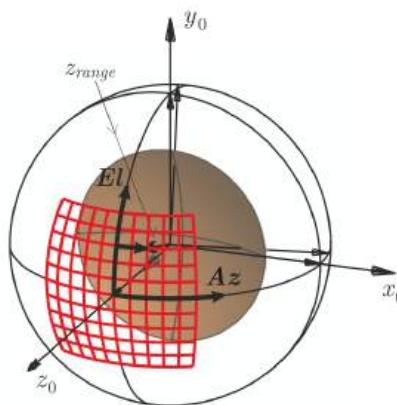


Figure 14

The grid is here shown with the field direction moved to $(Az, El) = (18^\circ, 12^\circ)$ and the antenna kept fixed, while in Figure 13 the antenna was moved. The antenna and the grid is therefore oriented in the same way as in Figure 11. The field direction is given by the z -axis of the range system, z_{range} , which here points out of the paper.

The *azimuth_over_elevation_EDX-grid* has poles on the y_0 -axis and is thus the same as the *elevation_over_azimuth-grid* illustrated in Figure 3 apart

from the positive direction of the azimuth rotation (Az) which for the EDX-grid increases to the right.

The Polarisation Coordinate System

When the field polarisation is requested in another coordinate system than the output coordinate system then the attribute *polarisation_modification* shall be applied with 'status: on' followed by a reference to the polarisation coordinate system.

The attribute *polarisation* may be specified as 'linear' (E_{co} - and E_{cx} -components), 'circular' (E_{rhc} - and E_{lhc} -components) or 'theta_phi' (E_θ - and E_ϕ -components). In the near field also an E_r -component will be present. In case of 'theta_phi' polarisation the electric field is given by

$$\bar{E} = E_r \hat{r} + E_\theta \hat{\theta} + E_\phi \hat{\phi}$$

where \hat{r} , $\hat{\theta}$ and $\hat{\phi}$ are the polarisation vectors in the polarisation coordinate system. The spherical grid in which the field is determined is always given in the output coordinate system.

An example is shown in Figure 15 in which only two perpendicular arcs of the grid are shown in red. The field is to be determined at the observation point P . The output coordinate system is denoted by $x_0y_0z_0$ and the polarisation coordinate system is $x_py_pz_p$. In the figure the latter is a simple rotation of the former around the y -axis but any coordinate system may be chosen as polarisation coordinate system.

Internally in GRASP the field is calculated in Cartesian components in the output coordinate system

$$\bar{E} = E_{ox} \hat{x}_0 + E_{oy} \hat{y}_0 + E_{oz} \hat{z}_0.$$

The field is then converted to the new components

$$\bar{E} = E_{px} \hat{x}_p + E_{py} \hat{y}_p + E_{pz} \hat{z}_p$$

before the Cartesian components are converted to (in this case) the polar components in the usual way.

The resulting θ_p -component (shown in blue) is pointing away from the z_p -axis in the same way as the standard θ_0 -component will point away from the z_0 -axis (along θ_0).

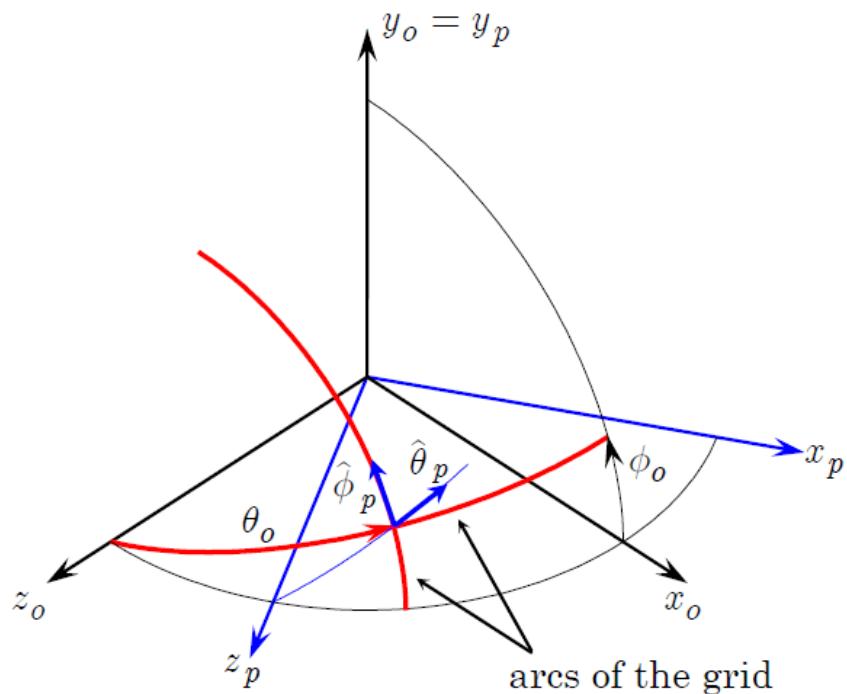


Figure 15

In red is shown two arcs of the spherical grid in the output coordinate system given by $x_0y_0z_0$. The observation point P is specified by the direction (θ_0, ϕ_0) . The polarisation is expressed in theta_phi components along $\hat{\theta}_p$ and $\hat{\phi}_p$ in the polarisation coordinate system $x_py_pz_p$.

PLANAR GRID (planar_grid)

Purpose

The class *Planar Grid* defines field points in a 2D grid on a plane where the field shall be calculated. The grid may be a rectangular xy -grid or it may be a similar grid in plane-polar coordinates, $\rho\phi$. Only near fields can be computed.

Links

Classes→*Electrical Objects*→*Field Storage*→*Grid*→*Planar Grid*

Remarks

Syntax

```
<object name> planar_grid
(
    coor_sys           : ref(<n>),
    near_dist          : <rl>,
    grid_type          : <si>,
    x_range            : struct(start:<r>, end:<r>, np:<i>,
                                unit:<si>),
    y_range            : struct(start:<r>, end:<r>, np:<i>),
    truncation         : <si>,
    e_h                : <si>,
    polarisation       : <si>,
    file_name          : <f>,
    file_format        : <si>,
    comment            : <s>,
    frequency          : ref(<n>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<f> = file name
<si> = item from a list of character strings
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system (the output coordinate system) in which the field points and the calculated field vectors will be expressed.

Near Dist (*near_dist*) [real number with unit of length], default: **0**.

Defines the distance along the z -axis of the output coordinate system (defined by *coor_sys*), from the origin to the plane in which the cuts are placed.

Grid Type (*grid_type*) [item from a list of character strings], default: **xy**.

The field points are positioned in a 2D grid over the plane. The 2D grid is defined by the variables X and Y (see X-Range and Y-Range) according to the following definitions (see also the figures in the remarks below).

xy

$(X, Y) = (x, y)$, the usual Cartesian coordinates.

rho_phi

$(X, Y) = (\rho, \phi)$, where ρ and ϕ are usual polar coordinates.

X-Range (*x_range*) [struct].

Defines the range (values and unit of length) and the number of points along the first grid coordinate, X , as specified by the attribute Grid Type above.

Start (*start*) [real number].

Start value of the first grid coordinate X .

End (*end*) [real number].

End value of the first grid coordinate X .

Np (*np*) [integer].

Number of field points along first grid coordinate X .

Unit (*unit*) [item from a list of character strings], default: **m**.

Defines the unit of length in which the lengths, ρ , x and y , are given (mm, cm, m, km, in, ft).

Y-Range (*y_range*) [struct].

Defines the range and number of points along the second grid coordinate, Y , as specified by the attribute Grid Type above. When Grid Type is 'rho_phi' then Y-Range specifies the angular ϕ -range, and start and end are given in degrees. When Grid Type is 'xy' then Y-Range specifies the y -range, and start and end are given in the unit of length defined by the struct member unit of attribute X-Range.

Start (*start*) [real number].

Start value of the second grid coordinate Y .

End (*end*) [real number].

End value of the second grid coordinate Y .

Np (*np*) [integer].

Number of field points along second grid coordinate Y .

Truncation (*truncation*) [item from a list of character strings], default: **rect-angular**.

Must be rectangular or elliptical.

rectangular

The field is calculated in all of the grid points within the rectangular area defined by *x_range* and *y_range*.

elliptical

The field is only calculated at the grid points inside the elliptical area with the axes of the ellipse defined by *x_range* and *y_range* (in a plane rectangular *XY* coordinate system).

E/H-Field (*e_h*) [item from a list of character strings], default: **e_field**.

Specifies whether the complex *E*-field or *H*-field shall be calculated.

e_field

The complex E-field is calculated.

h_field

The complex H-field is calculated.

Polarisation (*polarisation*) [item from a list of character strings], default: **linear**.

Defines how the calculated field shall be decomposed in two field components parallel to the plane. Additionally, the *z*-component of the field is calculated as a third component. All coordinates refer to the output coordinates defined by *coor_sys*.

linear

The linear components are the field components with the first component along *x* and the second component along *y*. These components are E_{co} and E_{cx} , respectively, in the determination of the circular components.

circular

Circular components are calculated based on the linear components defined above. The first component is the right hand circular (E_{rhc}) and the second is left hand circular component (E_{lhc}).

rho_phi

The field is decomposed along the ρ - and ϕ -unit vectors with the ρ -component (E_ρ) being the first component and the ϕ -component being the second (E_ϕ).

major_minor

The field is decomposed along the major and minor axes of the polarisation ellipse. The first field component is parallel to the major axis (E_{maj}) and the second to the minor axis (E_{min}).

linear_xpd

The ratios E_{co}/E_{cx} and E_{cx}/E_{co} , where E_{co} and E_{cx} are the first and second components as defined for the '*polarisation: linear*'. This is the linear cross-polar discrimination ratio.

circular_xpd

The ratios E_{rhc}/E_{lhc} and E_{lhc}/E_{rhc} , where E_{rhc} and E_{lhc} are the first and second components as defined for the '*polarisation: circular*' above. This is the circular cross-polar discrimination ratio.

rho_phi_xpd

The ratios E_ρ/E_ϕ and E_ϕ/E_ρ , where E_ρ and E_ϕ are the first and second components as defined for the '*polarisation: rho_phi*' above.

major_minor_xpd

The ratios E_{maj}/E_{min} and E_{min}/E_{maj} , where E_{maj} and E_{min} are the first and second components as defined for the '*polarisation: major_minor*' above.

power

The first component is the amplitude of the field, $|\vec{E}|$, (i.e. the square root of the power) and the second component is the complex square root $\sqrt{E_{rhc}/E_{lhc}}$. The phase of the latter component is the rotation angle of the polarisation ellipse.

In the far field the square root of the power is determined from

$$|\vec{E}| = \sqrt{|E_{co}|^2 + |E_{cx}|^2}$$

and in the near field it is determined from all three field components: $|\vec{E}| = \sqrt{|E_{co}|^2 + |E_{cx}|^2 + |E_r|^2}$, E_r being the r -component of the field.

File Name (*file_name*) [file name].

Name of a file, to which the calculated field values shall be written.

File Format (*file_format*) [item from a list of character strings], default: **TICRA**.

The file format used to store the field data:

TICRA

The field is written in GRASP units and stored as an ASCII-file according to the TICRA-format described in *Field Data in Rectangular Grid*. The recommended file extension is *.grd*.

EDX

The field is written in SI units in a format according to the Electromagnetic Data Exchange (EDX) standard. The recommended file extension is `.grd`.

Files in this format cannot be read by the PostProcessor.

EDI

Obsolete file format name. The same as EDX.

Comment (*comment*) [character string].

A line of text which will be written as a header in the file specified by *file_name* above.

Frequency (*frequency*) [name of an object], default: **blank**.

Reference to a *Frequency* object, defining the frequencies for which the field is calculated. The reference must be to the same object as specified in the *frequency* attribute of the source generating the field. A *frequency* needs not to be specified. In that case, the frequencies in the *frequency* object of the source generating the field will be applied.

Command Types

The *Planar Grid* class is derived from the class *Field Storage*. See this class for available commands.

Remarks

The field points are defined in a regular grid in the two grid variables, *X* and *Y*. These are given by

$$X = X_s + \Delta X(i - 1)$$

$$Y = Y_s + \Delta Y(j - 1)$$

where *i* and *j* run through the values

$$i = 1, 2, \dots, N_x$$

$$j = 1, 2, \dots, N_y$$

X_s and *Y_s* are the *start* values and *N_x* and *N_y* are the number of values, *np*, of the attributes *x_range* and *y_range*, respectively, and ΔX and ΔY are the spacings in the grid

$$\Delta X = (X_e - X_s)/(N_x - 1)$$

$$\Delta Y = (Y_e - Y_s)/(N_y - 1)$$

where *X_e* and *Y_e* are the *end* values of the attributes *x_range* and *y_range*, respectively.

The plane on which the field is calculated is parallel to the *xy*-plane of the specified output coordinate system, *coor_sys*. The distance *z_d* from the origin to the plane is defined by the attribute *near_dist*.

If the `grid_type` attribute is 'xy' the grid points are given in the rectangular grid

$$\bar{r} = \hat{x}_0 x_0 + \hat{y}_0 y_0 + \hat{z}_0 z_d$$

of the output coordinate system $x_0 y_0 z_0$, and $(X, Y) = (x, y)$:

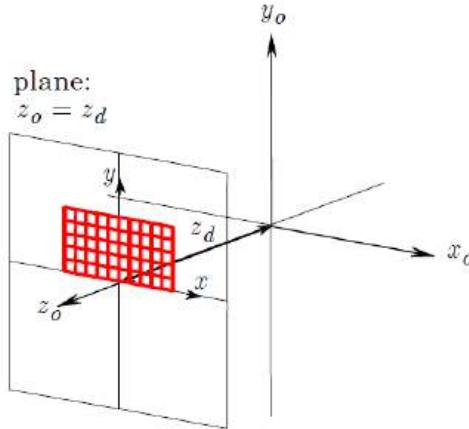


Figure 1

Planar output grid for `grid_type:xy`. The xy -grid is drawn for $-0.5 \text{ m} \leq x \leq 0.5 \text{ m}$, $0 \text{ m} \leq y \leq 0.5 \text{ m}$ in the plane $z_0 = z_d$. The grid spacings are 0.1 m in both x and y .

If the `grid_type` attribute is 'rho_phi' the grid points are given in the polar grid

$$\bar{r} = \hat{x}_0 \rho \cos \phi + \hat{y}_0 \rho \sin \phi + \hat{z}_0 z_d$$

and $(X, Y) = (\rho, \phi)$:

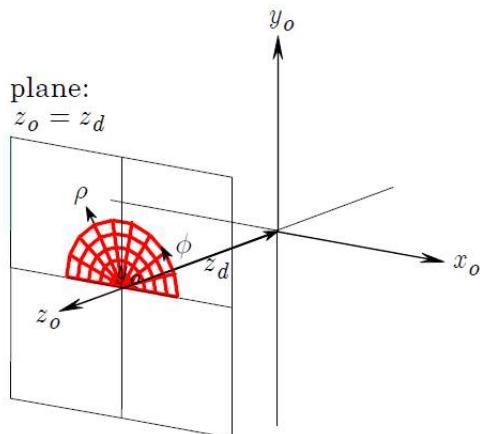


Figure 2

Planar output grid for `grid_type: rho_phi`. The $\rho\phi$ -grid is drawn for $0 \text{ m} \leq \rho \leq 0.5 \text{ m}$, $0^\circ \leq \phi \leq 180^\circ$ in the plane $z_0 = z_d$. The grid spacings are 0.1 m in ρ and 20° in ϕ .

CYLINDRICAL GRID (cylindrical_grid)

Purpose

The class *Cylindrical Grid* defines field points in a 2D grid on the surface of a circular cylinder where the field shall be calculated. The axis of the cylinder coincides with the z -axis of the coordinate system to which the output is referred. The cylinder surface is described by the parameters ϕ and z for constant value of ρ in a conventional circular cylindrical coordinate system. Thus, ϕ is the conventional azimuth angle measured in the xy -plane with the x -axis defining $\phi = 0$; ρ is the radius of the cylinder. Only near fields can be computed.

Links

Classes→*Electrical Objects*→*Field Storage*→*Grid*→*Cylindrical Grid*

Remarks

Syntax

```
<object name> cylindrical_grid
(
    coor_sys           : ref(<n>),
    radius             : <rl>,
    phi_range          : struct(start:<r>, end:<r>, np:<i>),
    z_range             : struct(start:<r>, end:<r>, np:<i>,
                                unit:<si>),
    truncation         : <si>,
    e_h                : <si>,
    polarisation       : <si>,
    file_name          : <f>,
    file_format        : <si>,
    comment            : <s>,
    frequency          : ref(<n>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<f> = file name
<si> = item from a list of character strings
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system (the output coordinate system) in which the field points and the calculated field vectors will be expressed.

Radius (*radius*) [real number with unit of length].

Defines the radius of the cylinder.

phi-Range (*phi_range*) [struct].

Defines the range of the azimuthal angle ϕ (see the remarks below).

Start (*start*) [real number].

Start value of the azimuthal coordinate ϕ , in degrees.

End (*end*) [real number].

End value of the azimuthal coordinate ϕ , in degrees.

Np (*np*) [integer].

Number of ϕ -values.

z-Range (*z_range*) [struct].

Defines the range of the axial coordinate z (see the remarks below).

Start (*start*) [real number].

Start value of the axial coordinate z .

End (*end*) [real number].

End value of the axial coordinate z .

Np (*np*) [integer].

Number of z -values.

Unit (*unit*) [item from a list of character strings], default: **m**.

Defines the unit of length in which the *z_range* is given (mm, cm, m, km, in, ft).

Truncation (*truncation*) [item from a list of character strings], default: **rect-angular**.

Must be rectangular or elliptical.

rectangular

The field is calculated in all of the grid points within the rectangular area defined by *phi_range* and *z_range* (in a plane rectangular ϕ z-coordinate system).

elliptical

The field is only calculated at the grid points inside the elliptical area with the axes of the ellipse defined by *phi_range* and *z_range* (in a plane rectangular ϕ z-coordinate system).

E/H-Field (*e_h*) [item from a list of character strings], default: **e_field**.

Specifies whether the complex *E*-field or *H*-field shall be calculated.

e_field

The complex E-field is calculated.

`h_field`

The complex H-field is calculated.

Polarisation (*polarisation*) [item from a list of character strings], default: **linear**.

Defines how the calculated field shall be decomposed in two field components parallel to the surface of the cylinder. Additionally, the ρ -component of the field is calculated as a third component.

`linear`

The linear components are the field components with the first component along ϕ and the second component along z . These components are E_{co} and E_{cx} , respectively, in the determination of the circular components.

`circular`

Circular components are calculated based on the linear components defined above. The first component is the right hand circular (E_{rhc}) and the second is the left hand circular component (E_{lhc}).

`major_minor`

The field is de-composed along the major and minor axes of the polarisation ellipse. The first field component is parallel to the major axis (E_{maj}) and the second to the minor axis (E_{min}).

`linear_xpd`

The ratios E_{co}/E_{cx} and E_{cx}/E_{co} , where E_{co} and E_{cx} are the first and second components as defined for the '*polarisation: linear*' above. This is the linear cross-polar discrimination ratio.

`circular_xpd`

The ratios E_{rhc}/E_{lhc} and E_{lhc}/E_{rhc} , where E_{rhc} and E_{lhc} are the first and second components as defined for the '*polarisation: circular*' above. This is the circular cross-polar discrimination ratio.

`major_minor_xpd`

The ratios E_{maj}/E_{min} and E_{min}/E_{maj} , where E_{maj} and E_{min} are the first and second components as defined for the '*polarisation: major_minor*' above.

power

The first component is the amplitude of the field, $|\vec{E}|$, (i.e. the square root of the power) and the second component is the complex square root $\sqrt{E_{rhc}/E_{lhc}}$. The phase of the latter component is the rotation angle of the polarisation ellipse.

In the far field the square root of the power is determined from

$$|\vec{E}| = \sqrt{|E_{co}|^2 + |E_{cx}|^2}$$

and in the near field it is determined from all three field components: $|\vec{E}| = \sqrt{|E_{co}|^2 + |E_{cx}|^2 + |E_r|^2}$, E_r being the r -component of the field.

File Name (*file_name*) [file name].

Name of a file, to which the calculated field values shall be written.

File Format (*file_format*) [item from a list of character strings], default: **TICRA**.

The file format used to store the field data:

TICRA

The field is written in GRASP units and stored as an ASCII-file according to the TICRA-format described in [Field Data in Rectangular Grid](#). The recommended file extension is *.grd*.

EDX

The field is written in SI units in a format according to the Electromagnetic Data Exchange (EDX) standard. The recommended file extension is *.grd*.

Files in this format cannot be read by the PostProcessor.

EDI

Obsolete file format name. The same as EDX.

Comment (*comment*) [character string].

A line of text which will be written as a header in the file specified by *file_name* above.

Frequency (*frequency*) [name of an object], default: **blank**.

Reference to a [Frequency](#) object, defining the frequencies for which the field is calculated. The reference must be to the same object as specified in the *frequency* attribute of the source generating the field. A *frequency* needs not to be specified. In that case, the frequencies in the *frequency* object of the source generating the field will be applied.

Command Types

The [Cylindrical Grid](#) class is derived from the class [Field Storage](#). See this class for available commands.

Remarks

The cylindrical surface is defined in the usual cylindrical coordinates (ρ, ϕ, z) by the points

$$\vec{r}(\phi, z) = \hat{x}R \cos \phi + \hat{y}R \sin \phi + \hat{z}z$$

where $\rho = R$ is the fixed *radius* of the cylinder.

The attributes *phi_range* and *z_range* define the ranges for the ϕ - and z -values.

The grid points (ϕ_i, z_i) are given by

$$\begin{aligned}\phi_i &= \phi_{start} + \Delta\phi \cdot (i - 1) \\ z_j &= z_{start} + \Delta z \cdot (j - 1)\end{aligned}$$

where i and j run through the values

$$\begin{aligned}i &= 1, 2, \dots, n_\phi \\ j &= 1, 2, \dots, n_z\end{aligned}$$

ϕ_{start} and z_{start} are the *start* values and n_ϕ and n_z are the number of values, *np*, of the attributes *phi_range* and *z_range*, respectively, and $\Delta\phi$ and Δz are the spacings in the grid

$$\begin{aligned}\Delta\phi &= (\phi_{end} - \phi_{start}) / (n_\phi - 1) \\ \Delta z &= (z_{end} - z_{start}) / (n_z - 1)\end{aligned}$$

where ϕ_{end} and z_{end} are the *end* values of the attributes *phi_range* and *z_range*, respectively.

An example of a cylindrical grid is shown in Figure 1. In the figure, $x_0y_0z_0$ is the output coordinate system as defined by *coor_sys*.

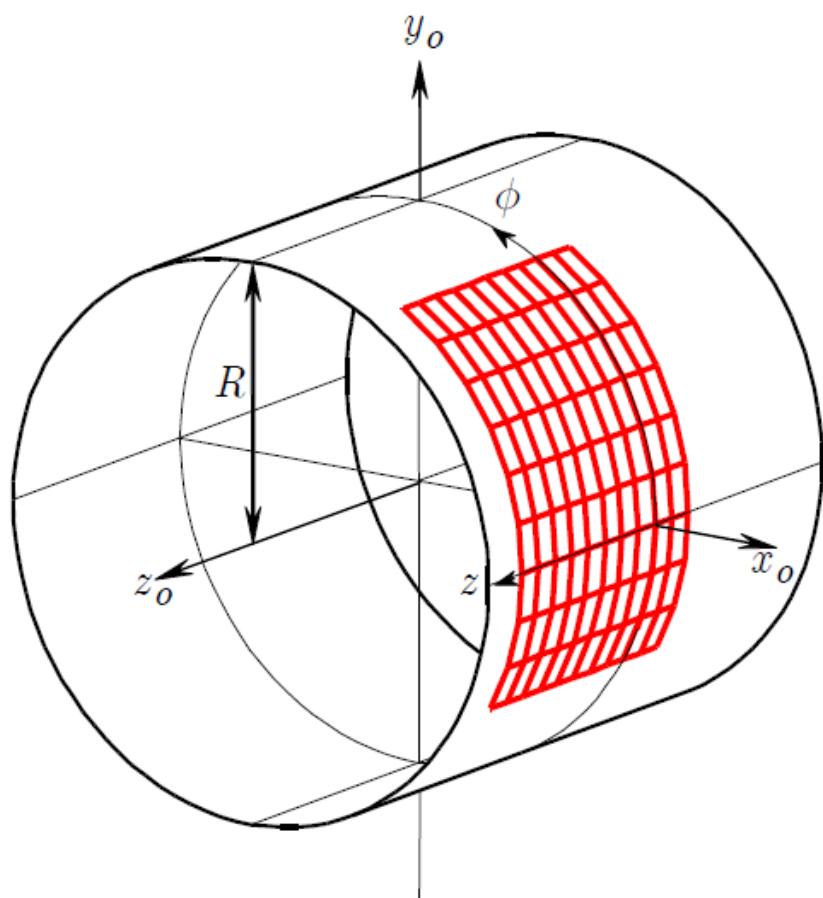


Figure 1

A cylindrical grid on a cylinder with radius $R = 2 \text{ m}$ drawn for $-30^\circ \leq \phi \leq 60^\circ$ and $-0.4 \text{ m} \leq z \leq 1.6 \text{ m}$ with a spacing of 10° in ϕ and 0.2 m in z .

SURFACE GRID (surface_grid)

Purpose

The class **Surface Grid** defines field points in a rectangular grid on a **Reflector** or a **Plate** (**Triangular Plate**, **Parallelogram** and **Rectangular Plate**) where the field shall be calculated. This class is useful for calculating an aperture field or displaying the PO current distribution on a reflector or a **Plate**.

Links

[Classes](#)→[Electrical Objects](#)→[Field Storage](#)→[Grid](#)→[Surface Grid](#)

Remarks

Syntax

```
<object name> surface_grid
(
    scatterer : ref(<n>),
    x_range   : struct(start:<r>, end:<r>, np:<i>,
                       unit:<si>),
    y_range   : struct(start:<r>, end:<r>, np:<i>),
    field_type : <si>,
    polarisation : <si>,
    polarisation_and_phase_coor_sys: ref(<n>),
    phase_adjustment : <si>,
    file_name   : <f>,
    comment     : <s>,
    frequency   : ref(<n>)
)
where
<i> = integer
<n> = name of an object
<r> = real number
<s> = character string
<f> = file name
<si> = item from a list of character strings
```

Attributes

Scatterer (*scatterer*) [name of an object].

Reference to a **Reflector** object or a **Plate** object (**Triangular Plate**, **Parallelogram** and **Rectangular Plate**) on which the surface field is calculated.

x-Range (*x_range*) [struct].

Defines the range of the local variable *x* in which the surface of the Scatterer is defined. The local coordinate system is defined for the different scatterers in the manual.

Start (*start*) [real number].

Start value of the coordinate *x*.

End (*end*) [real number].

End value of the coordinate *x*.

Np (*np*) [integer].

Number of *x*-values.

Unit (*unit*) [item from a list of character strings], default: **m**.

Defines the unit of length applied in x-Range as well in y-Range (mm, cm, m, km, in, ft).

y-Range (*y_range*) [struct].

Defines the range of the local variable *y* in which the surface of the Scatterer is defined. The start and end values are given in the unit specified by the member *unit* of the attribute x-Range. The local coordinate system is defined for the different scatterers in the manual.

Start (*start*) [real number].

Start value of the coordinate *y*.

End (*end*) [real number].

End value of the coordinate *y*.

Np (*np*) [integer].

Number of *y*-values.

Field Type (*field_type*) [item from a list of character strings], default: **incident_e_field**.

Different field types may be determined. The field vector is the normal vector to the front surface of the scatterer.

`incident_e_field`

The incident E-field on the surface is determined.

`incident_h_field`

The incident H-field on the surface is determined.

`reflected_e_field`

The reflected E-field, \bar{E}_r , on the surface is determined according to the formula $\bar{E}_r = 2\hat{n}(\hat{n} \cdot \bar{E}_i) - \bar{E}_i$, where \bar{E}_i is the incident E-field.

`reflected_h_field`

The reflected H-field, \bar{H}_r , on the surface is determined according to the formula $\bar{H}_r = \bar{H}_i - 2\hat{n}(\hat{n} \cdot \bar{H}_i)$, where \bar{H}_i is the incident H-field.

`currents`

The PO currents are determined as $2\hat{n} \times \bar{H}_i$ where \bar{H}_i is the incident H-field.

Polarisation (*polarisation*) [item from a list of character strings], default: **linear**.

Defines how the calculated field shall be decomposed in two field components parallel to the plane. Additionally, the z -component of the field is calculated as a third component.

linear

The linear components are the field components with the first component along x and the second component along y of the *polarisation_and_phase_coor_sys*. These field components are E_{co} and E_{cx} , respectively, in the determination of the circular components.

circular

Circular components are calculated based on the linear components, E_{co} and E_{cx} , defined above. The first component is right hand circular (E_{rhc}) and the second is left hand circular component (E_{lhc}).

rho_phi

The field is decomposed along the ρ - and ϕ -unit vectors of the *polarisation_and_phase_coor_sys*. The ρ -component (E_ρ) is the first field component and the ϕ -component is the second (E_ϕ).

major_minor

The field is decomposed along the major and minor axes of the polarisation ellipse. The first field component is parallel to the major axis (E_{maj}) and the second to the minor axis (E_{min}).

linear_xpd

The ratios E_{co}/E_{cx} and E_{cx}/E_{co} , where E_{co} and E_{cx} are the first and second components as defined for the '*polarisation*: linear' above. This is the linear cross-polar discrimination ratio.

circular_xpd

The ratios E_{rhc}/E_{lhc} and E_{lhc}/E_{rhc} , where E_{rhc} and E_{lhc} are the first and second components as defined for the '*polarisation*: circular'. This is the circular cross-polar discrimination ratio.

rho_phi_xpd

The ratios E_ρ/E_ϕ and E_ϕ/E_ρ , where E_ρ and E_ϕ are the first and second components as defined for the '*polarisation*: rho_phi' above.

major_minor_xpd

The ratios E_{maj}/E_{min} and E_{min}/E_{maj} , where E_{maj} and E_{min} are the first and second components as defined for the '*polarisation*: major_minor' above.

power

The first component is the amplitude of the field,

$$|\vec{E}| = \sqrt{|E_{co}|^2 + |E_{cx}|^2 + |E_z|^2} \quad (1)$$

E_z being the z -component of the field in the *polarisation_and_phase_coor_sys* (i.e. it is the square root of the power of all three field components) and the second component is the complex square root $\sqrt{E_{rhc}/E_{lhc}}$. The phase of the latter component is the rotation angle of the polarisation ellipse.

Polarisation and Phase Coor Sys (*polarisation_and_phase_coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate System*. The calculated field vectors on the surface will be resolved in components according to the axes of this coordinate system. The x - and y -components of the field may be converted according to the specified polarisation as described below whereas the z -component is unchanged. If a phase adjustment is required the phase of the field will be modified as described in the remarks below.

Phase Adjustment (*phase_adjustment*) [item from a list of character strings], default: **off**.

A phase adjustment of the calculated field is possible.

off

No phase adjustment is performed.

on

The phase of the field on the surface is adjusted as explained in the remarks below.

File Name (*file_name*) [file name].

Name of a file, the field file, to which the calculated field values shall be written. The content of the file is described in *Field Data in Rectangular Grid*. The recommended file extension is *.grd*.

Comment (*comment*) [character string].

A line of text which will be written as a header in the file specified by *file_name* above.

Frequency (*frequency*) [name of an object], default: **blank**.

Reference to a *Frequency* object, defining the frequencies for which the field is calculated. The reference must be to the same object as specified in the *frequency* attribute of the source generating the field. A *frequency* needs not to be specified. In that case, the frequencies in the *frequency* object of the source generating the field will be applied.

Command Types

The *Surface Grid* class is derived from the class *Field Storage*. See this class for available commands.

Remarks

For a *Reflector*, the surface grid is defined in the coordinates (x, y) in the xy -plane the coordinate system of the *Reflector*. The reflector surface may be described by

$$\bar{r}(x, y) = \hat{x}x + \hat{y}y + \hat{z}z(x, y)$$

where $z(x, y)$ defines the *Surface* of the reflector. For a *Plate*, the surface grid is defined correspondingly in the local coordinates (x, y) in the plane of the *Plate*, see the individual sub-classes of *Plate*.

The field points are defined in a regular grid in the two grid variables, x and y . These are given by

$$\begin{aligned} x &= x_s + \Delta x \cdot (i - 1), i = 1, 2, \dots, n_x \\ y &= y_s + \Delta y \cdot (j - 1), j = 1, 2, \dots, n_y \end{aligned}$$

with Δx and Δy being the spacings in the grid,

$$\begin{aligned} \Delta x &= (x_e - x_s)/(n_x - 1) \\ \Delta y &= (y_e - y_s)/(n_y - 1) \end{aligned}$$

x_s and x_e are the members *start* and *end*, respectively, of the attribute *x_range*, and n_x is the member *np* of the same attribute. Correspondingly, y_s and y_e are the members *start* and *end*, respectively, of the attribute *y_range*, and n_y is the member *np* of the same attribute.

Phase adjustment

If *phase_adjustment* is set to 'off', the field is computed at the surface point R , cf. the figure below.

If *phase_adjustment* is set to 'on', the computed field on the surface at point R is multiplied by e^{-jkd} , where k is the wavenumber and d is the distance from R to the xy -plane of the *polarisation_and_phase_coor_sys*, which is denoted by the subscript pp in the figure.

Setting *phase_adjustment* to 'on' means that the phase of the calculated field is the one it would have on the xy -plane of the *polarisation_and_phase_coor_sys* if the field propagated along parallel rays in direction z_{pp} .

The distance d becomes negative if the plane is located behind the reflector.

In the figure, an offset paraboloidal reflector with focus at F is illuminated by a feed, which is positioned below this focus. Rays from the feed, reflected in the reflector, are shown.

When the *phase_adjustment* is 'off', the field is determined at points on the reflector, the field components being parallel to x_{pp} and y_{pp} , which are x and y of the *polarisation_and_phase_coor_sys*. The field amplitudes will illustrate the field intensity, but the phase will vary drastically as the ray path length for the different rays to the reflector varies.

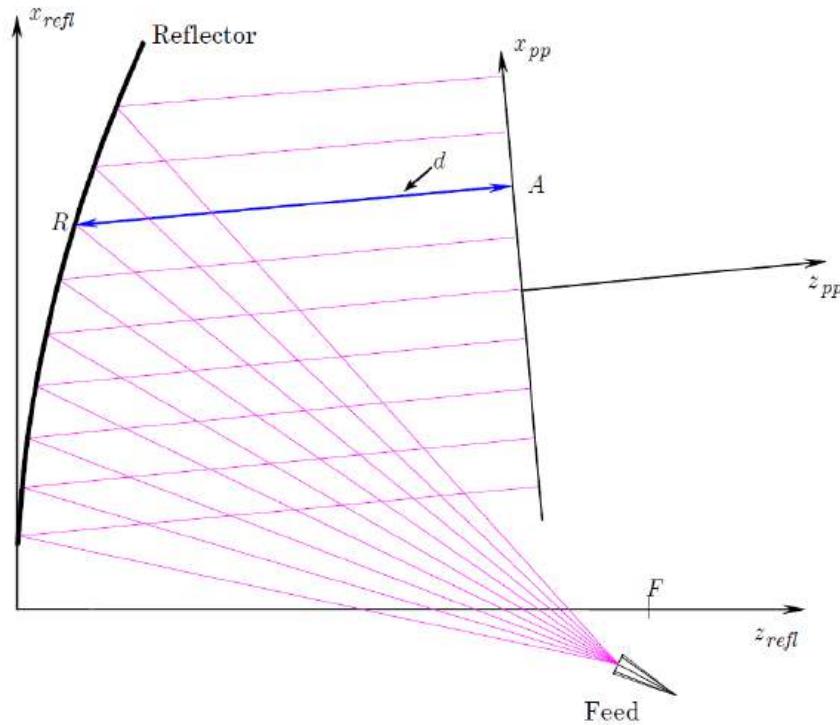


Figure 1 Example illustrating the *phase_adjustment*.

When the *phase_adjustment* is 'on', the field is determined at the same points but phase-corrected by the distance d to the xy -plane of the *polarisation_and_phase_coor_sys*, the coordinates of which are denoted x_{pp} , y_{pp} and z_{pp} . When this coordinate system is tilted to having the z_{pp} -axis parallel to the reflected rays, the corrected phase is nearly constant as all ray paths from the feed via the reflector to the x_{pp} y_{pp} -plane are nearly of the same lengths. As the reflector is defocused, the phase will not be precisely uniform, but the deviations may easily be evaluated in this way.

The field components are again parallel to x_{pp} and y_{pp} , x and y of the *polarisation_and_phase_coor_sys*. The third component in the file is the component parallel to z_{pp} .

TABULATED UV-POINTS (tabulated_uv_points)

Purpose

The class *Tabulated uv-Points* defines a set of arbitrarily distributed far-field points at which the field is to be calculated.

Links

Classes→*Electrical Objects*→*Field Storage*→*Tabulated uv-Points*

Remarks

Syntax

```
<object name> tabulated_uv_points
(
    coor_sys           : ref(<n>),
    stat_name          : <f>,
    file_name          : <f>,
    frequency          : ref(<n>)
)
where
<n> = name of an object
<f> = file name
```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate System* defining the coordinate system (the output coordinate system) in which the field points and the calculated field vectors will be expressed.

Stat Name (*stat_name*) [file name].

Name of an input file in which the positions of the field points are given. The content of the file is described in Section *Field Directions*.

File Name (*file_name*) [file name].

Name of a file, the field file, to which the calculated field values shall be written. The content of the file is described in Section *Field Data in Tabulated Directions*. The recommended file extension is *.fmt*.

Frequency (*frequency*) [name of an object], default: **blank**.

Reference to a *Frequency* object, defining the frequencies for which the field is calculated. The reference must be to the same object as specified in the *frequency* attribute of the source generating the field. A *frequency* needs not to be specified. In that case, the frequencies in the *frequency* object of the source generating the field will be applied.

Command Types

The *Tabulated uv-Points* class is derived from the class *Field Storage*. See this class for available commands.

Remarks

Only far fields can be computed. The field points are defined in the file specified by *stat_name* and must be given in *uv*-coordinates

$$\begin{aligned} u &= \sin \theta \cos \phi \\ v &= \sin \theta \sin \phi \end{aligned}$$

where θ and ϕ are the usual spherical polar and azimuthal angles, respectively, of the output coordinate system. The direction to the field point \hat{r} specified by (u,v) then becomes

$$\hat{r} = (u, v, \sqrt{1 - u^2 - v^2})$$

Objects of this class are typically applied to determine the field from a transmitting antenna on receiving stations specified in the file given by *stat_name*.

ELECTRICAL PROPERTIES

Purpose

Electrical Properties is a menu with classes, which define various electrical properties of a scatterer.

In general, the electrical properties of a scatterer is described by the parameters for the material of the scatterer or, more precisely, for the material layered on the *Surface* of the *Scatterer*. The surface layer with the electrical properties may be displaced relative to the *Surface* specified for the *Scatterer*. This is useful when several layers with *Electrical Properties* are modelled.

If the material is penetrable for the field then also a thickness of the layer of the material shall be specified, otherwise the material can be considered as an infinitely thin layer.

The following classes of *Electrical Properties* are available:

- Ideal Grid*
- Strip Grid*
- Strip Grid in Dielectric Layer*
- Wire Grid*
- Dielectric Layer*
- Wire Mesh*
- Perfect Conductivity*
- Perfect Absorption*
- Power Splitting*
- Finite Conductivity*
- Tabulated Electrical Properties*

Links

Classes→*Electrical Objects*→*Electrical Properties*

IDEAL GRID (ideal_grid)

Purpose

The class ***Ideal Grid*** is used to define an ideal polarisation sensitive grid. A surface made from this surface material would reflect all waves polarised parallel to the direction of the grid, whereas it would transmit all orthogonally polarised waves.

Links

[Classes](#)→[Electrical Objects](#)→[Electrical Properties](#)→***Ideal Grid***

Remarks

Syntax

```
<object name> ideal_grid
(
    displacement : <rl>,
    ref_coor_sys   : ref(<n>),
    ref_angle      : <r>,
    plot_lines     : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
```

Attributes

Displacement (*displacement*) [real number with unit of length], default: **0**.

Defines where the surface with the ideal grid is located relative to the specified surface of the scatterer (see the remarks below). The displacement is positive in direction of the positive surface normal of the ***Scatterer***.

Reference Coordinate System (*ref_coor_sys*) [name of an object].

Reference to an object of one of the ***Coordinate Systems*** classes. The grid lines on the surface will be parallel when projected on the $x_c y_c$ -plane of this $x_c y_c z_c$ -coordinate system. A reference must be given, as there is no default value for this attribute.

Reference Angle (*ref_angle*) [real number], default: **0**.

Rotation (in degrees) of the grid direction from the x_c -axis around the z_c -axis of the reference coordinate system.

Plot Lines (*plot_lines*) [integer], default: **11**.

Number of lines (approximately) drawn in a plot of the scatterer having this grid as electrical property.

Remarks

The layer with the ideal grid is assumed to be infinitely thin.

Note, that when an *Ideal Grid* is specified for a scatterer then a conducting surface - if such one shall be included in the model - must be specified as a separate layer (e.g. of class *Perfect Conductivity*) in the specification of the electrical properties of the scatterer.

The following Figure 1 illustrates an *Ideal Grid* on the surface of a scatterer. The Displacement is specified to 0.0 mm. The next Figure 2 illustrates a *Reflector* with an *Ideal Grid* positioned 20 mm in front of it. Thus, the Displacement is specified to 20.0 mm. Further, the conducting surface of the *Reflector* must be included by specifying a layer having *Perfect Conductivity*.

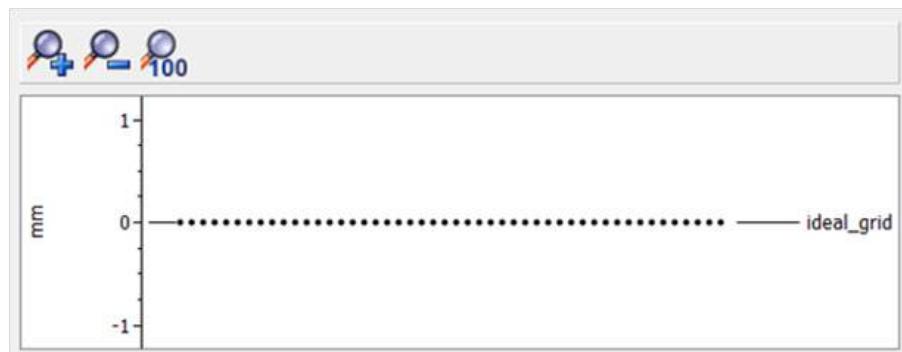


Figure 1 Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP. An *Ideal Grid* is placed at the surface of the scatterer, thus the Displacement is zero.



Figure 2 Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with an *Ideal Grid* positioned 20 mm above a *Reflector*. The Displacement for the grid is thus specified to 20 mm. The reflector itself is specified as a layer of *Perfect Conductivity*.

As seen in the above figures, the *Ideal Grid* has a signature given by a series of dense dots in the illustration of the specified Electrical Properties.

If the *Ideal Grid* is placed on a dielectric layer, which in turn is placed on the specified surface of the scatterer, it is necessary to define where the grid surface is located relative to the specified surface. In such case, the

displacement of the reference plane along the positive surface normal shall be equal to the thickness of the dielectric. The dielectric layer is defined in class *Dielectric Layer*.

The grid is deposited on the curved reflector as a projection from some direction, e.g. the boresight. Seen from this direction the strips will be parallel on an aperture plane. The direction defines the z_c -axis, and the x_c - and y_c -axis defines the aperture plane, where the strips by default are parallel to the x_c -axis, see Figure 3. It is, however, possible to define a rotation of the strip direction in the x_cy_c -plane. The angle of rotation, Reference Angle (α in Figure 3), is positive from the x_c -axis towards the y_c -axis.

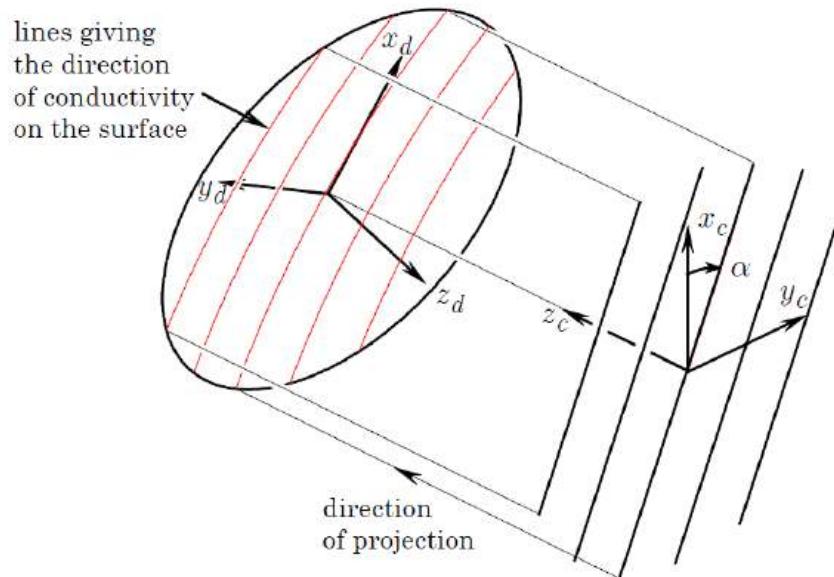


Figure 3

The reference coordinate system, $x_cy_cz_c$, specifying the local orientation of the material definition coordinate system, $x_dy_dz_d$, on the reflector surface. The lines in the x_cy_c -plane are parallel and rotated the angle α with respect to the x_c -axis. These lines are projected in the direction of z_c on the reflector surface.

A linearly polarised plane wave incoming along the direction of the z_c -axis will have polarisation parallel to the grid, and will be reflected. A plane wave polarised perpendicular to the first will propagate through the grid.

The orientation of the *Ideal Grid* can be illustrated on a plot of the scatterer. The number of grid lines drawn on the scatterer are specified by Plot Lines. Because of the way the lines are generated, the actual numbers of lines drawn may be slightly smaller than Plot Lines for some scatterers.

The grid lines will be drawn on the surface of the scatterer and the value of Displacement is thus not illustrated in the plot.

In order to see the grid clearly in the 3D-view the surface lines on the scatterer may be hidden (by right-clicking the 3D view canvas).

STRIP GRID (strip_grid)

Purpose

The class *Strip Grid* defines the electrical properties of a surface by means of a strip grid, i.e. a regular grid of conducting parallel strips. This would typically be used to model a polarisation sensitive reflector antenna. If the grid is placed on a dielectric layer, the more accurate *Strip Grid in Dielectric Layer* should be used.

Links

[Classes](#)→[Electrical Objects](#)→[Electrical Properties](#)→[Strip Grid](#)

Remarks

Syntax

```
<object name> strip_grid
(
    displacement          : <rl>,
    ref_coor_sys          : ref(<n>),
    ref_angle              : <r>,
    spacing                : <rl>,
    width                  : <rl>,
    plot_lines             : <i>
)
```

where

<i> = integer

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

Attributes

Displacement (*displacement*) [real number with unit of length], default: **0**.

Defines where the surface with the strip grid is located relative to the specified surface of the scatterer (see the remarks below). The displacement is positive in direction of the positive surface normal of the *Scatterer*.

Reference Coordinate System (*ref_coor_sys*) [name of an object].

Reference to an object of one of the *Coordinate System* classes. The strips on the surface will be parallel when projected on the $x_c y_c$ -plane of this $x_c y_c z_c$ -coordinate system. A reference must be given, as there is no default value for this attribute.

Reference Angle (*ref_angle*) [real number], default: **0**.

Rotation (in degrees) of the grid direction from the x_c -axis around the z_c -axis of the reference coordinate system.

Spacing (*spacing*) [real number with unit of length].

Spacing of the strips in the grid, measured in the xy -plane of the reference coordinate system as the distance from the centre line of one strip to the centre line of the next strip.

Width (*width*) [real number with unit of length].

The width of the strips in the grid.

Plot Lines (*plot_lines*) [integer], default: **11**.

Number of lines (approximately) drawn in a plot of the scatterer having this grid as electrical property.

Remarks

The layer with the strip grid is assumed to be infinitely thin.

Note, that when a *Strip Grid* is specified for a scatterer then a conducting surface - if such one shall be included in the model - must be specified as a separate layer (e.g. of class *Perfect Conductivity*) in the specification of the electrical properties of the scatterer.

The following Figure 1 illustrates a *Strip Grid* placed at the surface of the scatterer. The Displacement is specified to 0.0 mm. The next Figure 2 illustrates a *Reflector* with a *Strip Grid* positioned 20 mm in front of it. Thus, the Displacement is specified to 20.0 mm. Further, the conducting surface of the *Reflector* must be included by specifying a layer having *Perfect Conductivity*.

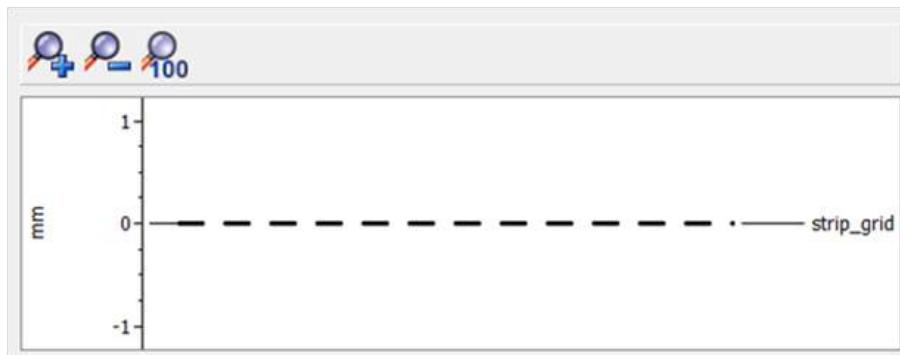


Figure 1

Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with a *Strip Grid* located at the surface of the scatterer. Therefore, the Displacement is set to zero.

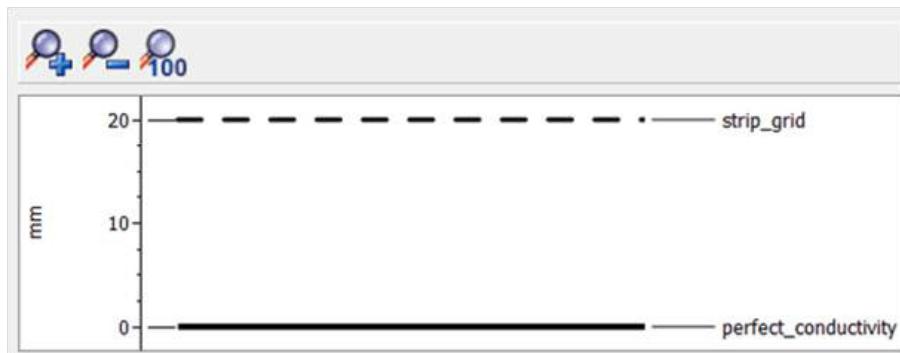


Figure 2 Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with a *Strip Grid* positioned 20 mm above a *Reflector*. The Displacement for the grid is thus specified to 20 mm. The reflector itself is specified as a layer of *Perfect Conductivity*.

As seen in the above figures, the *Strip Grid* has a signature given by a series of dashes in the illustration of the specified Electrical Properties. The values of the Spacing and the Width of the strips are not reflected in the illustrations.

If the *Strip Grid* is placed on a dielectric layer, it is recommended that the user specifies the electrical properties using the class *Strip Grid in Dielectric Layer*, which is more accurate.

When the strip grid is deposited on a curved reflector it is constructed as a projection from some direction, e.g. the boresight. Seen from this direction the strips will be parallel and equispaced on an aperture plane. It is the z_c -axis of the Reference Coordinate System which defines the direction of projection and the x_c - and y_c -axis which define the aperture plane where the strips by default are parallel to the x_c -axis, see Figure 3. It is, however, possible to define a rotation angle, the Reference Angle (α in Figure 3), of the strip direction with respect to the x_c -axis.

The orientation of the *Strip Grid* can be illustrated on a plot of the scatterer. The number of grid lines drawn on the scatterer are specified by Plot Lines. Because of the way the lines are generated, the actual numbers of lines drawn may be slightly smaller than Plot Lines for some scatterers.

The Width of the strips is not illustrated in the plots. Further, the grid lines will be drawn on the surface of the scatterer and the value of Displacement is neither illustrated in the plot.

In order to see the grid clearly in the 3D-view the surface lines on the scatterer may be hidden. This is done by right-clicking the 3D view canvas, choosing 3D-VIEW SETTINGS and here to choose the actual scatterer. For this the number of lines on the surface shall be reduced to one or zero.

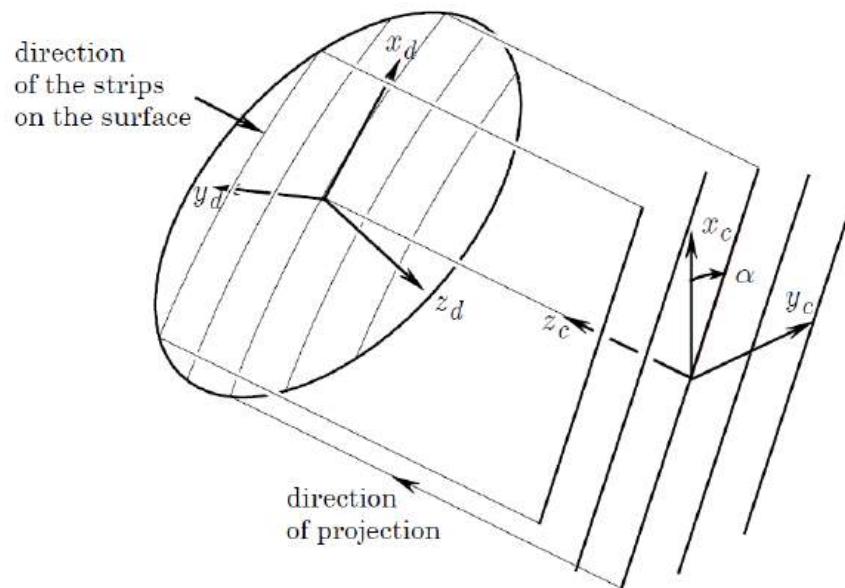


Figure 3

The reference coordinate system, $x_c y_c z_c$, specifies the orientation of the local material definition coordinate system, $x_d y_d z_d$, on the reflector surface. The projected strips are parallel and equispaced in the $x_c y_c$ -plane and rotated the angle α with respect to the x_c -axis. The direction of the projection is along the z_c -axis.

STRIP GRID IN DIELECTRIC LAYER (strip_grid_dielectric_layer)

Purpose

The class *Strip Grid in Dielectric Layer* defines the electrical properties of a strip grid, i.e. a regular grid of conducting parallel strips, located within a dielectric layer. This would typically be used to model a polarisation sensitive reflector antenna, e.g., a dual-gridded reflector. If the grid is not placed within (or on) a dielectric layer then the simpler modelled *Strip Grid* may be used instead.

Links

Classes→*Electrical Objects*→*Electrical Properties*→*Strip Grid in Dielectric Layer*

Remarks

Syntax

```
<object name> strip_grid_dielectric_layer
(
    displacement           : <rl>,
    strip_grid             : struct(ref_coor_sys:ref(<n>),
                                    ref_angle:<r>, spacing:<rl>,
                                    width:<rl>),
    substrate_above         : struct(thickness:<rl>,
                                    dielectric_constant:<r>,
                                    loss_tangent:<r>),
    substrate_below         : struct(thickness:<rl>,
                                    dielectric_constant:<r>,
                                    loss_tangent:<r>),
    plot_lines              : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
```

Attributes

Displacement (*displacement*) [real number with unit of length], default: **0**.

Defines where the surface of the strip grid (without dielectrics) is located relative to the specified surface of the scatterer (see the remarks below). The displacement is positive in direction of the positive surface normal of the *Scatterer*.

Strip Grid (*strip_grid*) [struct].

Specification of the strip grid in the dielectric layer

Reference Coordinate System (*ref_coor_sys*) [name of an object].

Reference to an object of one of the *Coordinate System* classes.
The strips on the surface will be parallel when projected on the x_cy_c -plane of this $x_cy_cz_c$ -coordinate system. A reference must be given, as there is no default value for this attribute.

Reference Angle (*ref_angle*) [real number], default: **0**.

Rotation (in degrees) of the grid direction from the x_c -axis around the z_c -axis of the reference coordinate system.

Spacing (*spacing*) [real number with unit of length].

Spacing of the strips in the grid, measured in the xy -plane of the reference coordinate system as the distance from the centre line of one strip to the centre line of the next strip.

Width (*width*) [real number with unit of length].

The width of the strips in the grid.

Substrate Above Strip Grid (*substrate_above*) [struct].

The dielectric substrate layer above the strip grid is that at the positive side (with respect to the surface normal) of the surface of the strip grid.

Thickness (*thickness*) [real number with unit of length].

Thickness of the dielectric layer above the strip grid, must be positive or zero. If there is no layer above the strip grid, the thickness shall be specified to be zero.

Dielectric Constant (*dielectric_constant*) [real number], default: **1**.

Relative dielectric constant of the layer above the strip grid.

Loss Tangent (*loss_tangent*) [real number], default: **0**.

Loss tangent of the dielectric substrate above the strip grid.

Substrate Below Strip Grid (*substrate_below*) [struct].

The dielectric substrate layer below the strip grid is that at the negative side (with respect to the surface normal) of the surface of the strip grid.

Thickness (*thickness*) [real number with unit of length].

Thickness of the dielectric layer below the strip grid, must be positive or zero. If there is no layer below the strip grid, the thickness shall be specified to be zero.

Dielectric Constant (*dielectric_constant*) [real number], default: **1**.

Relative dielectric constant of the layer below the strip grid.

Loss Tangent (*loss_tangent*) [real number], default: **0**.

Loss tangent of the dielectric substrate below the strip grid.

Plot Lines (*plot_lines*) [integer], default: **11**.

Number of lines (approximately) drawn in a plot of the scatterer having this grid as electrical property.

Remarks

Note, that when a *Strip Grid in Dielectric Layer* is specified for a scatterer then a conducting surface - if such one shall be included in the model - must be specified as a separate layer (e.g. of class *Perfect Conductivity*) in the specification of the electrical properties of the scatterer.

The *Strip Grid in Dielectric Layer* describes a strip grid that is placed within or at the surface of a dielectric layer. The dielectric constant of the substrate above the strip grid can be different from that of the substrate below the strip grid. It is possible to have only one substrate layer above or below the strip grid. In this case, the Thickness of the non-existing substrate shall be set to zero. If the Thickness of both substrates are zero, a *Strip Grid in Dielectric Layer* is identical to a *Strip Grid*.

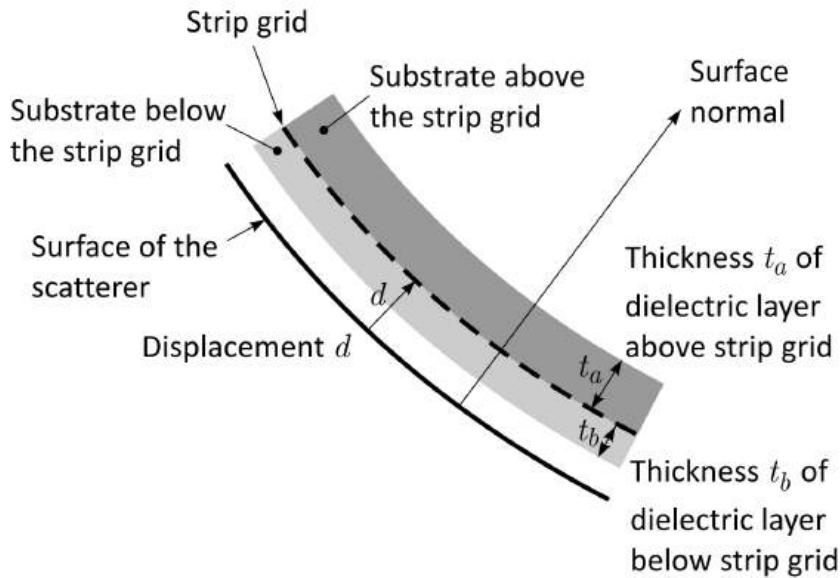


Figure 1

The *Strip Grid in Dielectric Layer*. The grid is offset with the Displacement d from the surface of the actual scatterer measured along its surface normal. The dielectric substrate above the scatterer has the thickness t_a and the dielectric substrate below the scatterer has the thickness t_b .

The strip grid itself is assumed to be infinitely thin, and the specified Displacement is the distance from the surface of the *Scatterer* to the surface of the strip grid, positive along the surface normal to the *Scatterer*, cf. Figure 1. This means that the strip grid - when Displacement is specified to zero - is located on the surface of the scatterer and that the substrate below the strip grid is behind this surface, see Figure 2.

If the *Strip Grid in Dielectric Layer* is placed atop another dielectric layer, which in turn is placed on the specified surface of the scatterer, see Figure 3, the displacement of the strip grid along the positive surface normal shall be equal to the thickness of the two dielectric layers below the strip grid. The lowest positioned dielectric layer may be defined in class *Dielectric Layer*.

When the strip grid is deposited on a curved reflector it is constructed as a projection from some direction, e.g. the boresight. Seen from this direction

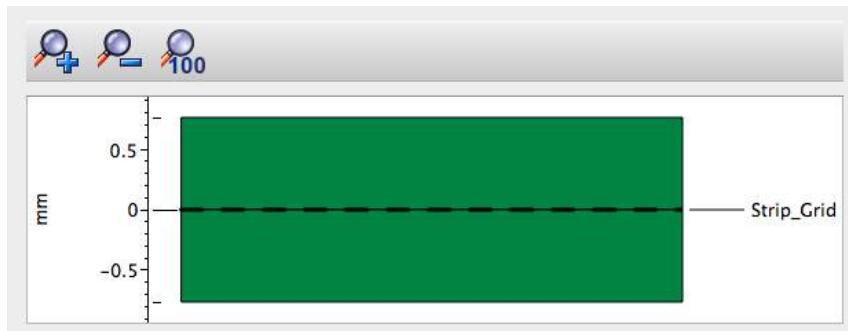


Figure 2

Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP. Here, a *Strip Grid in Dielectric Layer* with Displacement specified to 0 mm and with substrate both below and above the strip grid, both substrates have thickness 0.762 mm. The surface of the scatterer is defining the zero at the axis.

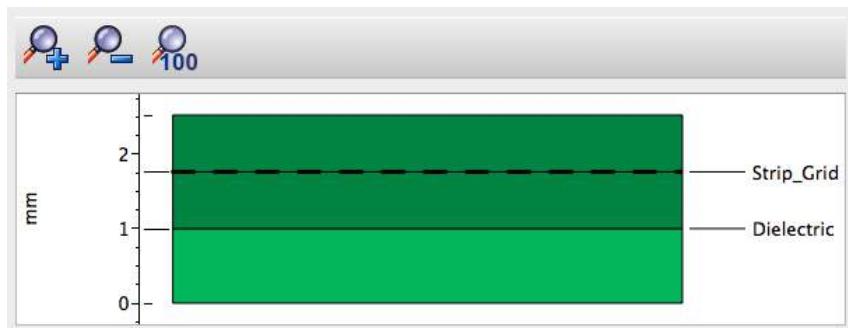


Figure 3

Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP. Here, a *Strip Grid in Dielectric Layer* and a *Dielectric Layer* have been specified. The thickness and Displacement of the *Dielectric Layer* is 1 mm. The thickness of the substrate below the strip grid in *Strip Grid in Dielectric Layer* is 0.762 mm, thus the Displacement for *Strip Grid in Dielectric Layer* needs to be 1.762 mm.

the strips will be parallel and equispaced on an aperture plane. It is the z_c -axis of the Reference Coordinate System which defines the direction of projection and the x_c - and y_c -axis which define the aperture plane where the strips by default are parallel to the x_c -axis, see Figure 4. It is, however, possible to define a rotation angle, Reference Angle in the attribute Strip Grid (α in Figure 4), of the strip direction with respect to the x_c -axis.

The orientation of the *Strip Grid in Dielectric Layer* can be illustrated on a plot of the scatterer. The number of grid lines drawn on the scatterer are specified by Plot Lines. Because of the way the lines are generated, the actual numbers of lines drawn may be slightly smaller than Plot Lines for some scatterers.

The Width of the strips is not illustrated in the plots. Further, the grid lines will be drawn on the surface of the scatterer and the value of Displacement is neither illustrated in the plot.

In order to see the grid clearly in the 3D-view the surface lines on the scatterer may be hidden. This is done by right-clicking the 3D view canvas, choosing 3D-VIEW SETTINGS and here to choose the actual scatterer. For this the number of lines on the surface shall be reduced to one or zero.

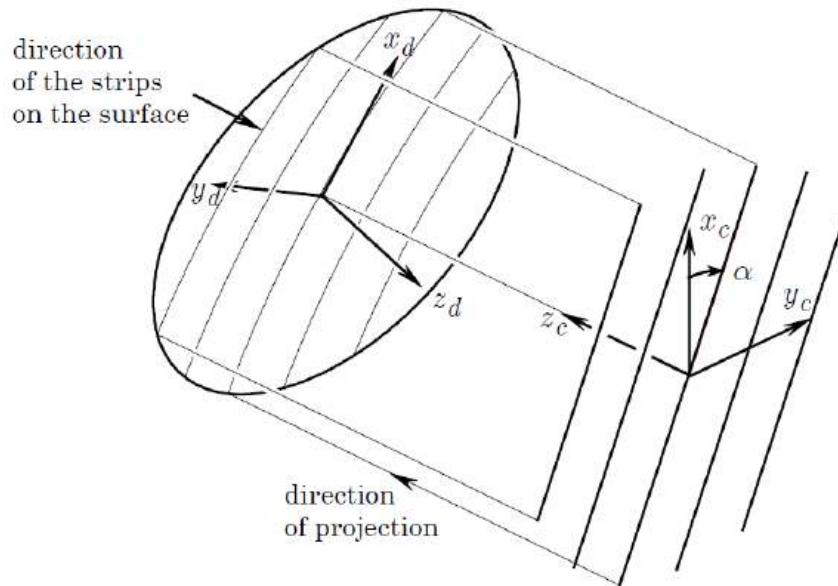


Figure 4

The reference coordinate system, $x_c y_c z_c$, specifies the orientation of the local material definition coordinate system, $x_d y_d z_d$, on the reflector surface. The projected strips are parallel and equispaced in the $x_c y_c$ -plane and rotated the angle α with respect to the x_c -axis. The direction of the projection is along the z_c -axis.

WIRE GRID (wire_grid)

Purpose

The class **Wire Grid** defines the electrical properties of a surface by means of a wire grid, i.e. a regular grid of conducting parallel wires. This would typically be used to model a polarisation sensitive reflector antenna.

Links

Classes→*Electrical Objects*→*Electrical Properties*→**Wire Grid**

Remarks

Syntax

```
<object name> wire_grid
(
    displacement          : <rl>,
    ref_coor_sys          : ref(<n>),
    ref_angle              : <r>,
    spacing                : <rl>,
    diameter               : <rl>,
    plot_lines             : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
```

Attributes

Displacement (*displacement*) [real number with unit of length], default: **0**.

Defines where the surface with the wire grid is located relative to the specified surface of the scatterer (see the remarks below). The displacement is positive in direction of the positive surface normal of the **Scatterer**.

Reference Coordinate System (*ref_coor_sys*) [name of an object].

Reference to an object of one of the **Coordinate System** classes. The wires on the surface will be parallel when projected on the $x_c y_c$ -plane of this $x_c y_c z_c$ -coordinate system. A reference must be given, as there is no default value for this attribute.

Reference Angle (*ref_angle*) [real number], default: **0**.

Rotation (in degrees) of the wire direction from the x_c -axis around the z_c -axis of the reference coordinate system.

Spacing (*spacing*) [real number with unit of length].

Spacing of the wires in the grid, measured in the $x_c y_c$ -plane of the reference coordinate system as the distance from the centre line of one wire to the centre line of the next wire.

Diameter (*diameter*) [real number with unit of length].

The diameter of the wires in the grid.

Plot Lines (*plot_lines*) [integer], default: **11**.

Number of lines (approximately) drawn in a plot of the scatterer having this grid as electrical property.

Remarks

The layer with the wire grid is assumed to be infinitely thin.

Note, that when a *Wire Grid* is specified for a scatterer then a conducting surface - if such one shall be included in the model - must be specified as a separate layer (e.g. of class *Perfect Conductivity*) in the specification of the electrical properties of the scatterer.

The following Figure 1 illustrates a *Wire Grid* at the surface of a scatterer. The Displacement is specified to 0.0 mm. The next Figure 2 illustrates a *Reflector* with a *Wire Grid* positioned 20 mm in front of it. Thus, the Displacement is specified to 20.0 mm. Further, the conducting surface of the *Reflector* must be included by specifying a layer having *Perfect Conductivity*.



Figure 1

Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with a *Wire Grid* at the surface of a scatterer. As the grid is located at the surface of the scatterer, the Displacement is zero.

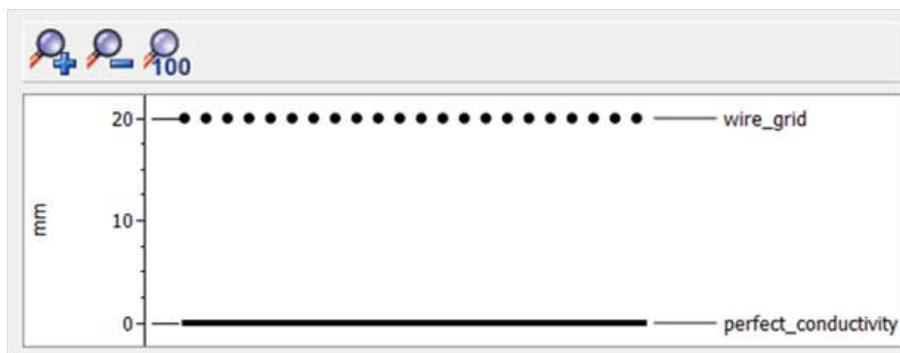


Figure 2

Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with a *Wire Grid* positioned 20 mm above a *Reflector*. The Displacement for the grid is thus specified to 20 mm. The reflector itself is specified as a layer of *Perfect Conductivity*.

As seen in the above figures, the *Wire Grid* has a signature given by a series of large dots in the illustration of the specified Electrical Properties. The values of the Spacing and the Diameter of the wires are not reflected in the illustrations.

If the *Wire Grid* is placed on a dielectric layer, which in turn is placed on the specified surface of the scatterer, it is necessary to define where the grid surface is located relative to the specified surface. In such case, the displacement of the reference plane along the positive surface normal shall be equal to the thickness of the dielectric. The dielectric layer is defined in class *Dielectric Layer*.

The grid is deposited on the curved reflector as a projection from some direction, e.g. the boresight. Seen from this direction the strips will be parallel and equispaced on an aperture plane. The direction defines the z_c -axis, and the x_c - and y_c -axis define the aperture plane, where the strips by default are parallel to the x_c -axis, see Figure 3. It is, however, possible to define a rotation angle, Reference Angle (α in Figure 3), of the strip direction with respect to the x_c -axis.

The orientation of the *Wire Grid* can be illustrated on a plot of the scatterer. The number of grid lines drawn on the scatterer are specified by Plot Lines. Because of the way the lines are generated, the actual numbers of lines drawn may be slightly smaller than Plot Lines for some scatterers.

The grid lines will be drawn on the surface of the scatterer and the value of Displacement is thus not illustrated in the plot.

In order to see the grid clearly in the 3D-view the surface lines on the scatterer may be hidden (by right-clicking the 3D view canvas).

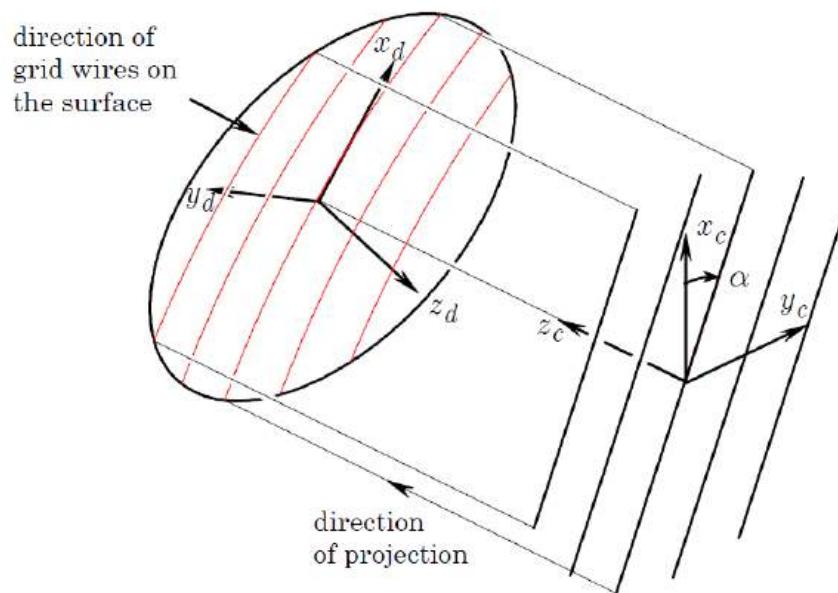


Figure 3

The reference coordinate system, $x_c y_c z_c$, specifying the local orientation of the material definition coordinate system, $x_d y_d z_d$, on the reflector surface. The projected lines are parallel and equispaced in the $x_c y_c$ -plane and rotated the angle α with respect to the x_c -axis.

WIRE MESH (wire_mesh)

Purpose

The class **Wire Mesh** defines the conductivity of a surface by the properties of a mesh consisting of two sets of orthogonal wires with a given thickness and a spacing, which can be different for the two directions. The scatterer is assumed to be divided into segments, as would be the case for an unfurlable rib reflector (surface describing class: *Unfurlable Surface, Rib Attached*), each segment having the mesh oriented with one set of wires parallel to the angle bisector of the rib segment.

Links

Classes→*Electrical Objects*→*Electrical Properties*→*Wire Mesh*

Remarks

Syntax

```
<object name> wire_mesh
(
    displacement           : <rl>,
    ref_coor_sys          : ref(<n>),
    n_segments             : <i>,
    ref_angle               : <r>,
    spacing                : struct(a:<rl>, b:<rl>),
    wire_radius            : <rl>,
    plot_lines              : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
```

Attributes

Displacement (*displacement*) [real number with unit of length], default: **0**.

Defines where the surface of the mesh is located relative to the specified surface of the scatterer. The Displacement is positive in direction of the positive surface normal of the *Scatterer*.

Reference Coordinate System (*ref_coor_sys*) [name of an object].

Reference to an object of one of the *Coordinate Systems* classes. The mesh orientation is defined with respect to this $x_c y_c z_c$ -coordinate system. A reference must be given, as there is no default value for this attribute.

n-Segments (*n_segments*) [integer].

Number of mesh segments (ribs). For an unfurlable rib reflector the number of segments and their orientation may coincide with the ribs specified in the surface described by class *Unfurlable Surface, Rib Attached*. The minimum number of segments is two, and in that case the mesh will have same orientation over the entire reflector surface (see the remarks below).

Reference Angle (*ref_angle*) [real number], default: **0**.

Angle from the x_c -axis to the first rib, positive around the z_c -axis (axes of the reference coordinate system).

Spacing (*spacing*) [struct].

Spacing between the wires in the mesh.

A (a) [real number with unit of length].

Spacing between wires parallel to the rib-segment bisector, "A"-wires.

B (b) [real number with unit of length].

Spacing between "B"-wires, orthogonal to the "A"-wires.

Wire Radius (*wire_radius*) [real number with unit of length].

Radius of the wires of the mesh.

Plot Lines (*plot_lines*) [integer], default: **11**.

Control of the number of lines drawn in a plot of the scatterer having this mesh as electrical property. Lines are plotted parallel to the "B"-wires of each segment. The spacing is chosen such that the number of lines - if continued across the full reflector - is Plot Lines. The number of lines parallel to the "A"-wires will be adjusted to reflect the ratio between a and b as defined in Spacing above. See the examples in the remarks below.

Remarks

The layer with the mesh is assumed to be infinitely thin.

Note, that when a *Wire Mesh* is specified for a scatterer then a conducting surface - if such one shall be included in the model - must be specified as a separate layer (e.g. of class *Perfect Conductivity*) in the specification of the electrical properties of the scatterer.

The following Figure 1 illustrates a *Wire Mesh* as the surface of a scatterer. The Displacement is specified to 0.0 mm. The next Figure 2 illustrates a *Reflector* with a *Wire Mesh* positioned 20 mm in front of it. Thus, the Displacement is specified to 20.0 mm. Further, the conducting surface of the *Reflector* must be included by specifying a layer having *Perfect Conductivity*.



Figure 1 Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with a **Wire Mesh** as the surface of a the scattering and the Displacement is zero.

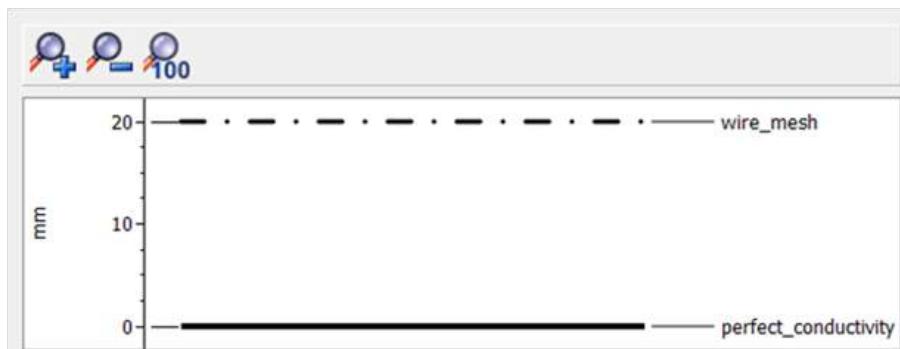


Figure 2 Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with a **Wire Mesh** positioned 20 mm above a **Reflector**. The Displacement for the mesh is thus specified to 20 mm. The reflector itself is specified as a layer of **Perfect Conductivity**.

As seen in the above figures, the **Wire Mesh** has a signature given by a series of dashes and dots in the illustration of the specified Electrical Properties. The values of the Spacings and the Wire Radius of the wires are not reflected in the illustrations.

If the **Wire Mesh** is placed on a dielectric layer, which in turn is placed on the specified surface of the scatterer, it is necessary to define where the mesh surface is located relative to the specified surface. In such case, the displacement of the reference plane along the positive surface normal shall be equal to the thickness of the dielectric. The dielectric layer is defined in class **Dielectric Layer**.

The mesh is deposited on the curved reflector as a projection from some direction, e.g. the boresight. Seen from this direction the wires of the mesh will consist of two orthogonal sets of parallel and equispaced wires on an aperture plane. The direction defines the z_c -axis, and the x_c -axis and y_c -axis define the aperture plane.

The segments of the mesh are defined as equal angular segments in the $x_c y_c$ -plane. The segments are rotated such that the angle from the x_c -axis to the first rib is Reference Angle (ϕ_0 Figure 3).

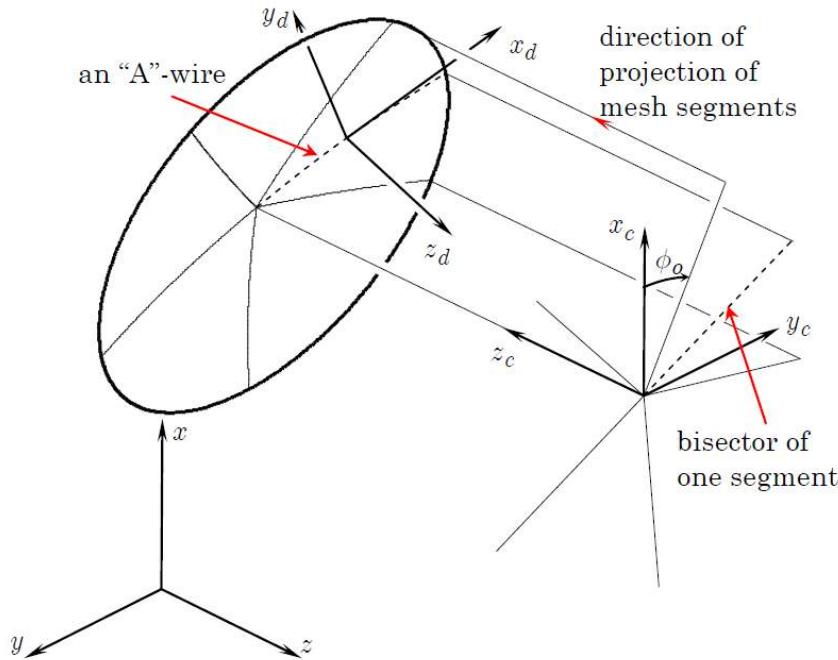


Figure 3

The reference coordinate system, $x_c y_c z_c$, specifying the local orientation of the mesh definition coordinate system, $x_d y_d z_d$, on the reflector surface. The latter defines the mesh, segment for segment. In this example n-Segments is 5.

The mesh is oriented with wires (the "A"-wires) parallel to the bisector of the ribs of the segment or, more precisely, parallel to the plane defined by the bisector and the z_c -axis. The bisector is shown dotted in the Figure. The orthogonal set of wires, the "B"-wires, will be assumed locally perpendicular to the "A"-wires. The spacing between the "A"- and "B"-wires is a and b, respectively.

It is a mathematical assumption that the "B"-wires locally are perpendicular to the "A"-wires. This is not physically possible when the surface is not unfurlable.

For the case of a single mesh over the full reflector (n-Segments = 2), the "A"-wires will be parallel to the y_c -axis and the "B"-wires will be parallel to the x_c -axis when Reference Angle is specified to 0° , cf. the last of the following two Figures.

The orientation of the mesh wires is illustrated in the following two examples of an offset reflector with 5 segments and no segments which is obtained by specifying n-Segments to 5 and 2, respectively. The plots are made by [Reflector Plot](#).

For both examples a total of 11 lines in the direction of the "B"-wires is specified across the full reflector (default value of Plot Lines). This results in 5 "B"-wires on each segment. When n-Segments is specified to 2 the segments will each cover half of the surface and the wires of the two segments will be parallel. The surface acts as a non-segmented surface.

The number of lines in the "A"-wire direction is adjusted to reflect the relative

spacing between the "A"- and "B"-wires.

The radius of the grid wires is not illustrated in the plots, and neither is any displacement of the mesh relative to the surface.

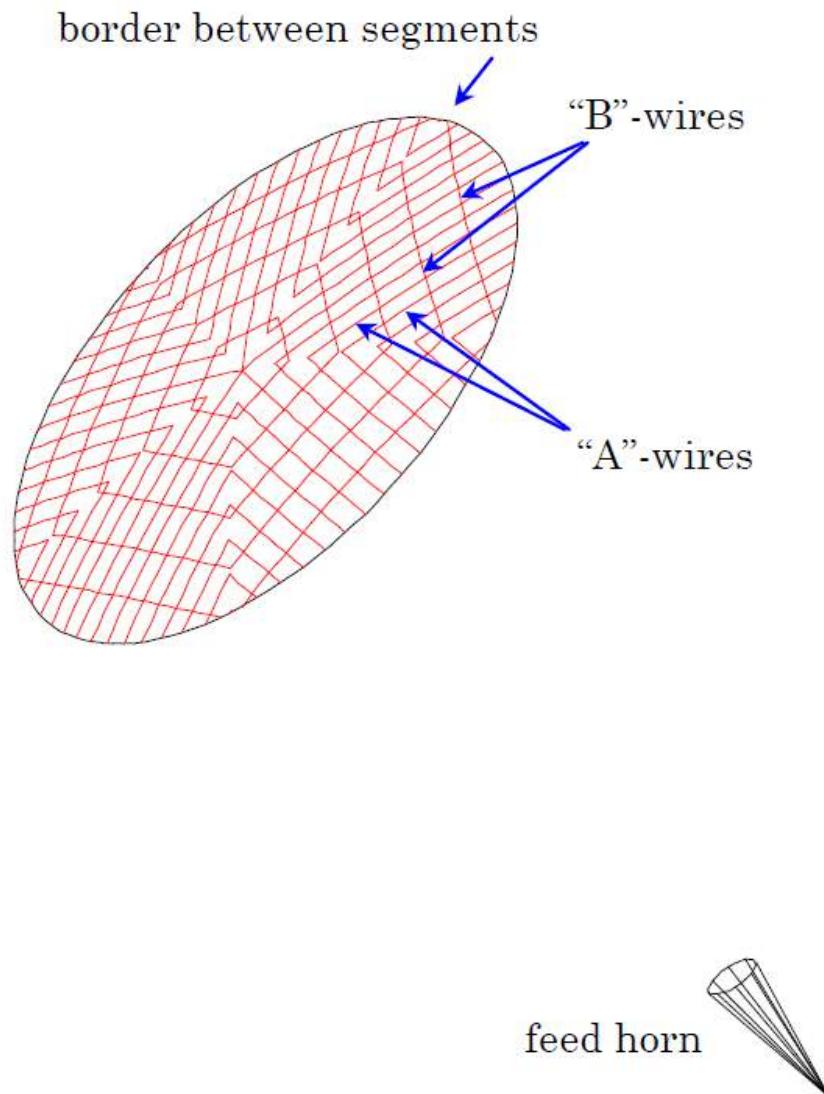


Figure 4

Illustration of a mesh upon the same reflector as in the previous figure with n-Segments specified to 5. The spacing of the "A"-wires is half the spacing of the "B"-wires, Plot Lines is specified to 11.

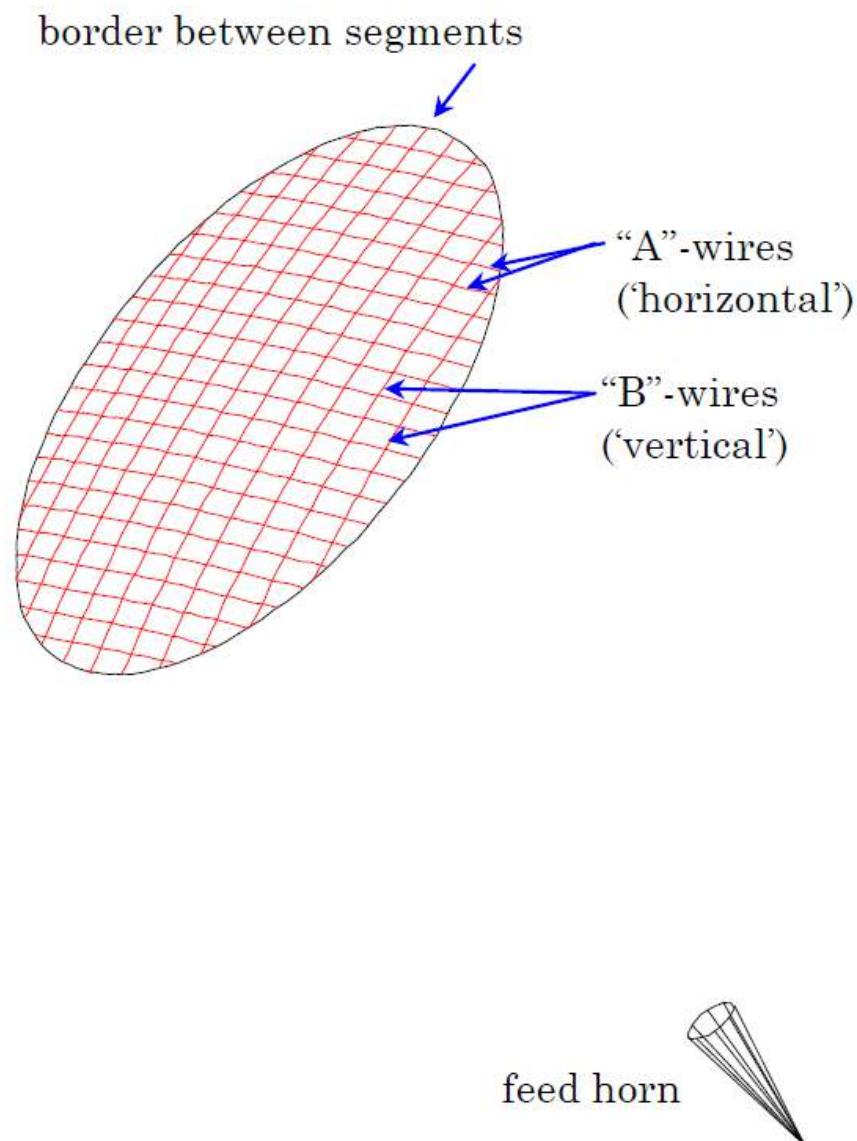


Figure 5

The same reflector with n-Segments specified to 2. The other attributes are unchanged.

DIELECTRIC LAYER (dielectric_layer)

Purpose

The class *Dielectric Layer* defines the electrical properties of a dielectric layer together with its thickness and relative position.

Links

Classes→*Electrical Objects*→*Electrical Properties*→*Dielectric Layer*

Remarks

Syntax

```
<object name> dielectric_layer
(
    displacement           : <rl>,
    thickness              : <rl>,
    dielectric_constant    : <r>,
    loss_tangent           : <r>
)
where
<r> = real number
<rl> = real number with unit of length
```

Attributes

Displacement (*displacement*) [real number with unit of length], default: **0**.

Defines where the reference surface for the dielectric layer is located relative to the specified surface of the scatterer (the reference surface is usually the 'front' side of the layer, see the remarks below). The displacement is positive in direction of the positive surface normal of the *Scatterer*.

Thickness (*thickness*) [real number with unit of length].

Thickness of the dielectric layer, must be positive or zero.

Dielectric Constant (*dielectric_constant*) [real number].

Relative dielectric constant of the layer.

Loss Tangent (*loss_tangent*) [real number], default: **0**.

Loss tangent of the dielectric.

Remarks

General Remarks

Note, that when a *Dielectric Layer* is specified for a scatterer then a conducting surface - if such one shall be included in the model - must be specified

as a separate layer (e.g. of class *Perfect Conductivity*) in the specification of the electrical properties of the scatterer.

The following Figure 1 illustrates a *Reflector* with a *Dielectric Layer* of Thickness 7 mm positioned upon the reflector surface. As the Displacement shall refer to the positive side of the *Dielectric Layer* it shall also be 7 mm. Further, the conducting surface of the *Reflector* must be included by specifying a layer having *Perfect Conductivity*. See also Figure Figure 3 for this case.



Figure 1 Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with a *Dielectric Layer* on a *Reflector*. The Thickness of the layer is 7 mm and as the *Dielectric Layer* is placed on the surface of the reflector then the Displacement for the layer is also specified to 7 mm. The reflector itself is specified as a layer of *Perfect Conductivity*.

As seen in the above figure, the *Dielectric Layer* has a signature showing the Thickness of the layer in the illustration of the specified Electrical Properties. The layer is shown in a shade of green, the density of which depends on the value of the specified Dielectric Constant.

The effects of the dielectric layer are calculated by means of reflection and transmission coefficients for an infinite plane of the material illuminated by a plane wave. Near-field effects of the illuminating field are thus not included; this may be included by expanding the field in plane waves, cf. class *Plane Wave Expansion*.

A dielectric layer has a finite thickness, which defines two sides of the layer, a backside and a front side. These are defined such that the positive normal of the surface of the scatterer is directed from the backside to the front side of the layer.

Complex Dielectric Constant

A complex value for the dielectric constant may be defined. The dielectric constant, ϵ , shall be replaced by the complex dielectric constant, ϵ' , defined as

$$\epsilon' = \epsilon - j(\sigma/\omega) = \epsilon[1 - j(\sigma/(\omega\epsilon))] = \epsilon[1 - j\tan\delta] \quad (1)$$

where j is the complex unit (time factor $e^{j\omega t}$), ω is the angular frequency and σ is the conductivity. With ϵ_0 being the free-space dielectric constant

and ϵ_r is the relative dielectric constant we also have

$$\epsilon = \epsilon_0 \epsilon_r \quad (2)$$

It is the the value of ϵ_r which shall be inserted for Dielectric Constant and the value of $\tan \delta$ which shall be used for Loss Tangent. Both shall be non-negative.

Positioning of the Layer

The reference surface is defined as the *front* side of the dielectric layer and the attribute Displacement defines the distance from the specified surface of the scatterer to the reference surface, the front surface, of the dielectric layer, Figure 2.

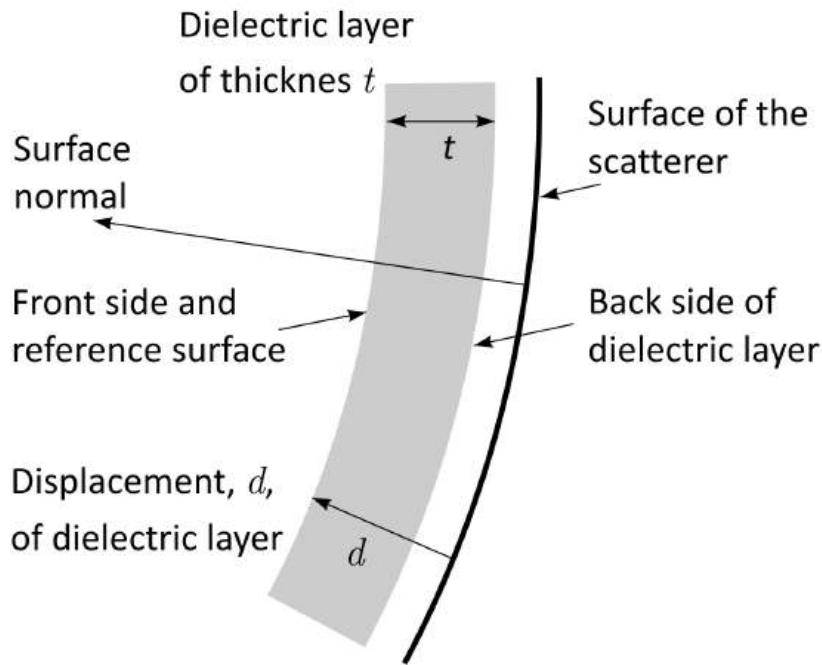


Figure 2

Dielectric Layer of Thickness t and with Displacement d .

As an example, consider a layer of paint (specified as a *Dielectric Layer*) with Thickness t on top of a metallic surface (specified as a object of *Perfect Conductivity*), Figure 3. The perfect conductivity of the metallic surface would be placed at the specified surface, i.e. the Displacement of this is specified to zero (in class *Perfect Conductivity*). The reference surface of the paint is the front side, which is positioned the layer thickness in front of the metallic surface, i.e. the Displacement is t to be specified in class *Dielectric Layer*. In addition, the Thickness of the layer is t .

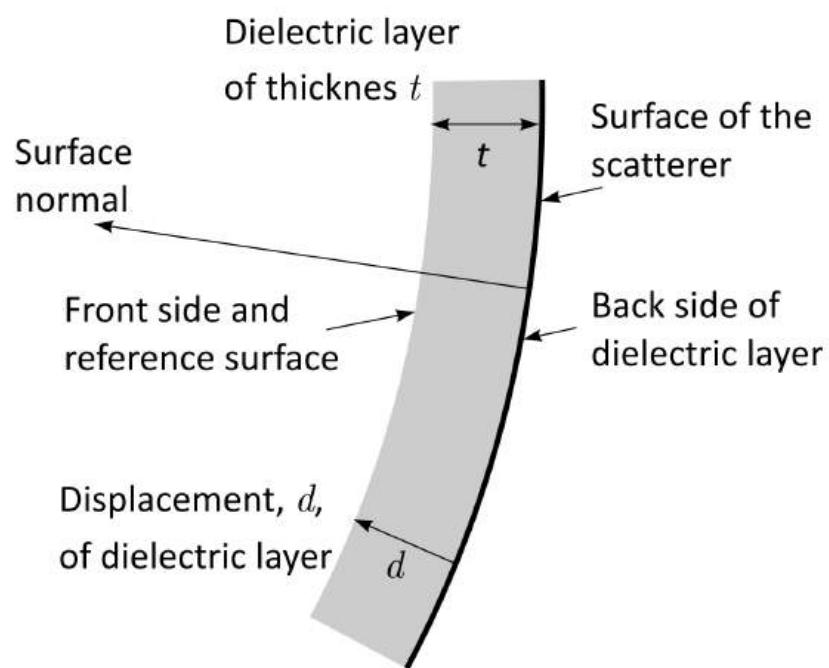


Figure 3

Dielectric Layer (e.g. paint) placed directly on the reflector surface. The Displacement d equals the Thickness t .

PERFECT CONDUCTIVITY (perfect_conductivity)

Purpose

The class **Perfect Conductivity** defines the electrical properties of a surface by means of a perfect conductor. This is already the default for all **Scatterers**, but the class is useful for surfaces composed of several materials, for example a layer of paint on a perfectly conducting surface. This could be modelled as the **Perfect Conductivity** and a **Dielectric Layer**.

Links

[Classes](#)→[Electrical Objects](#)→[Electrical Properties](#)→[Perfect Conductivity](#)

Remarks

Syntax

```
<object name> perfect_conductivity
(
    displacement           : <rl>
)
where
<rl> = real number with unit of length
```

Attributes

Displacement (*displacement*) [real number with unit of length], default: **0**.

Defines where the perfect conducting surface is located relative to the specified surface of the scatterer. The displacement is positive in direction of the positive surface normal.

Remarks

The layer with perfect conductivity is assumed to be infinitely thin.

The following Figure 1 illustrates a **scatterer** having **Perfect Conductivity**. Usually the Displacement shall be specified to zero.

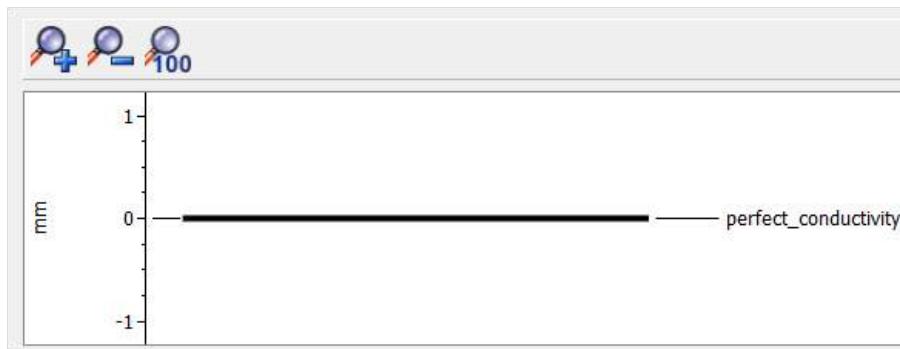


Figure 1

Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with a *Perfect Conductivity* on a *scatterer*. The *Perfect Conductivity* is placed as the surface of the *scatterer* by specifying Displacement for the layer to 0 (this is default).

As seen in the figure, the *Perfect Conductivity* has a thick line as signature.

PERFECT ABSORPTION (perfect_absorption)

Purpose

The class **Perfect Absorption** defines the electrical properties of a surface by means of an imaginary layer that absorbs all the incident energy. In other words, the reflection and transmission coefficients are all zero.

Links

[Classes](#)→[Electrical Objects](#)→[Electrical Properties](#)→[Perfect Absorption](#)

Remarks

Syntax

```
<object name> perfect_absorption
(
    displacement           : <rl>
)
where
<rl> = real number with unit of length
```

Attributes

Displacement (*displacement*) [real number with unit of length], default: **0**.

Defines where the surface layer with perfect absorption is located relative to the specified surface of the scatterer. The displacement is positive in direction of the positive surface normal.

Remarks

The surface layer with perfect absorption is assumed to be infinitely thin.

The following Figure 1 illustrates a **scatterer** having **Perfect Absorption**.

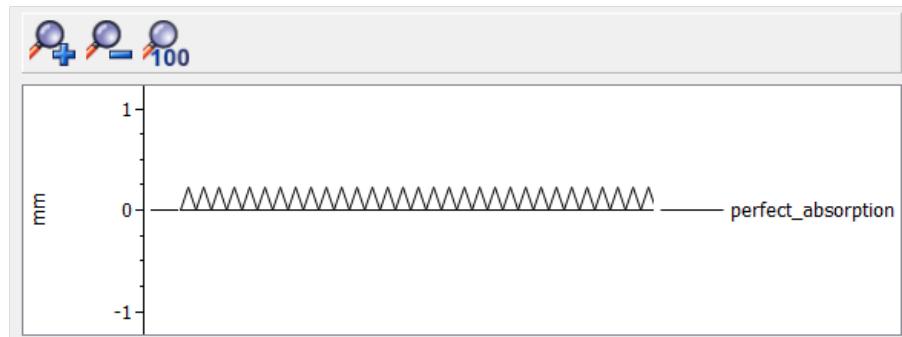


Figure 1 Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with a **Perfect Absorption** at the surface of a **scatterer**.

As seen in the figure, the **Perfect Absorption** is illustrated as a series of pyramidal absorbers.

POWER SPLITTING (power_splitting)

Purpose

The class **Power Splitting** defines the electrical properties of a power splitter together with its relative position.

Links

Classes→*Electrical Objects*→*Electrical Properties*→*Power Splitting*

Remarks

Syntax

```
<object name> power_splitting
(
    displacement           : <rl>,
    characteristics       : sequence(
                                struct(
                                    power_reflection:<r>,
                                    power_transmission:<r>,
                                    f_min:<rf>,
                                    f_max:<rf>),
                                ...))
where
<r> = real number
<rf> = real number with unit of frequency
<rl> = real number with unit of length
```

Attributes

Displacement (*displacement*) [real number with unit of length], default: **0**.

Defines where the reference surface for the power splitter is located relative to the specified surface of the scatterer (see the remarks below). The displacement is positive in direction of the positive surface normal of the **Scatterer**.

Characteristics (*characteristics*) [sequence of structs].

The attributes of the power splitter may depend on the frequency. Each struct in this sequence characterises the power reflection and transmission characteristics of the power splitter over a certain frequency band.

Power Reflection (*power_reflection*) [real number], default: **100**.

Percentage of the incident power which is reflected by the power splitter. The value must be in the interval from 0 to 100.

Power Transmission (*power_transmission*) [real number], default: **0**.

Percentage of the incident power which is transmitted by the power splitter. The value must be in the interval from 0 to 100.

Frequency Minimum (f_{min}) [real number with unit of frequency], default: **0**.

The minimum frequency of the frequency band over which the above values for power reflection and transmission apply.

Frequency Maximum (f_{max}) [real number with unit of frequency], default: **-1**.

The maximum frequency of the frequency band over which the above values for power reflection and transmission apply. Specifying f_{min} greater than f_{max} has a special significance, see the remarks below.

Remarks

The power-splitting layer models an infinitely thin surface which partly reflects and partly transmits the incident field and it is thus a simple model of a frequency selective reflector as applied in e.g. a beam waveguide.

The reflected field is the percentage of the field reflected from a perfect conductor. Thus, the amplitude is different, but the phase and the polarisation are as for a perfectly reflected field.

Similarly, the transmitted field is the percentage of the field transmitted through free space. Again, the phase and the polarisation are as for free-space transmission.

The sum of the power-reflection percentage and the power-transmission percentage must not exceed 100%. The sum may be less than 100% to simulate loss in the power splitter.

The values apply only in the related frequency range from f_{min} to f_{max} . Since the attribute is a sequence it is possible to give several frequency bands with corresponding reflection- and transmission data.

It is allowed to specify f_{min} greater than f_{max} for one and only one member of the sequence. In that case, the corresponding values for reflection and transmission apply to all other frequencies than those defined by the other members of the sequence.

Overlapping frequency bands are not allowed. Attempt to calculate surface material parameters at a frequency that is not defined in the characteristics attribute will result in an error message.

The following Figure 1 illustrates a power splitter.

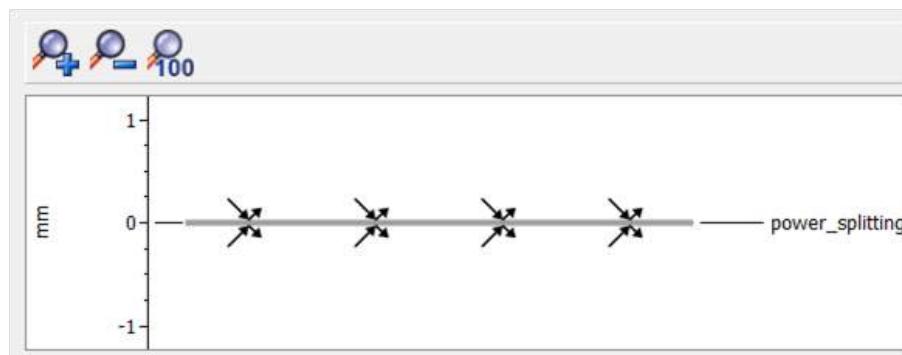


Figure 1 Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with a *Power Splitting* at the surface of a *scatterer*.

As seen in the figure, the *Power Splitting* is showed as a series of arrows illustrating incoming and reflected/transmitted power at the surface.

FINITE CONDUCTIVITY (finite_conductivity)

Purpose

The class *Finite Conductivity* defines the conductivity of a scatterer for which the conductivity is not infinite. An example is a CFRP reflector for which the carbon fibres are not perfectly conducting.

Links

Classes→*Electrical Objects*→*Electrical Properties*→*Finite Conductivity*

Remarks

Syntax

```
<object name> finite_conductivity
(
    displacement           : <rl>,
    conductivity          : <rc>
)
where
<rc>= real number with conductivity unit, S/m
<rl> = real number with unit of length
```

Attributes

Displacement (*displacement*) [real number with unit of length], default: **0**.

Defines where the finite conductivity layer is located relative to the specified surface of the scatterer. The displacement is positive in direction of the positive surface normal.

Conductivity (*conductivity*) [real number with conductivity unit, S/m].

Conductivity of the surface material measured in Siemens per meter (S/m).

Remarks

There is no transmission through a layer with *Finite Conductivity*.

As an alternative a *Dielectric Layer* may be applied.

The following Figure 1 illustrates a *scatterer* having *Finite Conductivity*.

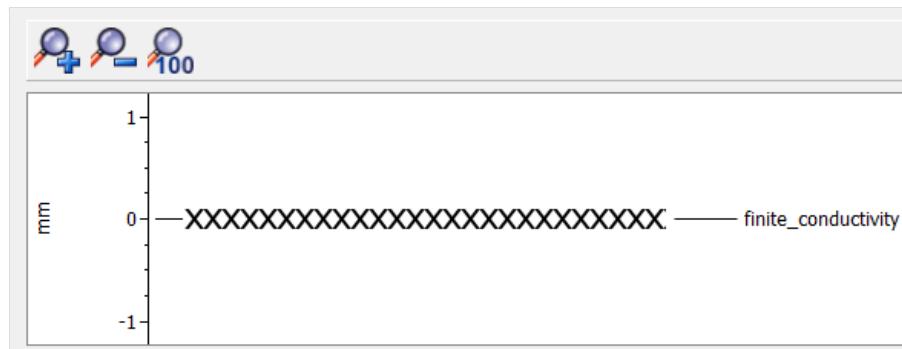


Figure 1

Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with a *Finite Conductivity* on a *scatterer*. The *Finite Conductivity* is placed as the surface of the *scatterer* by specifying Displacement for the layer to 0 (this is default).

As seen in the figure, the *Finite Conductivity* has a series of Xs as signature.

TABULATED ELECTRICAL PROPERTIES (tabulated_el_prop)

Purpose

The class *Tabulated Electrical Properties* defines the electrical properties of a surface by means of tabulated data for the reflection and transmission coefficients.

Links

Classes→*Electrical Objects*→*Electrical Properties*→*Tabulated Electrical Properties*

Remarks

Syntax

```
<object name> tabulated_el_prop
(
    displacement          : <rl>,
    file_name             : <f>,
    frequency             : ref(<n>),
    ref_coor_sys          : ref(<n>),
    ref_angle              : <r>,
    list                  : <si>
)
```

where

<n> = name of an object
 <r> = real number
 <rl> = real number with unit of length
 <f> = file name
 <si> = item from a list of character strings

Attributes

Displacement (*displacement*) [real number with unit of length], default: **0**.

Defines where the surface, to which the tabulated data refer, is located relative to the specified surface of the scatterer (see the remarks below). The displacement is positive in direction of the positive surface normal of the *Scatterer*.

File Name (*file_name*) [file name].

Name of file containing tabulated reflection and transmission coefficients. The file format shall be in accordance with the format for tabulated electrical properties as described in *Tabulated Electrical Properties for Scatterers*.

Frequency (*frequency*) [name of an object], default: **blank**.

Reference to a *Frequency* object. See the remarks below.

Reference Coordinate System (*ref_coor_sys*) [name of an object].

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system $x_c y_c z_c$. The z_c -axis defines the direction from where the tabulated surface parameters shall be projected on to the actual surface (see the remarks below). A reference must be given, as there is no default value for this attribute.

Reference Angle (*ref_angle*) [real number], default: **0**.

The surface parameter data may originate from a system which is rotated by this angle, positively around the z_c -axis (see the remarks below).

List (*list*) [item from a list of character strings], default: **off**.

Specifies whether a list of the reflection and transmission coefficients shall be reproduced in the standard output file.

Remarks

The layer with the specified reflection and transmission coefficients is assumed to be infinitely thin.

If the layer is combined with other layers, then there is a risk for the physical extent of the layers are overlapping as this can not directly be seen in the illustration of the VIEW ELECTRICAL PROPERTIES in the Object Editor for the *scatterer* for which the *Tabulated Electrical Properties* are defined. It is therefore recommended that the user inspect this window, VIEW ELECTRICAL PROPERTIES, and - from knowledge of the thicknesses of the tabulated properties - check that the layers do not overlap.

Further, in the case of more layers, it is recommended to have the reference surface (positioned by the attribute Displacement) of the *Tabulated Electrical Properties* positioned at the top of the layer. GRASP sorts the layers according to the positions of the reference surfaces, and if these are arbitrarily positioned then the layers may be analyzed in a wrong order.

The following Figure 1 illustrates a *scatterer* having *Tabulated Electrical Properties*.

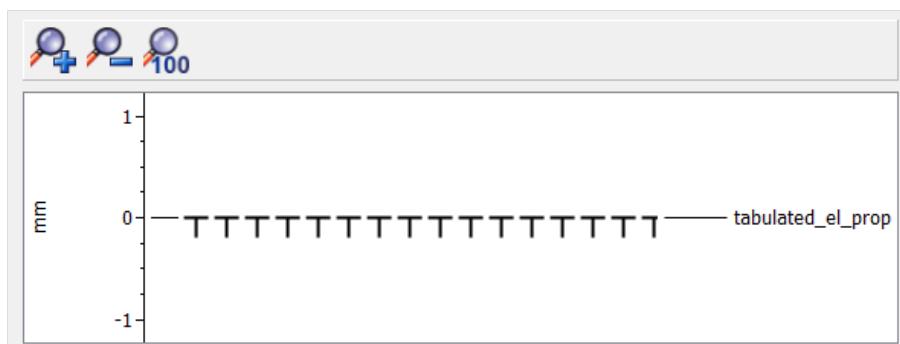


Figure 1

Screen shot from VIEW ELECTRICAL PROPERTIES in GRASP with *Tabulated Electrical Properties* at the surface of a *scatterer*.

As seen in the figure, the *Tabulated Electrical Properties* is illustrated as a series of Ts.

The transmission and reflection coefficients are tabulated in a grid of the angles of incidence (θ, ϕ) and a third order interpolation is carried out between the grid points to obtain the coefficients for the actual angles. When a requested angle is outside the tabulated grid the coefficients will be determined by extrapolation, and a warning message is issued.

The frequencies given by the reference to a *Frequency* object shall be in accordance with the frequencies for which the tabulated data are given in the file. The *Frequency* object shall be the same as that for which the calculations are to be carried out.

A *Frequency* object need not to be specified. As default the first data set in the file will then be used for all frequencies in the calculations.

The tabulated data are defined for different angular incidences on a planar, infinite structure with the given reflection and transmission coefficients. The coefficients are tabulated for angles of incidence, (θ_d, ϕ_d) in usual spherical coordinates in polar cuts with the normal of the plane being the z_d -axis. The plane of the structure is the $x_d y_d$ -plane with the x_d -axis defining the reference direction $\phi_d = 0$. For more than one frequency, defined by the *Frequency* object, the data set must be repeated in the file.

The coefficients that relate to the generally curved surface of the scatterer are now defined. It is obvious that the z_d -axis at any surface point shall be directed along the positive surface normal, see the Figure below. The x_d -axis will be oriented by means of the attribute `ref_coor_sys` defining a reference coordinate system $x_c y_c z_c$. The $x_c y_c$ -plane is an aperture plane, which can be projected on the surface along the z_c -axis. The reference direction on the surface, x_d or $\phi_d = 0$, follows the projection of a line which in the $x_c y_c$ -plane is rotated the angle `ref_angle` (α in the Figure) from the x_c -axis.

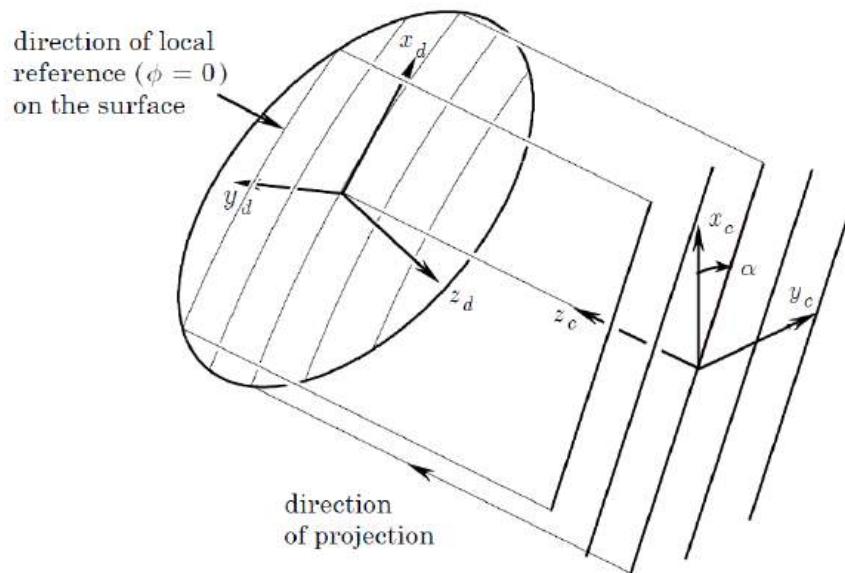


Figure 2

The reference coordinate system, $x_c y_c z_c$, specifies the local orientation of the material definition coordinate system, $x_d y_d z_d$, on the reflector surface. The projected lines are parallel in the $x_c y_c$ -plane and rotated the angle α with respect to the x_c -axis.

DERIVED ELECTRICAL DATA

Purpose

The following classes for analysis and output specifications are available for tabulating electrical data:

The field from a radiating structure may be uniquely expanded and given by the coefficients of a

Spherical Wave Expansion (SWE)

To generate tables with the electrical properties of a reflector, use :

Electrical Properties Data Output

The same data as well as geometrical data may be generated with an object of class *Reflector Data Output*. These two classes are only for data management, and have no impact on RF calculations.

Coupling add-on: The next classes are part of the add-on package Coupling for calculation of the coupling of two antennas. The coupling system is defined by an object of class

Coupling System

For backward compatibility an obsolete version of this class is still available:

Coupling (Obsolete) (obsolete, use Coupling System instead).

Links

Classes→*Electrical Objects*→*Derived Electrical Data*

SPHERICAL WAVE EXPANSION (SWE) (swe)

Purpose

In the class *Spherical Wave Expansion (SWE)* it is specified how the spherical wave coefficients of a field will be calculated. The specifications include how the field shall be determined in a regular grid on a far-field sphere and where the coefficients shall be stored.

The spherical wave coefficients constitute a unique way to define a field (e.g. of a measured feed) from which the near field as well as the far field then may be constructed.

Links

Classes→*Electrical Objects*→*Derived Electrical Data*→*Spherical Wave Expansion (SWE)*

Remarks

Syntax

```
<object name> swe
(
    file_name           : <f>,
    sphere_sample       : struct(n_phi:<i>, n_theta:<i>),
    power_norm          : <si>,
    list                : <si>,
    file_format         : <si>,
    comment             : <s>,
    obsolete_near_far  : <si>,
    obsolete_sphere_radius : <rl>,
    obsolete_file_coef  : <si>,
    obsolete_frequency   : ref(<n>)
)
where
<i> = integer
<n> = name of an object
<rl> = real number with unit of length
<s> = character string
<f> = file name
<si> = item from a list of character strings
```

Attributes

File Name (*file_name*) [file name].

Name of a file, to which the calculated field values shall be written.

Sphere Sample (*sphere_sample*) [struct].

Defines the number of field samples on the sphere (see the remarks below) along the ϕ - and θ -coordinates, respectively.

N phi (*n_phi*) [integer].

Number of field samples along a full azimuth circle, $0^\circ < \phi < 360^\circ$; N phi must be even and larger than or equal to 4.

N theta (*n_theta*) [integer].

Number of field samples along a half polar circle, $0^\circ \leq \theta \leq 180^\circ$, both end points included; N theta must be larger than or equal to 3.

Power Norm (*power_norm*) [item from a list of character strings], default: **off**.

Power normalisation of the field expansion.

off

The field is not normalised.

on

The power contained in the spherical wave expansion is normalised to 4π . Hereby a field calculated from the coefficients is given relative to isotropic level.

List (*list*) [item from a list of character strings], default: **off**.

Specifies whether a list of the spherical coefficients shall be reproduced in the standard output file.

off

No list is generated.

on

A list is generated.

File Format (*file_format*) [item from a list of character strings], default: **TICRA**.

The file format used to store the field data:

TICRA

The file is stored as an ASCII-file and fulfills the TICRA-format described in *Spherical Wave ab-Coefficients* and *Spherical Wave Q-Coefficients*. The recommended file extension is *.sph*.

EDX

The file is stored in a format according to the Electromagnetic Data Exchange (EDX) standard.

EDI

Obsolete file format name. The same as EDX.

Comment (*comment*) [character string], default: **SWE**.

A line of text which will be written as a header in the file specified by File Name above.

Near/far (Obsolete) (*obsolete_near_far*) [item from a list of character strings], default: **far**.

Specifies whether the sphere of field samples is positioned at a finite or an infinite distance from the source. See the remarks below for the orientation and position of the sphere. The attribute is obsolete and the default value is recommended.

far

The sphere is at an infinite distance, a far field is calculated.

near

The sphere is at a finite distance, a near field is calculated.

Sphere Radius (Obsolete) (*obsolete_sphere_radius*) [real number with unit of length], default: **0**.

Radius of the sphere on which near-field data are calculated. Not used when *near_far* is specified to 'far'. The attribute is obsolete and the default value is recommended.

Coefficients in File (Obsolete) (*obsolete_file_coef*) [item from a list of character strings], default: **Q**.

Specifies the type of spherical coefficients to be generated. The attribute is obsolete.

Q

The coefficients are the *Q*-coefficients (recommended)

ab

The coefficients are the *ab*-coefficients (obsolete, for compatibility only).

Frequency (Obsolete) (*obsolete_frequency*) [name of an object], default: **blank**.

The attribute is obsolete and is maintained for backward compatibility. The attribute is ignored.

Command Types

The calculation of the spherical coefficients is activated by the command:

Get SWE

in which the source, for which the spherical coefficients are determined, is specified.

Remarks

The spherical wave expansion is a very precise way to characterize the field from a source, and this class, *Spherical Wave Expansion (SWE)*, is used for generating the coefficients of the expansion.

When a field of a *Feed* is calculated, this calculation is automatically determined on the basis of a spherical expansion of the far field of the feed and objects of class *Spherical Wave Expansion (SWE)* need not to be set up by the user³.

Objects of the present class shall then be used in cases when the coefficients of the spherical wave expansion are needed for export or when special attention is requested such as for a (measured) pattern for which the most correct values of the number of modes are uncertain.

This section contains remarks on

- The reference centre for the SWE,
- The operation of the SWE class,
- Special considerations for measured patterns,
- The basic concepts of the SWE,
- The sample spacing.
- Examples.

The section concludes with two examples on the application of the SWE. Other examples are found under the remarks to the various *Feeds*. The example given in the class *Tabulated Pattern* is particularly important in relation to expansion of measured patterns. The example found in class *Elliptical Pattern* illustrates the consequences of trying to expand a feed model which is given as a simple mathematical model not fulfilling Maxwell's equations.

Reference Centre

The SWE is calculated from the field on a sphere. The expansion depends on the position and the orientation of this sphere with respect to the source generating the field. The sphere is therefore defined in the coordinate system of this source with the polar axis of the sphere being the z -axis of the coordinate system and the centre of the sphere being at the origin of the system. The coordinate system is specified by the relevant attribute of the actual *Source* object.

For *Sources* fulfilling Maxwell's equations there will be no difference in the spherical expansions based on evaluation of the field at different distances, near field or far field. The *Feeds* of GRASP all fulfill Maxwell's equations as the fields are based on spherical coefficients (which originally are determined from the far field of the actual *Feed* model).

The *Source* object is specified in the *Get SWE* command by which the coefficients are calculated.

³The exceptions from this rule constitute the Gaussian beam feeds, *Gaussian Beam, Pattern Def.* and *Gaussian Beam, Near-Field Def.* (which includes a precise near-field model), and the spherical wave expansion feed, *Tabulated SWE Coefficients* (which already is expressed in spherical modes).

Operation

Objects of the class *Spherical Wave Expansion (SWE)* are used for defining a SWE of a radiating field. The field originate from a source which is specified in the command *Get SWE*. When this command is issued the SWE is calculated in two steps. First, the field of the source is calculated on a sphere related to this source (as specified in the attribute *Sphere Sample*) and second, the requested expansion coefficients are calculated from that field. When the source is a *Feed* an internal SWE will automatically be carried out to generate the field. This increases the computation time, and for a *Tabulated Pattern* it may also introduce interpolation inaccuracies. The coefficients are stored on the specified file. The Q-coefficients are standard.

The field is completely described by the coefficients for all angular directions and distances – including the far field – outside a minimum sphere. In a reflector antenna configuration it is thus possible to consider geometries in which the illuminated scatterer is located in the near field of the source. This is particularly useful when the source pattern is known only in the far field. The field based on the coefficients will automatically be determined by a near-field calculation at a finite distance.

Measured Patterns

A measured pattern can be read from file when the source is specified as a *Tabulated Pattern*. See the details in the description of that class.

When the *Get SWE* command is issued, the field is determined as specified in the *Tabulated Pattern* object. It is possible to suppress the internal spherical expansion (by specifying Far-Field Forced to ‘on’ in the *Tabulated Pattern* object) and it is recommended to specify N phi and N theta (in the attribute *Sphere Sample* of the present class) in accordance with the spacings applied in the measurements in order not to introduce additional interpolation errors. The attributes Near/far (Obsolete) and Sphere Radius (Obsolete) must be specified according to the radius applied in the measurement. See also the section ‘Sampling on the Sphere’ below.

For measured patterns the origin of the source coordinate system must be the point around which the antenna is rotated during the measurement.

Basic concepts of the SWE

The SWE is a general expansion of a radiating field in spherical modes. The coefficients of these modes are the spherical coefficients.

The modes are described by the two indices (m, n) related to the size of the source, and a third index (s) related to the TE and TM properties of the modes.

The integer m is referred to as the azimuthal index and describes the field variations in ϕ by $\sin m\phi$ and $\cos m\phi$ terms. Similarly, the polar index n describes the field variations in θ as $\sin n\theta$ and $\cos n\theta$ terms.

The polar index n has the range

$$1 \leq n \leq N \quad (1)$$

For modes with polar index n , the azimuthal index m is within the range

$$-n \leq m \leq n. \quad (2)$$

The number of modes are thus limited by N , and N is derived from

$$N = kr_0 + \max(10, 3.6\sqrt[3]{kr_0}) \quad (3)$$

where k is the wavenumber, $2\pi/\lambda$, and r_0 is the radius of the smallest sphere completely enclosing all the sources of the field. The maximum mode index N thus depends on the extend of the radiating sources but also on the position of the centre of the sphere as the radii are measured from that centre.

The centre of the sphere is the origin of the coordinate system of the source specified in the **Get SWE** command. A clever choice of the coordinate system with respect to the extend of the source can thus reduce the number of modes which again reduces the computation time.

Sampling on the Sphere

As mentioned above, the field is evaluated on a sphere in order to calculate the coefficients. This sphere must be concentric with the minimum sphere of radius r_0 and it must surround the source completely. Thus, the radius of the sphere, A , must fulfill

$$A \geq r_0. \quad (4)$$

The spacing of the field points on the sphere must be small enough to represent all spherical modes. In the polar angle the fastest field variation is given by $\sin n\theta$ and $\cos n\theta$ with n being equal to its maximum value, $n = N$. The spacing between sample points must then be

$$\Delta\theta = 180^\circ/N \quad (\text{or less}) \quad (5)$$

with N given by Eq. (3). Therefore the number of sample points along a half circle are (both end points included):

$$N_{\text{theta}} = \frac{180^\circ}{\Delta\theta} + 1 = N + 1. \quad (\text{or larger}) \quad (6)$$

A spacing less than that requested by N given by Eqs. (3) and (5) is only meaningful when the field originates in a measurement in which the dense spacing is applied. In this case the spacing should follow that of the measurements in order not to spread out the measurement inaccuracies by interpolating to a new spacing.

Similarly,

$$\Delta\phi = 180^\circ/N \quad (\text{or less}) \quad (7)$$

for the general source and the number of sample points, N_{phi} along a full azimuthal circle is

$$N_{\text{phi}} = \frac{360^\circ}{\Delta\phi} = 2N \quad (\text{or larger.}) \quad (8)$$

Very often the field variation in ϕ is limited to only a few azimuthal modes and the maximum values for m is limited to M

$$|m| \leq M < N. \quad (9)$$

In this case

$$N \text{ phi} = 2(M + 1). \quad (10)$$

The sample increment in ϕ is then

$$\Delta\phi = 180^\circ/(M + 1). \quad (11)$$

The increment in ϕ may thus be much coarser than in θ when $M \ll N$.

For a pattern fully described by $\sin \phi$ and $\cos \phi$, i.e.

$$M = 1$$

we have

$$N \text{ phi} = 4$$

which means that the pattern may be fully described by the principle-plane patterns

$$\Delta\phi = 90^\circ.$$

It is the values of the specified $N \text{ phi}$ and $N \text{ theta}$ which determines the maximum mode indices M and N , given by

$$M = (N \text{ phi})/2 - 1 = 180^\circ/\Delta\phi - 1, \quad (12)$$

$$N = (N \text{ theta}) - 1 = 180^\circ/\Delta\theta. \quad (13)$$

Summations of the power contained in all modes up to the current m and, respectively, up to the current n , $n \leq N$ are given in the Additional Job Output of the Results pane. These two tables shall be studied carefully. The power of excluded modes will spread out within the determined modes, and the power in the highest involved modes (in n as well as in m) will increase. It is therefore a good check to assure that the contribution to the total power from the highest involved modes is below the desired accuracy threshold. See also the following examples.

It must be noted that a dense angular spacing will result in corresponding high values of M and N . It must be checked that

$$M \leq N \quad (14)$$

is not violated hereby, i.e. by Eqs. (10) and (6)

$$(N \text{ phi})/2 \leq (N \text{ theta}) \quad (15)$$

shall be fulfilled. Furthermore, Eq. (4) must be fulfilled, i.e. by Eqs. (3) and (6) that

$$(N \text{ theta}) - 1 \leq kA + \max(10, 3.6 \sqrt[3]{kr_0}). \quad (16)$$

The term $+\max(..)$ herein may be relaxed with caution, see the discussion in the example below.

There is, however, no reason to apply higher values of N_{phi} and N_{theta} than given by Eqs. (8), (6) and (3) as the space-limited source will not result in modes with higher indices.

For measured patterns a dense spacing may have been chosen e.g. for reducing the measurement noise and N_{theta} may be too high for fulfilling Eq. (16). In such a case the source (in *Get SWE*) must be specified to be a *Tabulated Pattern*. It is here possible to limit the number of modes such that Eq. (16) is fulfilled.

The chosen values of N_{theta} and N_{phi} influence the computing time for determination of the coefficients though this is based on the Fast Fourier Transform algorithm (FFT). This has only importance for very large field sources (antenna structures). The FFT is fastest when $N_{\text{theta}} = 1 (=N)$ as well as N_{phi} can be written as a product of many small prime factors, e.g. $N = 180 = 2 \cdot 2 \cdot 3 \cdot 3 \cdot 5$. Factors above 5 should then be avoided.

Finally, the correct number of expansion coefficients for a source does neither depend on the radius, A , nor on N_{phi} nor N_{theta} .

Example: Dual Mode Conical Horn (Potter Horn)

In the first example we will consider a Potter horn for illumination of a reflector which is in the near field of the horn (see class *Potter Horn* for definitions of the horn). In the default calculation of the horn field this is automatically expanded in spherical modes (the attribute `far_forced` is by default 'off'). But we may also calculate the coefficients directly by *Get SWE*.

We consider a horn with an aperture diameter of $D = 4\lambda$ and a slant flare length of 5λ . The excitation shall be RHC. The far field is shown in Figure 1 below.

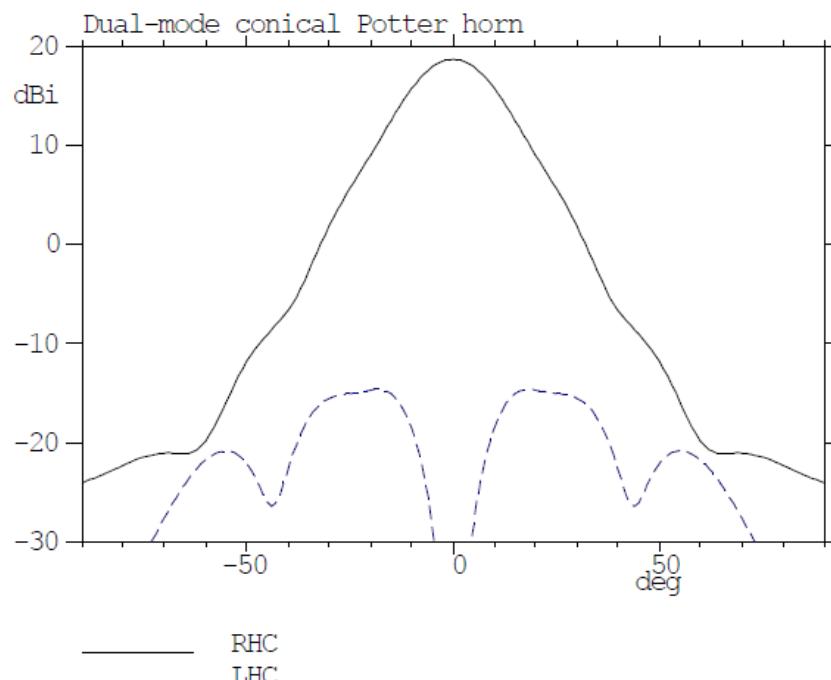


Figure 1

Far-field pattern for the Potter horn.

The origin of the coordinate system for the Potter horn is at the centre of the aperture (as phase_displacement is specified to zero). The smallest sphere surrounding the subreflector thus has the radius

$$r_0 = D/2 = 2\lambda$$

and we have, according to Eq. (3),

$$N = 23,$$

resulting in

$$N \text{ theta} = 24.$$

The excitation and the rotational geometry of the horn ensures a pure $|m| = 1$ radiation field. We therefore get

$$N \text{ phi} = 4.$$

Otherwise, the default values are applied.

A summary of the power in the modes is given in the standard output file, first for m and next for n . The table in m shows in the first column the value of $|m|$ up to M as given by Eq. (12). In the second column the power contained in all modes with this value of $|m|$ is given (summed over n), and in the third column the accumulated power in all modes with this and lower value of $|m|$ is given. The table for n gives correspondingly the power for the n -modes up to $n = N$. The power values in the tables are relative to 4π watts. The total power in the tables shall thus be unity for a source normalised to isotropic level i.e. radiating 4π watts.

The table for m has the following content

Power of spherical wave expansion modes		
Power / (4*pi watts) in each Azimuthal mode		
M	Power	Accumulated
0	2.20096E-28	2.20096E-28
1	0.99950	0.99950

From column two it is seen that all power, as expected, are contained in the modes with $|m| = 1$. The Potter horn is normalised to isotropic level i.e. radiating 4π watts and the total power in the table shall thus be unity. But it is only 0.99950. This is due to an approximation in the power normalisation of the horn.

The table for n varying is

Power / (4*pi watts) in each Polar mode		
N	Power	Accumulated

1	8.36260E-02	8.36260E-02
2	0.10173	0.18536
3	0.13798	0.32333
4	0.17733	0.50066
5	0.15243	0.65309
6	0.13643	0.78952
7	9.16680E-02	0.88119
8	6.11490E-02	0.94234
9	3.36252E-02	0.97596
10	1.47250E-02	0.99069
11	6.44685E-03	0.99714
12	1.64465E-03	0.99878
13	6.01521E-04	0.99938
14	9.01518E-05	0.99947
15	2.89966E-05	0.99950
16	2.67211E-06	0.99950
17	7.83278E-07	0.99950
18	4.62930E-08	0.99950
19	1.27423E-08	0.99950
20	5.34884E-10	0.99950
21	1.35435E-10	0.99950
22	4.29539E-12	0.99950
23	8.53255E-13	0.99950

It is here seen that most of the power is contained for $n \leq kr_0 \cong 13$ which is a theoretical upper limit for n . For increasing n the power of the modes drops continuously and when n has been increased by 10 (as requested by Eq. (3)) the power content of omitted modes has dropped at least seven decades.

The values in the columns 2 and 3 are shown in dB in Figure 2.

If the power content does not decrease drastically for the highest values of m and n , respectively, in the two tables then it is a sign on inaccuracies. These may be due to too low values of N_{θ} or N_{ϕ} for the given source. Thus, Eqs. (3), (14), (15) or (16) shall be checked. The inaccuracies may also be caused by an inaccurate source model such as a discontinuous pattern input or measurements inaccuracies in tabulated patterns.

Example 2, Rectangular Standard Gain Horn

The rectangular horn is a more general source than the conical horn in the first example. The field is thus not restricted to modes with $|m| = 1$.

We will consider a WR-75 rectangular waveguide-fed pyramidal horn with the following dimensions:

Waveguide cross-section $a \times b$: 19.05 × 9.525 mm Aperture cross-section $a \times b$: 94.55 × 67.40 mm Axial length of flared section L : 202 mm frequency f : 10 GHz

With reference to the centre of the aperture we have

$$r_0 = 46.66 \text{ mm} \quad (17)$$

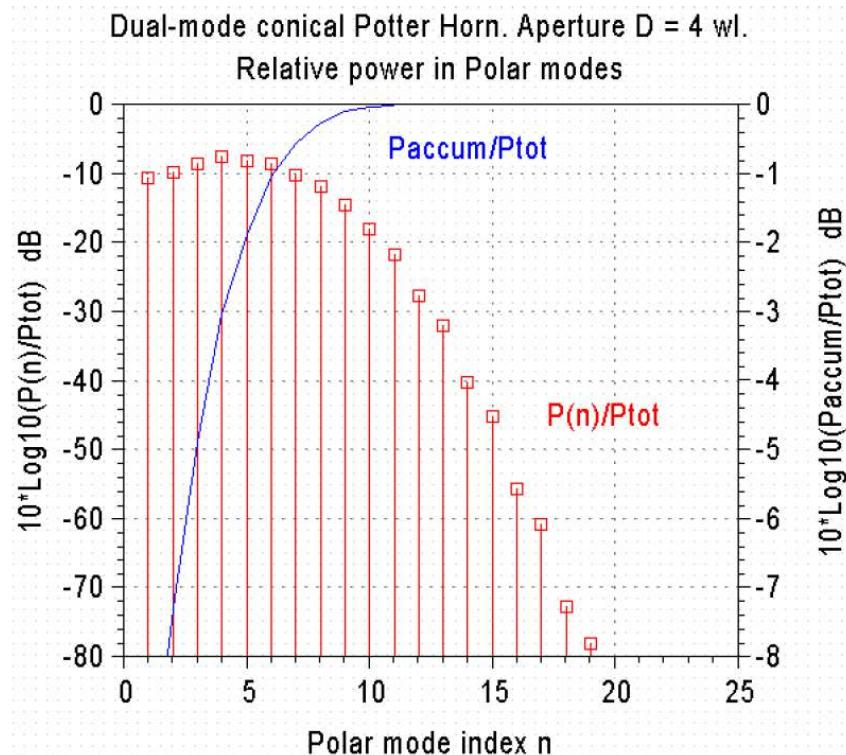


Figure 2 Relative power in the modes as function of the polar index n (red squares, left ordinate scale) and relative power accumulated up to the given n (blue curve, right ordinate scale).

which results in

$$N = kr_0 + 10 \cong 20 \quad (18)$$

To have all modes represented requires a spacing (Eq. (5)) of 9° .

The far-field radiation pattern has been calculated by an external mode-matching program in 32 constant ϕ -cuts ($\Delta\phi = 11.25^\circ$), each cut sampled from $\theta = 0^\circ$ to $\theta = 180^\circ$ (asymmetric cuts, $\Delta\theta = 5^\circ$). The calculated data was stored in a file, Grasp_swe2.cut, which is included in the software delivery. Contents and format are detailed in Section [Field Data in Cuts](#).

As the source is given in a table, an object of class [Tabulated Pattern](#) is used to get the pattern into GRASP. In this object `far_forced` must be specified to 'on'. If it is 'off', which is default, the tabulated-pattern feed will be expanded in spherical wave modes before used in [Spherical Wave Expansion \(SWE\)](#). It is also important that the number of Sphere Sample in [Spherical Wave Expansion \(SWE\)](#) is exactly the same as the number of cuts and cut values in the tabulated file:

```
sphere_sample: struct (n_phi:32, n_theta:37),
```

By the command [Get SWE](#) the spherical modes are determined and the following lists summarise the power content of the m -modes

Power of spherical wave expansion modes

Power / (4*pi watts) in each Azimuthal mode

M	Power	Accumulated
0	4.26504E-12	4.26504E-12
1	0.96814	0.96814
2	3.60694E-12	0.96814
3	1.95264E-02	0.98767
4	3.85903E-13	0.98767
5	1.02395E-02	0.99791
6	1.04589E-13	0.99791
7	1.14874E-03	0.99906
8	2.42789E-14	0.99906
9	6.76267E-04	0.99973
10	8.11241E-15	0.99973
11	9.88034E-05	0.99983
12	4.20051E-16	0.99983
13	2.95137E-07	0.99983
14	5.30826E-17	0.99983
15	1.96474E-07	0.99983

and of the n -modes

Power / (4*pi watts) in each Polar mode

N	Power	Accumulated
1	6.39849E-02	6.39849E-02
2	0.10257	0.16656
3	0.16664	0.33320
4	0.24757	0.58077
5	0.18977	0.77054
6	0.12174	0.89228
7	6.94778E-02	0.96176
8	2.30451E-02	0.98480
9	1.17651E-02	0.99657
10	2.09002E-03	0.99866
11	1.02496E-03	0.99968
12	9.90863E-05	0.99978
13	4.40064E-05	0.99983
14	2.62478E-06	0.99983
15	1.25977E-06	0.99983
16	4.22613E-08	0.99983
17	1.77485E-08	0.99983
18	4.28272E-10	0.99983
19	1.65421E-10	0.99983
20	2.87486E-12	0.99983
21	1.03504E-12	0.99983

22	1.35932E-14	0.99983
23	4.75494E-15	0.99983
24	2.62553E-16	0.99983
25	2.46904E-16	0.99983
26	2.10433E-16	0.99983
27	1.80451E-16	0.99983
28	1.62808E-16	0.99983
29	1.51350E-16	0.99983
30	1.76185E-16	0.99983
31	2.00360E-16	0.99983
32	2.11397E-16	0.99983
33	2.67252E-16	0.99983
34	3.95886E-16	0.99983
35	5.36983E-16	0.99983
36	4.08415E-16	0.99983

Of the last tables it is seen that the n -modes show cut-off around $n = 13$ and modes with n larger than 23 represent noise at a rather constant level.

From the first table with the m -modes it is seen that the ϕ -spacing probably is too coarse as the noise level is not reached here. The accuracy is 'only' about 10^{-7} . It is further seen that the m -modes carrying power are all odd. This is due to the two-fold rotational symmetry of the horn. Almost 97% of the radiated power is contained in the $|m| = 1$ azimuthal mode, while $|m| = 3$ and $|m| = 5$ account for, respectively, 1.95% and 1.02%, leaving 0.2% to the remaining odd modes.

The following two figures give a graphical presentation of the power distributions in azimuthal and polar modes, respectively.

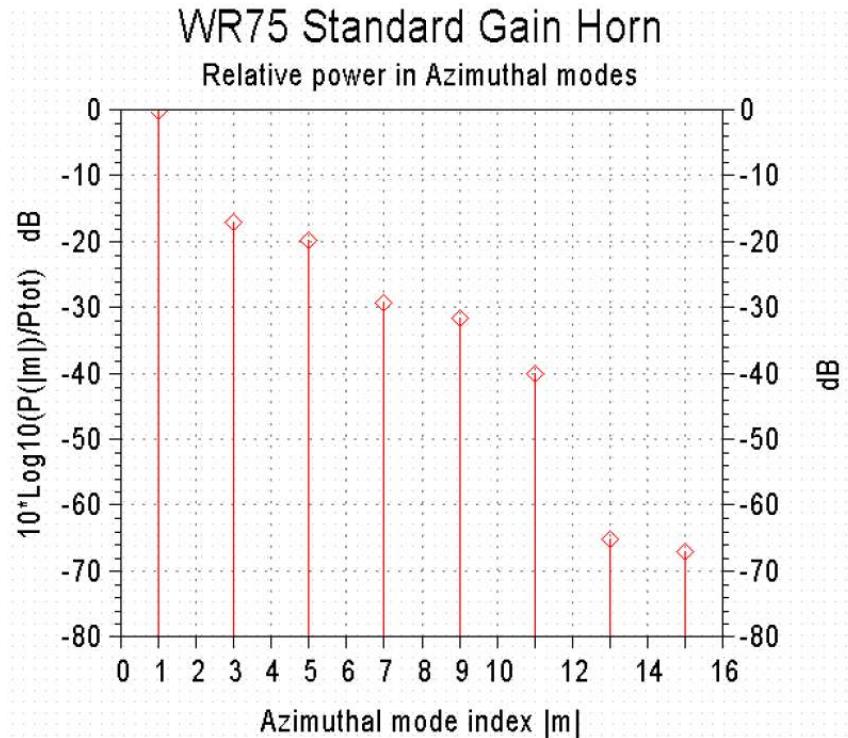


Figure 3 Power distribution in azimuthal modes.

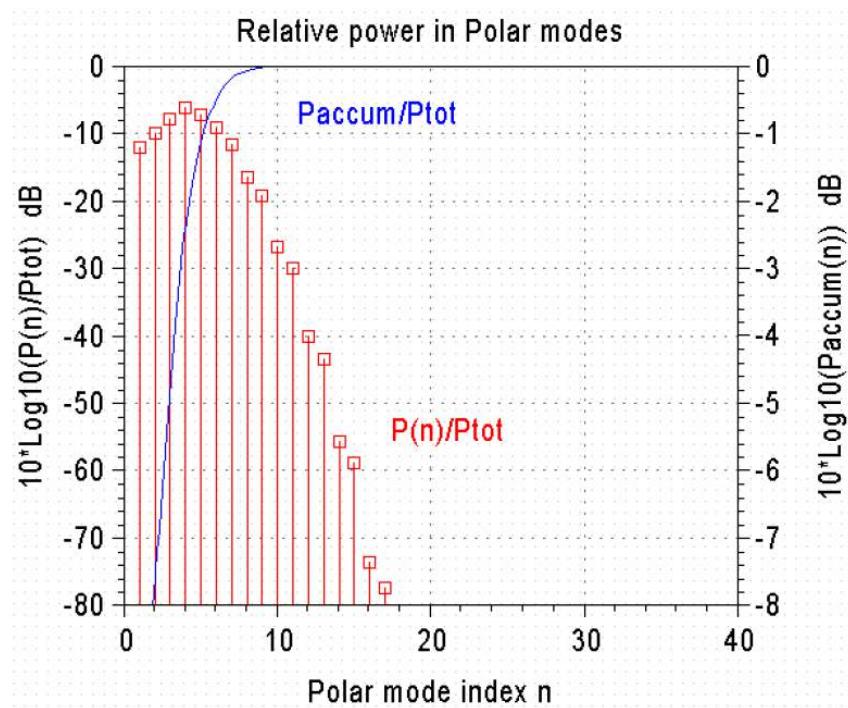


Figure 4

Power distribution in polar modes (red squares, left ordinate scale) and accumulated power (blue curve, right ordinate scale).

ELECTRICAL PROPERTIES DATA OUTPUT (el_prop_data_output)

Purpose

The class *Electrical Properties Data Output* is used to produce a file containing the reflection and transmission coefficients for the surface of a particular reflector.

The coefficients are written as functions of the incidence angles theta and phi, theta being the polar angle and phi the azimuthal angle, with respect to the surface normal, in the same format as when tabulated reflection and transmission coefficients are used to characterise the surface, see the Remarks section.

Links

[Classes](#)→[Electrical Objects](#)→[Derived Electrical Data](#)→[Electrical Properties Data Output](#)

[Remarks](#)

Syntax

```
<object name> el_prop_data_output
(
    frequency           : ref(<n>),
    theta_start         : <r>,
    theta_end           : <r>,
    theta_points        : <i>,
    phi_start           : <r>,
    phi_end             : <r>,
    phi_points          : <i>,
    file_name           : <f>,
    list                : <si>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object.

theta Start (*theta_start*) [real number], default: **0**.

Start value of polar angle θ for grid specification.

theta End (*theta_end*) [real number], default: **0**.

End value of polar angle θ for grid specification. Must not exceed 90°.

theta Points (*theta_points*) [integer], default: **1**.

Number of θ -values in the grid.

phi Start (*phi_start*) [real number], default: **0**.

Start value of azimuthal angle ϕ for grid specification.

phi End (*phi_end*) [real number], default: **0**.

End value of azimuthal angle ϕ for grid specification.

phi Points (*phi_points*) [integer], default: **1**.

Number of ϕ -values in the grid.

File Name (*file_name*) [file name].

Name of the file to which the calculated data are written. The format of the file is described in *Tabulated Electrical Properties for Scatterers*.

List (*list*) [item from a list of character strings], default: **on**.

Generates a list of the data (reflection and transmission coefficients) in the standard output file.

on

A list is generated.

off

No list is generated.

Command Types

The *Electrical Properties Data Output* class is derived from the class *Reflector Data Output* in which the available commands are described.

Remarks

The calculated reflection and transmission data are determined for a planar, infinite structure with the given electrical properties, and the direction angles, θ and ϕ , are related to a reference coordinate system with the z -axis as the normal to the plane.

In the written file, the first line is a header, which will contain the name of the object generating the file. The file can be used as input for *Tabulated Electrical Properties*, (tabulated reflection and transmission coefficients) but only when theta Start and phi Start have been specified to zero.

The file is only written if a file name is specified. If a file name is not specified, the output may be written in the output file by specifying *list* to on.

COUPLING SYSTEM (coupling_system)

Purpose

The class **Coupling System** is used for describing the circumstances under which coupling calculations shall be carried out. The coupling is determined as the coupling quotient between transmitting **Sources** and receiving **Sources**. The transmitters are defined in the command **Get Coupling** activating the coupling calculations while the receivers are defined here in class **Coupling System**.

One of the RF components, receivers and transmitters, may contain **Sources** which are in the near field of the **Sources** of the other component.

The receiving sources may, as an entity, be translated and rotated relatively to the fixed transmitters and the coupling is calculated as function of these movements.

This class is only available with the Coupling add-on.

Links

[Classes](#)→[Electrical Objects](#)→[Derived Electrical Data](#)→[Coupling System](#)

Remarks

Syntax

```
<object name> coupling_system
(
    frequency : ref(<n>),
    receiver_sources : sequence(ref(<n>), ...),
    amplitude_only : <si>,
    movement_definition : ref(<n>),
    list : <si>,
    file_name : <f>,
    file_form : <si>,
    comment : <s>
)
where
<n> = name of an object
<s> = character string
<f> = file name
<si> = item from a list of character strings
```

Attributes

Frequency (*frequency*) [name of an object].

Reference to a **Frequency** object, defining the frequencies at which the coupling calculations shall be performed.

Receiver Sources (*receiver_sources*) [sequence of names of other objects].

A sequence of one or more references to objects of the class *Source*. These objects define the receivers in the coupling analysis. The coupling contributions for all receivers are added according to the attribute *amplitude_only*.

Amplitude Only (*amplitude_only*) [item from a list of character strings], default: **off**.

When more than one transmitter (or receiver) is involved it must be specified how the complex coupling contributions shall be added.

off

The coupling quotients are superimposed in amplitude and phase.

on

The coupling quotients from the different transmitters (receivers) are added in amplitudes giving a maximum coupling estimate. This may be useful when the relative phases between the transmitters (receivers) are uncertain or rapidly varying with the antenna positions.

Note, that also the coupling from or to an antenna consisting of many elements (such as a PO current object consisting of many current elements) will be determined by adding in amplitude the coupling contributions for each element. This may cause a meaningless high coupling value.

Movement Definition (*movement_definition*) [name of an object], default: **blank**.

Reference to an object of class *Movement Definition*, in which possible movements of the receivers are defined.

List (*list*) [item from a list of character strings], default: **off**.

The coupling quotients from all different receivers and transmitters may be listed in the standard output file.

off

No list is generated.

on

The list is generated.

File Name (*file_name*) [file name].

Name of file in which the coupling quotients are stored. If a file name is not given, the coupling values will not be stored. The content of the file is described in *Coupling Data*. The recommended file extensions *.cut* or *.grd* according to the specified *file_form* (see below).

File Form (*file_form*) [item from a list of character strings], default: **cuts**.

The coupling quotients may be written in different forms to the file.
cuts

The coupling quotients are written as separate cuts in the file, each cut containing values for the fastest varying movement specified in the *Movement Definition*, and one cut for each of the steps in the slower varying movements specified. See also the remarks to *Movement Definition*.

grid

The coupling quotients are written as grids in the file, each grid containing values for the two fastest varying movement specified in the *Movement Definition*, and one grid for each of the steps in the slower varying movements specified. See also the remarks to *Movement Definition*.

one_cut

All the calculated coupling quotients are written as one single cut in the sequence specified in the *Movement Definition* object. This feature is useful in cases such as determining the coupling in a fixed geometry for a range of frequencies whereby a frequency sweep is generated in a cut.

Comment (*comment*) [character string].

A line with this text will be written as a header in the file specified by *file_name* above.

Command Types

The coupling calculations are activated by the command

Get Coupling.

Remarks

It shall be noted that the transmitters are fixed while the receivers move. If the case to be considered is for transmitters moving relatively to fixed receivers then the transmitters and the receivers may be interchanged as the coupling value is independent on the direction of transmission.

For the movements it is always important to keep track on the coordinate systems in which the receivers are defined, not least which coordinate system are defined in which. Especially for the current sources (the objects defined under *PO Analysis*) it shall be born in mind that the coordinate system of the currents are defined in the appropriate *PO Analysis* object.

COUPLING (OBSOLETE) (coupling)

Purpose

The class **Coupling (Obsolete)** is used for describing the circumstances under which coupling calculations shall be carried out. The coupling is determined as the coupling quotient between a transmitter and a receiver being in the far field of each other. Both the transmitter and the receiver are object of the general class **Source**.

The class is included for backwards compatibility only. Instead it is recommended to use the more versatile class **Coupling System**.

The receiver may be translated and rotated relatively to the transmitter. Only one translation may be specified in this class.

This class is only available with the Coupling add-on.

Links

[Classes](#)→[Electrical Objects](#)→[Derived Electrical Data](#)→[Coupling \(Obsolete\)](#)

Remarks

Syntax

```
<object name> coupling
(
    frequency : ref(<n>),
    receiver : ref(<n>),
    amplitude_only : <si>,
    list : <si>,
    file_name : <f>,
    comment : <s>,
    rec_move_number : <i>,
    combined : <si>,
    rec_move_coor_sys : ref(<n>),
    rec_trans_x_range : struct(start:<rl>, end:<rl>),
    rec_trans_y_range : struct(start:<rl>, end:<rl>),
    rec_trans_z_range : struct(start:<rl>, end:<rl>),
    rec_rotations : sequence(
        struct(around:<si>,
               start:<r>,
               end:<r>,
               n_angles:<i>),
        ...
    )
)
```

where

<i> = integer

<n> = name of an object

<r> = real number

<rl> = real number with unit of length

<s> = character string

`<f>` = file name

`<si>` = item from a list of character strings

Attributes

Frequency (*frequency*) [name of an object].

Reference to a *Frequency* object, defining the frequencies at which the feed operates.

Receiver (*receiver*) [name of an object].

Reference to an object of one of the classes *Source*, which is used as receiver in the coupling analysis.

Amplitude Only (*amplitude_only*) [item from a list of character strings], default: **off**.

The coupling quotients from the different transmitters may be summed up using the amplitude-only value of the individual quotients giving a worst case estimate.

off

The coupling quotients are superimposed in amplitude and phase.

on

The amplitude-only value is used.

List (*list*) [item from a list of character strings], default: **off**.

The coupling quotients from all transmitters to each of the receivers may be listed in the standard output file.

off

No list is generated.

on

The list is generated.

File Name (*file_name*) [file name].

Name of the file on which the coupling quotients are stored. The content of the file is described in Section *Coupling Data*. The recommended file extension is *.cut*.

Comment (*comment*) [character string].

A line of text which will be written as a header in the file specified by *file_name* above.

Rec Move Number (*rec_move_number*) [integer], default: **0**.

The receiver may be moved in one translation and in rotations. The number of positions in the translation is given by Rec Move Number. If the translation and the rotations are carried out simultaneously (combined movements), the number of positions is also specified by Rec Move Number. If the movements only include rotations then the numbers of angular positions are specified in the respective members of the structs in attribute Rec Rotations below.

Combined (*combined*) [item from a list of character strings], default: **off**.

The translation and rotations defined in the following may be carried out in combination (simultaneously) or sequentially.

off

The translation and rotations are carried out sequentially. The number of positions along the translation is given by Rec Move Number and the number of angles in the rotations is specified in attribute Rec Rotations below. The total number of positions of the receiver in the movements is then given by the number of positions in the translation times the numbers of angles in each of the specified rotations.

on

The translation and rotations are combined. There are thus Rec Move Number steps of combined (simultaneous) translation and rotations.

Rec Move Coor Sys (*rec_move_coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes defining the coordinate system to which the receiver movements are referred.

Rec Trans x Range (*rec_trans_x_range*) [struct].

Definition of the range along the *x*-axis of the receiver translation.

Start (*start*) [real number with unit of length], default: **0**.

Start value for *x* in the translation.

End (*end*) [real number with unit of length], default: **0**.

End value for *x* in the translation.

Rec Trans y Range (*rec_trans_y_range*) [struct].

Definition of the range along the *y*-axis of the receiver translation.

Start (*start*) [real number with unit of length], default: **0**.

Start value for *y* in the translation.

End (*end*) [real number with unit of length], default: **0**.

End value for *y* in the translation.

Rec Trans z Range (*rec_trans_z_range*) [struct].

Definition of the range along the *z*-axis of the receiver translation.

Start (*start*) [real number with unit of length], default: **0**.

Start value for *z* in the translation.

End (*end*) [real number with unit of length], default: **0**.

End value for *z* in the translation.

Rec Rotations (*rec_rotations*) [sequence of structs].

Each struct in this sequence defines a rotation in the movement coordinate system in the same order as the definitions such that the last defined rotation is the fastest. The rotations are specified by the struct member `around`, and the angles of rotation are specified by `start`, `end` and `n_angles` as follows:

Around (*around*) [item from a list of character strings], default: **x_original**.

Alphanumeric identification string of the axis of rotation. It is possible to rotate around 'original' axes as well as around 'rotated' axes. The 'original' axes are axes parallel to those of the nonrotated Rec Move Coor Sys coordinate system. If a translation has been specified then the axis of rotation is through the origin of the translated coordinate system. The 'rotated' axes are the axes of a coordinate system which has followed the previous movements.

x_original

The rotation is around the 'original' *x*-axis, positive from the *y*-axis towards the *z*-axis.

y_original

The rotation is around the 'original' *y*-axis, positive from the *z*-axis towards the *x*-axis.

z_original

The rotation is around the 'original' *z*-axis, positive from the *x*-axis towards the *y*-axis.

x_rotated

The rotation is around the new moved *x*-axis, positive from the *y*-axis towards the *z*-axis.

y_rotated

The rotation is around the new moved *y*-axis, positive from the *z*-axis towards the *x*-axis.

z_rotated

The rotation is around the new moved *z*-axis, positive from the *x*-axis towards the *y*-axis.

Start (*start*) [real number], default: **0**.

Start value of the rotation in degrees.

End (*end*) [real number], default: **0**.

End value of the rotation in degrees.

N Angles (*n_angles*) [integer], default: **0**.

Number of angular values in the rotation.

Command Types

The coupling calculations are activated by the command

Get Coupling.

Remarks

The set-up for the coupling between a transmitter and a receiver is defined by an object of this class. The transmitter as well as the receiver is defined as a *Source*. For a fixed geometry only one coupling value is output but it is often affordable to move the receiver to see the resulting variation in the coupling. The coupling is then determined as function of the movement and the values may be presented in a diagram (e.g. by the postprocessor) with the movement coordinate as abscissa and the coupling as ordinate.

It shall be noted that the transmitter is fixed while the receiver moves. If the case to be considered is for a transmitter moving relatively to a fixed receiver then the transmitter and the receiver may be interchanged as the coupling value is independent on the direction of transmission.

This section contains discussions on the following topics:

- Limitations
- Movements
- Translation only
- Rotation only
- Combined translation and rotation
- The positions of the receiver in the movements

Limitations

Due to computational constrains only feeds, feed arrays and currents can act as receiver but all kind of sources can be used as transmitters. This is not a severe restriction since the receiver and transmitter may be reversed in the coupling analysis.

Movements

The movements of the receiver consist of two types, translation and rotations, which are specified differently and which may be performed individually or in combinations. Only one translation may be specified.

Translation only

A translation is specified along the axes of the coordinate system referred to by the attribute Rec Move Coor Sys. The translation is a linear motion and the coupling is calculated in a number of positions as specified by Rec Move Number. Only one translation is possible.

As an example consider the coupling between two horns of which the receiving horn is moving along a line from $(x, y, z) = (1, 1, 0)$ to $(x, y, z) = (9, 5, 0)$ in cm in the global coordinate system (default value for Rec Move Coor

Sys). The coupling is desired at 51 points and the specifications for the movements shall then be

```
rec_move_number: 51,
combined: off,      (default value)
rec_trans_x_range: struct(start:1.0 cm, end:9.0 cm),
rec_trans_y_range: struct(start:1.0 cm, end:5.0 cm),
rec_trans_z_range: struct(start:0 cm, end:0 cm),
rec_rotations: sequence(struct(around:x_original, start:0, end:0,
n_angles:0))    (default values)
```

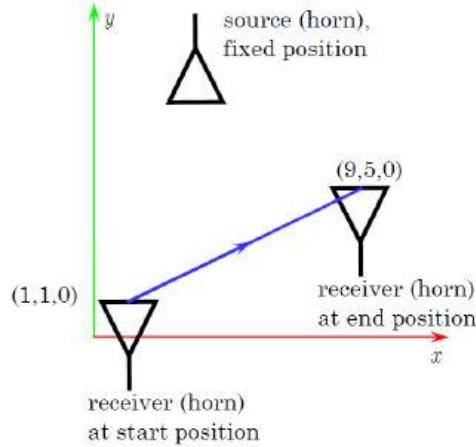


Figure 1 Translation of the receiver.

Rotation only

Let us in the next example consider rotations. A typical case may be an azimuth-over-elevation rotation with scan in azimuth (this is around the vertical *y*-axis, the azimuth axis) and step in elevation (around the horizontal *x*-axis, the elevation axis), see the following figure. As the set-up is an azimuth-over-elevation rotation, the azimuth rotation table is mounted upon the elevation table and the elevation rotation will thus rotate the azimuth axis (the *z*-axis) around the elevation axis, the *y*-axis.

Let the azimuth scan be $\pm 36^\circ$ in steps of 1° and the step values be -20° , 0° and $+20^\circ$.

The following specifications shall be given here:

```
rec_move_number:0,    (default value)
combined:off,      (default value)
rec_trans_x_range: struct(start:0 m, end:0 m),   (default value)
rec_trans_y_range: struct(start:0 m, end:0 m),   (default value)
rec_trans_z_range: struct(start:0 m, end:0 m),   (default value)
rec_rotations: sequence(struct
    (around:y_original, start:-20, end:20, n_angles:3),
    (around:z_rotated, start:-36, end:36, n_angles:73) )
```

As the latest specified rotation is the fastest we must start with the stepped rotation in elevation. This is the first struct in Rec Rotations and it is

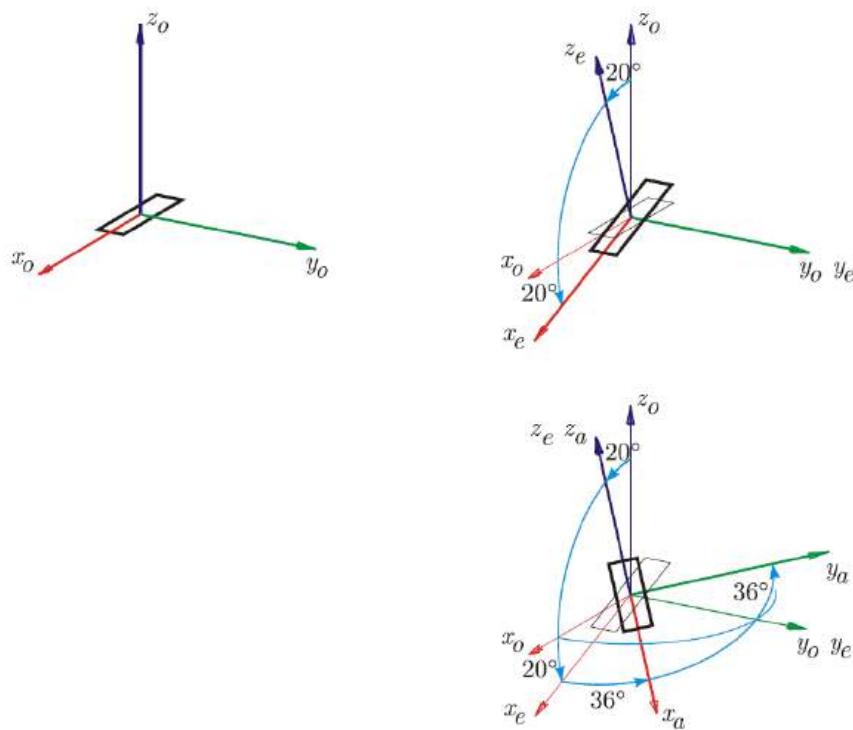


Figure 2

Rotation of the receiver. The receiver is illustrated as an aperture in the xy -plane of the non-rotated system (a). By an elevation rotation of $+20^\circ$ around the y -axis, the aperture plane tilts to the $x_e y_e$ -plane and the azimuth axis tilts to the z_e -axis (b). By the following azimuth rotation of 36° around the z_e -axis, the aperture rotates with the $x_a y_a$ -axes (c).

around the original y -axis. Next we specify the azimuth scan in the second struct. This rotation is around the new rotated z -axis (the z_e -axis in the figure).

Combined translation and rotation

Then we will consider an example on a combined translation and rotation movement. The case could be a rotating antenna upon a ship sailing along a straight line. The linear movement of the ship, the translation, is along the same line as in the example for translation only. Along this path the antenna rotates 90° around the z -axis.

The specifications are:

```
rec_move_number: 91,
combined: on,
rec_trans_x_range: struct(start:1.0 cm, end:9.0 cm),
rec_trans_y_range: struct(start:1.0 cm, end:5.0 cm),
rec_trans_z_range: struct(start:0 cm, end:0 cm),
rec_rotations: sequence(struct(around:z_original,
    start:0, end:90, n_angles:1))
```

As the movements are combined, the attribute combined is 'on' and the

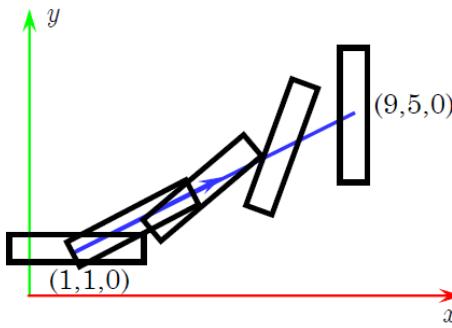


Figure 3

A combined translation and rotation. A rotating antenna (black) is mounted upon a ship sailing along the blue line. At the start position, the antenna is parallel to the x -axis.

number of positions is given by a non-zero value of Rec Move Number. The translation from $(x, y, z) = (1, 1, 0)$ to $(x, y, z) = (9, 5, 0)$ and the rotation are finally specified. Note that the number of angular values is dictated by Rec Move Number as the movements are combined; the value of n_angles is dummy.

The positions of the receiver in the movements

The movements may consist in a translation followed by some rotations. The translation is given as a linear movement from a start position (x_s, y_s, z_s) to an end position (x_e, y_e, z_e) with a total of N_t (rec_move_number) positions:

$$(x_i, y_i, z_i) = (x_s, y_s, z_s) + (i - 1)(\Delta x, \Delta y, \Delta z), \quad i = 1, 2, \dots, N_t$$

where the step in the translation is

$$(\Delta x, \Delta y, \Delta z) = \frac{(x_e - x_s, y_e - y_s, z_e - z_s)}{N_t - 1}$$

If *rec_move_number* is specified to 1 then the receiver is fixed at

$$(x_1, y_1, z_1) = (x_s, y_s, z_s)$$

and if *rec_move_number* is specified to 0 then the receiver is fixed at

$$(x_1, y_1, z_1) = (0, 0, 0)$$

The rotations are defined in a similar way, the number of each rotation given by *n_angles*.

When the movements are sequentially (*combined* is set to 'off'), then the coupling ratios are calculated in the following order (and stored on the file in this order, i.e. *i* being the record number exclusive headings):

$$i = (i_t - 1) \prod_{r=1}^R N_r + (i_1 - 1) \prod_{r=2}^R N_r + \dots + (i_{R-1} - 1) N_R + i_R$$

Where

- i_t is the index of the translation $1 \leq i_t \leq N_t$

- R is the number of different rotations
- r is the index of the rotations $1 \leq r \leq R$
- N_r is the number of positions in rotation r
- i_r is the angular index of rotation r , $1 \leq i_r \leq N_r$

The order of the rotations as given by r is the same as the order of the rotation definitions in the sequence of structs. Thus, the first coupling value, $i = 1$, is for

$$(i_t, i_1, i_2, \dots, i_{R-1}, i_R) = (1, 1, 1, \dots, 1, 1)$$

then index i_R increases 1 by 1 until $i_R = N_R$. Next, i_R is reset to 1 and index i_{R-1} counts 1 up, etc.

VARIABLES AND EXT. COMMANDS

Purpose

This group contains classes used to define variables and external commands.
A variable is defined by the class

Real Variable

An interface to an external command, e.g. a user-provided script or executable, is defined by the class

External Command

Links

Classes→*Variables and Ext. Commands*

REAL VARIABLE (real_variable)

Purpose

By the class *Real Variable* it is possible to define a real variable by name and value.

Links

[Classes](#)→[Variables and Ext. Commands](#)→[Real Variable](#)

Remarks

Syntax

```
<object name> real_variable
(
    value
)
where
<r> = real number
```

Attributes

Value (*value*) [real number].

The value of the real variable, given as a real number without a unit (the unit appears from the context in which the variable is applied).

Remarks

An object of class *Real Variable* may be referred to from other objects wherever a real number shall be given such as in the specification of an attribute. The value of the variable will then be inserted and applied. This is very useful when geometries which depends on other geometrical parameters shall be expressed.

The value of a *Real Variable* does not depend on the chosen unit. If the unit for a *Real Variable* is changed then the value of the *Real Variable* is unchanged.

A *Real Variable* may also be specified for an attribute which shall have an integer value but it will be handled as a floating point number in expressions. If an attribute which shall have an integer value is given by a non-integer expression this will be truncated to an integer when the value is assigned to the attribute.

The name of a *Real Variable* is case insensitive, i.e. 'R_out' and 'r_out' refer to the same variable.

In the GUI

A *Real Variable* may be defined in the OBJECTS tab through the CREATE menu. The Object Editor opens and the suggested name (real_variable) for the variable should be changed to a better name. Further, the value of the variable shall be given.

Available expressions

Variable may be applied in algebraic and trigonometric expressions with the following operators:

+	addition
-	subtraction
*	multiplication
/	division
^ or **	raising to power

the following functions (angles to be given in radians, for degrees see the tip below):

sqrt()	square root
sin()	sine
cos()	cosine
tan()	tangent
exp()	exponential
log()	logarithmic (base e)
log10()	logarithmic (base 10)
abs()	absolute value
aint()	truncate (round towards zero)
anint()	round to nearest integer
asin()	inverse sine
acos()	inverse cosine
atan()	inverse tangent
sinh()	hyperbolic sine
cosh()	hyperbolic cosine
tanh()	hyperbolic tangent
min(a;b)	minimum value of a and b is returned
max(a;b)	maximum value of a and b is returned

and any level of parenthesis.

Examples on expressions are

var1*(var2 + var3)

and

2*cos(var1/50)

where var1, var2 and var3 are defined as *Real Variable*.

TIP: Transformation from radians to degrees and vice versa can be carried out by multiplying:

angles in radians by RTD (Radians To Degrees) and
angles in degrees by DTR (Degrees To Radians)

where RTD and DTR are *Real Variable* to be defined in the project as

RTD = 45./atan(1.)

DTR = atan(1.)/45.

In the tor-file

When operating on the tor-file it is the name of the object (<object name> in the syntax) which corresponds to the name of a variable in usual mathematical context. The attribute `Value` specifies the value of the variable.

Let the name of the variable be `x` then the variable may be referred to by inserting "ref(`x`)" (including the double quotes) instead of the real number. When an expression of variables is used it is the full expression which shall be enclosed in double quotes. Examples on expressions are

"ref(var1)*(ref(var2) + ref(var3))"

and

"2*cos(ref(var1)/50)"

where `var1`, `var2` and `var3` are defined as *Real Variable*.

Example with a tor-file

A square horn may be defined as a feed of class *Rectangular Horn*. The geometry of a rectangular horn is specified by its width (`aperture_width`) and height (`aperture_height`) together with its flare lengths (`flare_length_xz` and `flare_length_yz`) which all shall be specified as real numbers with unit of length. In addition, a reference to a frequency object `freq` defining the frequency is added.

Thus the rectangular horn may be specified as:

```
horn_example rectangular_horn
(
    frequency : ref(freq),
    aperture_width : 50.0 mm,
    aperture_height : 50.0 mm,
    flare_length_xz : 100.0 mm,
    flare_length_yz : 100.0 mm,
)
```

If the user defines a *Real Variable* with the name `width` and the value 50.0 by

```
width real_variable
(
    value : 50.0
)
```

this variable may be used in the specification of the horn. Thus when `width` is defined as above it may be inserted instead of the number '50.0' in the specification of the `aperture_width`, and if the user wants to ensure that the horn aperture is a square then also the `aperture_height` shall be specified to `width`. If, further, the flare lengths shall be twice the aperture width this may also be specified:

```
horn_example rectangular_horn
(
    frequency : ref(freq),
    aperture_width : "ref(width)" mm,
    aperture_height : "ref(width)" mm,
    flare_length_xz : "2.*ref(width)" mm,
    flare_length_yz : "2.*ref(width)" mm,
)
```

Note that the attributes, as requested, are given by real numbers with unit of length, the real number is the value of `Value` in the referred object `width` and the unit is specified to `mm` (millimeter) after the value.

Another example on the application of expressions for a variable may be found in the table defining the nodes in a *BoR Mesh*.

EXTERNAL COMMAND (external_command)

Purpose

The class *External Command* defines an interface with external commands to be executed by the operating system. This can be used to integrate utilities or other tools, e.g., a script or an executable supplied by the user. The command may be executed with a sequence of user-configurable command line arguments, including strings, numbers, variables, and expressions. The command will be invoked when the object is created or modified, as well as whenever the value of a variable included in the argument list is changed. This class allows the user to define a plugin that creates input to GRASP. For example, an external command can be used to produce a file defining a complex geometry, such as a *Tabulated Rim* or a *Tabulated Mesh*. The geometry will then be automatically updated and reimported whenever a variable used to construct the command line arguments is changed.

Links

[Classes](#)→[Variables and Ext. Commands](#)→[External Command](#)

Remarks

Syntax

```

<object name> external_command
(
    command                  : <f>,
    command_arguments        : sequence(
                                struct(string:<s>,
                                       expression:<r>,
                                       arg:<si>),
                                ...),
    exit_code                : <si>,
    disable_execution         : <si>
)
where
<r> = real number
<s> = character string
<f> = file name
<si> = item from a list of character strings

```

Attributes

Command Name (*command*) [file name].

Name of a script or an executable supplied by the user. A list of command line arguments may be specified below. The command will be executed when the object is created or modified, or whenever a real variable included in one of the command line arguments is changed.

Arguments (*command_arguments*) [sequence of structs].

String (*string*) [character string].

String to be included in the argument.

Expression (*expression*) [real number].

Expression containing variables or a number.

Argument Type (*arg*) [item from a list of character strings], default:
string_and_expression.

This setting determines whether each command line argument consists of a string, an expression, or both.

string_and_expression

The argument will consist of a string followed by an expression. The expression may contain one or more real variables, or simply a real number.

string

The argument will consist of a string. The expression argument is not used.

expression

The argument will consist of an expression containing one or more variables, or simply a real number. The string argument is not used.

Exit Code (*exit_code*) [item from a list of character strings], default: **zero_on_success**.

Determines how the exit code is handled:

zero_on_success

The command is assumed to return a zero exit code upon successfull execution. If a non-zero exit code is detected, an error will be issued.

positive_on_success

The command is assumed to return a non-negative exit code upon succesfull execution. If a negative exit code is detected, an error will be issued.

ignore

The exit code will be ignored.

Disable Execution (*disable_execution*) [item from a list of character strings], default: **no**.

Determines if the command is disabled:

no

The command is enabled and will be executed when needed.

yes

The command is disabled and will never be executed.

Command Types

The external command can be executed by the command

Execute External Command.

Remarks

An object of class *External Command* defines the name and command line arguments of an external command to be executed by the operating system. The external command will be executed at each of the following events:

- When the object of class *External Command* is created or modified.
- When the value of a *Real Variable*, included in the list of command line arguments, changes.
- When a job is submitted.
- During job execution, if a command of type *Execute External Command* is included in the command list.

Command Line Arguments: An arbitrary number of command line arguments can be defined by the *Arguments* parameter. The list of arguments is constructed sequentially and each argument may consist of a string, an expression that evaluates to a real number, or both. For each argument, the user must specify whether the argument is a string, an expression, or a string followed by an expression. The expression may contain the name of an object of class *Real Variable*, see this class for a description of the available expressions.

An example list of command line arguments is illustrated in Figure 1. Assume that a command *MyHelix* produces a mesh of a helix antenna and that the first argument is the output filename, followed by a number of command line parameters. Further, assume that *r*, *pitch*, and *wr* are objects of class *Real Variable* and attain the values 0.1, 1, and 0.001, respectively. The command produced by the example argument list is then:

```
MyHelix helix.msh --radius=0.1 --pitch=1 --turns=10.0 --wire-radius=0.001
```

	String	Expression	Argument Type
1	helix.msh	0,0	string
2	--radius=	r	string_and_expression
3	--pitch	pitch	string_and_expression
4	--turns=	10,0	string_and_expression
5	--wire-radius=	wr	string_and_expression

Figure 1 An example arguments list.

Creating a Plugin for Defining Complex Geometry: The class *External Command* is useful for creating a plugin that provides a seamless integration with an external utility. For example, an external script can produce an input file which is understood by an object of class *Tabulated Rim* or *Tabulated Mesh*. The plugin is then created by connecting the object that reads

the input file with the object of class *External Command* that executes the script (consult the *Tabulated Rim* or *Tabulated Mesh* class descriptions for a description of the attribute that connects to the object of class *External Command*). This connection ensures that the object reading the input file is notified whenever the external command is executed. As a consequence, the 3D drawing will update automatically when the external script is executed, e.g., due to a change of a *Real Variable* used in a command line argument.

QUAST FRAME

Purpose

The only class available here is the class

Frame

which is used to specify the necessary attributes for a frame. These are autogenerated by the GUI and it is recommended that the frame is specified in that way.

Links

Classes→*Quast frame*

FRAME (frame)

Purpose

The purpose of class **Frame** is to specify the necessary attributes for a frame. These are autogenerated by the GUI and it is recommended that the frame attributes are generated in that way. The syntax is thus given but the function of the attributes is not explained.

Links

Classes→*Quast frame*→**Frame**

Remarks

Syntax

```
<object name> frame
(
    coor_sys           : ref(<n>),
    no_of_components  : <i>,
    frame_unit         : <si>,
    size               : struct(x_min:<r>, x_max:<r>, y_min:<r>,
                                y_max:<r>),
    grid               : struct(show:<si>, snap_to:<si>,
                                x_spacing:<r>, y_spacing:<r>),
    connection         : <si>,
    feeds              : sequence(
                            struct(name:<s>,
                                   index:<i>,
                                   center_x:<r>,
                                   center_y:<r>,
                                   orientation:<r>,
                                   beam1_to_component:<i>,
                                   beam1_to_port:<i>,
                                   beam_radius:<r>,
                                   phase_curvature:<r>,
                                   polarization_angle:<r>,
                                   wavelength:<r>),
                            ...),
    loads              : sequence(
                            struct(name:<s>,
                                   index:<i>,
                                   center_x:<r>,
                                   center_y:<r>,
                                   orientation:<r>,
                                   beam1_to_component:<i>,
                                   beam1_to_port:<i>,
                                   load_size_x:<r>,
                                   load_size_y:<r>),
                            ...),
```

```
lenses : sequence(
    struct(name:<s>,
           index:<i>,
           center_x:<r>,
           center_y:<r>,
           orientation:<r>,
           beam1_to_component:<i>,
           beam1_to_port:<i>,
           beam2_to_component:<i>,
           beam2_to_port:<i>,
           f_value:<r>,
           refractive_index:<r>,
           diameter:<r>,
           curvature_difference:<r>,
           bs1:<r>,
           bs2:<r>),
    ...),
plane_mirrors : sequence(
    struct(name:<s>,
           index:<i>,
           center_x:<r>,
           center_y:<r>,
           orientation:<r>,
           beam1_to_component:<i>,
           beam1_to_port:<i>,
           beam2_to_component:<i>,
           beam2_to_port:<i>,
           height:<r>,
           rim:<si>),
    ...),
reflectors : sequence(
    struct(name:<s>,
           index:<i>,
           center_x:<r>,
           center_y:<r>,
           orientation:<r>,
           beam1_to_component:<i>,
           beam1_to_port:<i>,
           beam2_to_component:<i>,
           beam2_to_port:<i>,
           f_value:<r>,
           curvature_difference:<r>,
           half_cone_angle:<r>,
           cone_axis_angle:<r>,
           cone_vertex_focal_point:<si>),
    ...),
```

```

beam_splitters : sequence(
    struct(name:<s>,
           index:<i>,
           center_x:<r>,
           center_y:<r>,
           orientation:<r>,
           beam1_to_component:<i>,
           beam1_to_port:<i>,
           beam2_to_component:<i>,
           beam2_to_port:<i>,
           beam3_to_component:<i>,
           beam3_to_port:<i>,
           beam4_to_component:<i>,
           beam4_to_port:<i>,
           type:<si>,
           frequency1_lower:<r>,
           frequency1_upper:<r>,
           frequency2_lower:<r>,
           frequency2_upper:<r>,
           reflection1:<r>,
           transmission1:<r>,
           reflection2:<r>,
           transmission2:<r>,
           grid_type:<si>,
           direction:<r>,
           distance:<r>,
           width:<r>,
           height:<r>,
           rim:<si>),
    ...),
mich_interfers : sequence(
    struct(name:<s>,
           index:<i>,
           center_x:<r>,
           center_y:<r>,
           orientation:<r>,
           beam1_to_component:<i>,
           beam1_to_port:<i>,
           beam2_to_component:<i>,
           beam2_to_port:<i>,
           diameter:<r>,
           l_1:<r>,
           l_2:<r>),
    ...),

```

```
marpup_interfers           : sequence(
                                struct(name:<s>,
                                       index:<i>,
                                       center_x:<r>,
                                       center_y:<r>,
                                       orientation:<r>,
                                       beam1_to_component:<i>,
                                       beam1_to_port:<i>,
                                       beam2_to_component:<i>,
                                       beam2_to_port:<i>,
                                       beam3_to_component:<i>,
                                       beam3_to_port:<i>,
                                       beam4_to_component:<i>,
                                       beam4_to_port:<i>,
                                       size_lac:<r>,
                                       size_lbc:<r>,
                                       side_length:<r>,
                                       grid_spacing_pga:<r>,
                                       grid_width_pga:<r>,
                                       grid_spacing_pgb:<r>,
                                       grid_width_pgb:<r>,
                                       grid_spacing_pgc:<r>,
                                       grid_width_pgc:<r>,
                                       l_rtma:<r>,
                                       l_rtmb:<r>,
                                       rim:<si>,
                                       grid_type_pga:<si>,
                                       grid_type_pgb:<si>,
                                       grid_type_pgc:<si>,
                                       grid_orientation_pga:<si>,
                                       grid_orientation_pgb:<si>,
                                       grid_orientation_pgc:<si>,
                                       roof_orientation_rtma:<si>,
                                       roof_orientation_rtmb:<si>),
                                ...),
connectors                 : sequence(
                                struct(name:<s>,
                                       index:<i>,
                                       center_x:<r>,
                                       center_y:<r>,
                                       orientation:<r>,
                                       beam1_to_component:<i>,
                                       beam1_to_port:<i>,
                                       connected_frame:ref(<n>),
                                       adjoining_connector:<s>,
                                       connector_angle:<r>,
                                       conn_coor_sys:ref(<n>)),
                                ...),
```

```

aperture_in_screen           : sequence(
                                struct(name:<s>,
                                       index:<i>,
                                       center_x:<r>,
                                       center_y:<r>,
                                       orientation:<r>,
                                       beam1_to_component:<i>,
                                       beam1_to_port:<i>,
                                       beam2_to_component:<i>,
                                       beam2_to_port:<i>,
                                       aperture_radius:<r>),
                                ...),
plane_waves                  : sequence(
                                struct(name:<s>,
                                       index:<i>,
                                       center_x:<r>,
                                       center_y:<r>,
                                       orientation:<r>,
                                       beam1_to_component:<i>,
                                       beam1_to_port:<i>,
                                       polarisation_angle:<r>,
                                       wavelength:<r>,
                                       aperture_radius:<r>),
                                ...),
near_fields                  : sequence(
                                struct(name:<s>,
                                       index:<i>,
                                       center_x:<r>,
                                       center_y:<r>,
                                       orientation:<r>,
                                       sampling_type:<si>,
                                       polarisation:<si>,
                                       number_of_cuts:<i>,
                                       np:<i>,
                                       rho_end:<r>,
                                       width:<r>),
                                ...),
far_fields                   : sequence(
                                struct(name:<s>,
                                       index:<i>,
                                       center_x:<r>,
                                       center_y:<r>,
                                       orientation:<r>,
                                       beam1_to_component:<i>,
                                       beam1_to_port:<i>,
                                       sampling_type:<si>,
                                       polarisation:<si>,
                                       number_of_cuts:<i>,
                                       np:<i>,
                                       theta_end:<r>,
                                       width:<r>),
                                ...),

```

```

rooftop_interfers          : sequence(
                           struct(name:<s>,
                                  index:<i>,
                                  center_x:<r>,
                                  center_y:<r>,
                                  orientation:<r>,
                                  beam1_to_component:<i>,
                                  beam1_to_port:<i>,
                                  beam2_to_component:<i>,
                                  beam2_to_port:<i>,
                                  side_length:<r>,
                                  grid_spacing_pgc:<r>,
                                  grid_width_pgc:<r>,
                                  l_rtma:<r>,
                                  l_rtmb:<r>,
                                  rim:<si>,
                                  grid_type_pgc:<si>,
                                  grid_orientation_pgc:<si>,
                                  roof_orientation_rtma:<si>,
                                  roof_orientation_rtmb:<si>),
                           ...)

where
<i> = integer
<n> = name of an object
<r> = real number
<s> = character string
<si> = item from a list of character strings

```

Attributes

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

No of Components (*no_of_components*) [integer], default: **0**.

Frame Unit (*frame_unit*) [item from a list of character strings], default: **m**.

Size (*size*) [struct].

 x Min (*x_min*) [real number], default: **0**.

 x Max (*x_max*) [real number], default: **1**.

 y Min (*y_min*) [real number], default: **0**.

y Max (*y_max*) [real number], default: **0.8**.

Grid (*grid*) [struct].

Show (*show*) [item from a list of character strings], default: **on**.

on

off

Snap to (*snap_to*) [item from a list of character strings], default: **on**.

on

off

x Spacing (*x_spacing*) [real number], default: **0.05**.

y Spacing (*y_spacing*) [real number], default: **0.05**.

Connection (*connection*) [item from a list of character strings], default: **free**.

free

fixed

Feeds (*feeds*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Beam Radius (*beam_radius*) [real number], default: **0.1**.

Phase Curvature (*phase_curvature*) [real number], default: **0**.

Polarization Angle (*polarization_angle*) [real number], default: **0**.

Wavelength (*wavelength*) [real number], default: **0.00999308333**.

Loads (*loads*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Load Size x (*load_size_x*) [real number], default: **0.2**.

Load Size y (*load_size_y*) [real number], default: **0.2**.

Lenses (*lenses*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Beam2 to Component (*beam2_to_component*) [integer], default: **0**.

Beam2 to Port (*beam2_to_port*) [integer], default: **0**.

F Value (*f_value*) [real number], default: **0.1**.

Refractive Index (*refractive_index*) [real number], default: **2**.

Diameter (*diameter*) [real number], default: **0.2**.

Curvature Difference (*curvature_difference*) [real number], default: **0**.

Bs1 (*bs1*) [real number], default: **0**.

Bs2 (*bs2*) [real number], default: **0**.

Plane Mirrors (*plane_mirrors*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Beam2 to Component (*beam2_to_component*) [integer], default: **0**.

Beam2 to Port (*beam2_to_port*) [integer], default: **0**.

Height (*height*) [real number], default: **0.2**.

Rim (*rim*) [item from a list of character strings], default: **square**.

square

circular

Reflectors (*reflectors*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Beam2 to Component (*beam2_to_component*) [integer], default: **0**.

Beam2 to Port (*beam2_to_port*) [integer], default: **0**.

F Value (*f_value*) [real number], default: **0.1**.

Curvature Difference (*curvature_difference*) [real number], default: **0**.

Half Cone Angle (*half_cone_angle*) [real number], default: **5**.

Cone Axis Angle (*cone_axis_angle*) [real number], default: **0**.

Cone Vertex Focal Point (*cone_vertex_focal_point*) [item from a list of character strings], default: **P1**.

P1

P2

Beam Splitters (*beam_splitters*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Beam2 to Component (*beam2_to_component*) [integer], default: **0**.

Beam2 to Port (*beam2_to_port*) [integer], default: **0**.

Beam3 to Component (*beam3_to_component*) [integer], default: **0**.

Beam3 to Port (*beam3_to_port*) [integer], default: **0**.

Beam4 to Component (*beam4_to_component*) [integer], default: **0**.

Beam4 to Port (*beam4_to_port*) [integer], default: **0**.

Type (*type*) [item from a list of character strings], default: **power_splitter**.

power_splitter

polarizer

Frequency1 Lower (*frequency1_lower*) [real number], default: **0**.

Frequency1 Upper (*frequency1_upper*) [real number], default: **-1**.

Frequency2 Lower (*frequency2_lower*) [real number], default: **0**.

Frequency2 Upper (*frequency2_upper*) [real number], default: **1**.

Reflection1 (*reflection1*) [real number], default: **50**.

Transmission1 (*transmission1*) [real number], default: **50**.

Reflection2 (*reflection2*) [real number], default: **50**.

Transmission2 (*transmission2*) [real number], default: **50**.

Grid Type (*grid_type*) [item from a list of character strings], default: **ideal**.

ideal

wire

strip

Direction (*direction*) [real number], default: **0**.

Distance (*distance*) [real number], default: **0**.

Width (*width*) [real number], default: **0.2**.

Height (*height*) [real number], default: **0.2**.

Rim (*rim*) [item from a list of character strings], default: **square**.

square

circular

Mich Interfers (*mich_interfers*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Beam2 to Component (*beam2_to_component*) [integer], default: **0**.

Beam2 to Port (*beam2_to_port*) [integer], default: **0**.

Diameter (*diameter*) [real number], default: **0.2**.

L 1 (*l_1*) [real number], default: **0.2**.

L 2 (*l_2*) [real number], default: **0.2**.

Marpup Interfers (*marpup_interfers*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Beam2 to Component (*beam2_to_component*) [integer], default: **0**.

Beam2 to Port (*beam2_to_port*) [integer], default: **0**.

Beam3 to Component (*beam3_to_component*) [integer], default: **0**.

Beam3 to Port (*beam3_to_port*) [integer], default: **0**.

Beam4 to Component (*beam4_to_component*) [integer], default: **0**.

Beam4 to Port (*beam4_to_port*) [integer], default: **0**.

Size Lac (*size_lac*) [real number], default: **0.075**.

Size Lbc (*size_lbc*) [real number], default: **0.075**.

Side Length (*side_length*) [real number], default: **0.05**.

Grid Spacing Pga (*grid_spacing_pga*) [real number], default: **0.001**.

Grid Width Pga (*grid_width_pga*) [real number], default: **0.0001**.

Grid Spacing Pgb (*grid_spacing_pgb*) [real number], default: **0.001**.

Grid Width Pgb (*grid_width_pgb*) [real number], default: **0.0001**.

Grid Spacing Pgc (*grid_spacing_pgc*) [real number], default: **0.001**.

Grid Width Pgc (*grid_width_pgc*) [real number], default: **0.0001**.

L Rtma (*l_rtma*) [real number], default: **0.075**.

L Rtmb (*l_rtmb*) [real number], default: **0.075**.

Rim (*rim*) [item from a list of character strings], default: **square**.

square

circular

Grid Type Pga (*grid_type_pga*) [item from a list of character strings], default: **ideal**.

ideal

wire

strip

Grid Type Pgb (*grid_type_pgb*) [item from a list of character strings], default: **ideal**.

ideal

wire

strip

Grid Type Pgc (*grid_type_pgc*) [item from a list of character strings], default: **ideal**.

ideal

wire

strip

Grid Orientation Pga (*grid_orientation_pga*) [item from a list of character strings], default: **horizontal**.

horizontal

vertical

Grid Orientation Pgb (*grid_orientation_pgb*) [item from a list of character strings], default: **horizontal**.

horizontal

vertical

Grid Orientation Pgc (*grid_orientation_pgc*) [item from a list of character strings], default: **horizontal+vertical**.

horizontal+vertical

horizontal-vertical

Roof Orientation Rtma (*roof_orientation_rtma*) [item from a list of character strings], default: **vertical**.

vertical

horizontal

Roof Orientation Rtmb (*roof_orientation_rtmb*) [item from a list of character strings], default: **vertical**.

vertical

horizontal

Connectors (*connectors*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Connected Frame (*connected_frame*) [name of an object], default: **blank**.

Adjoining Connector (*adjoining_connector*) [character string].

Connector Angle (*connector_angle*) [real number], default: **0**.

Conn Coor Sys (*conn_coor_sys*) [name of an object], default: **blank**.

Aperture In Screen (*aperture_in_screen*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Beam2 to Component (*beam2_to_component*) [integer], default: **0**.

Beam2 to Port (*beam2_to_port*) [integer], default: **0**.

Aperture Radius (*aperture_radius*) [real number], default: **0.2**.

Plane Waves (*plane_waves*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Polarisation Angle (*polarisation_angle*) [real number], default: **0**.

Wavelength (*wavelength*) [real number], default: **0.00999308333**.

Aperture Radius (*aperture_radius*) [real number], default: **0.2**.

Near Fields (*near_fields*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Sampling Type (*sampling_type*) [item from a list of character strings], default: **planar_cut**.

planar_cut

planar_grid

Polarisation (*polarisation*) [item from a list of character strings], default: **linear**.

linear

circular

Number of Cuts (*number_of_cuts*) [integer], default: **0**.

Np (*np*) [integer], default: **0**.

Rho End (*rho_end*) [real number], default: **0**.

Width (*width*) [real number], default: **0**.

Far Fields (*far_fields*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0**.

Center y (*center_y*) [real number], default: **0**.

Orientation (*orientation*) [real number], default: **0**.

Beam1 to Component (*beam1_to_component*) [integer], default: **0**.

Beam1 to Port (*beam1_to_port*) [integer], default: **0**.

Sampling Type (*sampling_type*) [item from a list of character strings], default: **spherical_grid**.

spherical_cut

spherical_grid

Polarisation (*polarisation*) [item from a list of character strings], default: **linear**.

linear

circular

theta-phi

Number of Cuts (*number_of_cuts*) [integer], default: **0**.

Np (*np*) [integer], default: **0**.

theta End (*theta_end*) [real number], default: **0**.

Width (*width*) [real number], default: **0**.

Rooftop Interfers (*rooftop_interfers*) [sequence of structs].

Name (*name*) [character string].

Index (*index*) [integer], default: **0**.

Center x (*center_x*) [real number], default: **0.**

Center y (*center_y*) [real number], default: **0.**

Orientation (*orientation*) [real number], default: **0.**

Beam1 to Component (*beam1_to_component*) [integer], default: **0.**

Beam1 to Port (*beam1_to_port*) [integer], default: **0.**

Beam2 to Component (*beam2_to_component*) [integer], default: **0.**

Beam2 to Port (*beam2_to_port*) [integer], default: **0.**

Side Length (*side_length*) [real number], default: **0.05.**

Grid Spacing Pgc (*grid_spacing_pgc*) [real number], default: **0.001.**

Grid Width Pgc (*grid_width_pgc*) [real number], default: **0.0001.**

L Rtma (*l_rtma*) [real number], default: **0.075.**

L Rtmb (*l_rtmb*) [real number], default: **0.075.**

Rim (*rim*) [item from a list of character strings], default: **square.**

square

circular

Grid Type Pgc (*grid_type_pgc*) [item from a list of character strings], default: **ideal.**

ideal

wire

strip

Grid Orientation Pgc (*grid_orientation_pgc*) [item from a list of character strings], default: **horizontal+vertical.**

horizontal+vertical

horizontal-vertical

Roof Orientation Rtma (*roof_orientation_rtma*) [item from a list of character strings], default: **vertical**.

vertical

horizontal

Roof Orientation Rtmb (*roof_orientation_rtmb*) [item from a list of character strings], default: **vertical**.

vertical

horizontal

Remarks

The attributes are autogenerated by the GUI for which reason a detailed explanation for the attributes is not given.

PLOT OBJECTS

Purpose

Plot objects are obsolete but maintained for compatibility with older GRASP-versions. The objects are used for drawing the defined geometry, a task which is now automatically performed by the GUI.

To create a plot, an object of the special class

Plot Settings (Obsolete)

must exist to specify the plot. Objects of this class manages the plotting of all other plot objects.

The coordinate systems to be plotted can be specified in objects of class:

Coordinate System Plot

and the plot of the scatterers may be managed by one of the objects in

Scatterer Plot

The plotting of the sources (including currents) are controlled from the relevant classes of

Source Plot

and the points for the field calculation may be visualized by

Output Points Plot

It is also possible to draw GO rays radiated from a source or another point and follow the rays through reflections in the modelled structure by objects of one of the classes in

Ray Plot

Remarks

The defined geometrical objects may be plotted as line drawings or as solids. Line drawings are obtained by XYZ-plots. The objects are drawn solid or as line drawings in GRASP by OpenGL-plots. See under remarks in class *Plot Settings (Obsolete)* for details.

Examples of the different types of plot may be found in the description of *Reflector Plot*.

Links

Classes→*Plot Objects*

PLOT SETTINGS (OBSOLETE) (*plot_settings*)

Purpose

The class *Plot Settings (Obsolete)* is used for defining a set of plot elements of the specified geometry. The calculated plot data may be stored on file and the plot may be shown in a window on the screen.

The class is obsolete but maintained for compatibility with older GRASP-versions. The necessary plot objects are automatically generated by the GUI.

Links

[Classes](#)→[Plot Objects](#)→*Plot Settings (Obsolete)*

Remarks

Syntax

```
<object name> plot_settings
(
    plot_accuracy           : <rl>,
    coor_sys                : ref(<n>),
    file_name               : <f>
)
where
<n> = name of an object
<rl> = real number with unit of length
<f> = file name
```

Attributes

Plot Accuracy (*plot_accuracy*) [real number with unit of length], default: **0**.

A curved line will be generated as a series of straight segments. The length in the 3D space of such straight segments is less than the specified Plot Accuracy. The Plot Accuracy must be specified to a length which is small compared to the curvature of the line pieces in the objects to be plotted in order to get a smooth appearance of the curve. On the other hand, when the length is too small the time for generating the plot may be unacceptably large. If Plot Accuracy is specified to zero the program automatically calculates a value that is suitable for the current geometry.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the classes *Coordinate Systems* defining the coordinate system (the plot coordinate system) in which the calculated 3D points will be expressed. In general, the plot will be shown with the *z*-axis of this coordinate system oriented as vertical up.

File Name (*file_name*) [file name].

Name of the file to which the calculated coordinates of the 3D data shall be written according to one of the commands given in the section below. When a File Name is not specified then data will not be written to a file. The available format of the file is described in *Three-Dimensional Plotter Data*. The recommended file extension is .xyz.

Command Types

The generation of the specified plot file is activated by one of the following commands for generating data in TICRA XYZ-format:

Get XYZ Plot (Obsolete)

Add XYZ Plot (Obsolete) and

Get All XYZ Plot (Obsolete)

commands for generating data in TICRA XYZ-format for OpenGL drawings:

Get OpenGL Plot (Obsolete)

Get All OpenGL Plot (Obsolete)

Remarks

The plots may be generated and presented on the screen in a GRASP session and the plot data may be stored on file for later usage.

Two different types of on-screen plots are invoked through the screen menus. The first type is a line drawing obtained by:

Cmds. > Plot Commands > Plot all objects

The plot is a parallel projection as specified by the user. It is possible to plot projections parallel to the coordinate planes and the plots may be printed in a fixed scale.

The second type of plots is OpenGL Plots obtained by:

Cmds. > Plot Commands > OpenGL, all objects

These plots are central projections (perspective plots) and the objects may be plotted as solids or as wire grid models. By mouse operations the geometry may be rotated giving a good impression of the relative positions of the objects.

If a file name is given in attribute File Name to the object named *D3_PLOT*, then this file will be written (or updated if it already exists) when a plot is generated on the screen applying the menu item *Cmds.*

The geometry plot may also be stored on file by the commands listed above (see section Commands). Three data formats are available, XYZ-plots, DXF-plots and OpenGL plots.

In the first format, the plot data are calculated as (x, y, z) -coordinates of the geometry and stored in a TICRA formatted XYZ-data file. A drawing may be generated by the program *XYZ_Plot* enclosed to GRASP. The drawing will appear as a line drawing in which the objects are transparent.

In the second format, the plot data of the geometry are expressed as a set of facets, which may be stored in a file in a general DXF-format. This format

can be read by many CAD software packages, and in a drawing based on the DXF-data the geometrical objects will appear as solids. Drawing tools of this format is not a part of GRASP.

The last format is applied for OpenGL plots which later may be plotted by GRASP by the menu commands

Cmds. > Plot Commands > OpenGL, XYZ file

The objects may appear as XYZ-plots (option *Wiregrids*), DXF-plots (*Hidden lines*) or as solids (*Solid surfaces*).

It shall be noted that lines upon a surface of a scatterer may not be seen at a drawing applying a hidden line technique as the lines may be hidden by the surface on which they are drawn. Instead a line drawing or wire grid plot shall be chosen.

Examples on the three plot types may be found under Remarks in the description of class *Reflector Plot*.

COORDINATE SYSTEM PLOT (coor_sys_plot)

Purpose

The class *Coordinate System Plot* specifies the details for plotting of coordinate systems.

The attributes of the class are in the GUI of GRASP controlled from the guitxt3D-View Settings obtained by right clicking the 3D-View canvas.

Links

[Classes](#)→[Plot Objects](#)→*Coordinate System Plot*

Remarks

Syntax

```
<object name> coor_sys_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    axis_length          : <rl>,
    relative_arrow_length : <r>
)
where
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Select Coordinate Systems (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Coordinate Systems*. The name of the reference may be 'all' which comprises references to all *Coordinate Systems* objects of the project. It is the specified (or all) objects, which are plotted.

Axis Length (*axis_length*) [real number with unit of length], default: **0**.

Length of the drawn part of the axes of the specified coordinate systems. If Axis Length is specified to zero, a value based on the structure dimensions will be applied.

Relative Arrow Length (*relative_arrow_length*) [real number], default: **0.1**.

Length of an arrow head in the end of the axes of the specified coordinate systems, given relative to Axis Length.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

Objects of this class define and plot the axes of one or more coordinate systems. Each of the three axes is drawn from the origin with a length as specified by Axis Length in the positive direction of the axis.

The axes x , y and z are drawn in the colours Red, Green and Blue (RGB), respectively, on the screen. In the figures of the manual this colour code is applied, however, colour representation is not standard and different colours may appear on different computer systems.

When some coordinate systems shall be drawn with long axes, others with short axes, these may be drawn by two different objects of class *Coordinate System Plot*. One object defines a long Axis Length and references those coordinate systems, which shall be drawn with long axes. The other object defines a short Axis Length and references the coordinate systems, which shall be drawn with short axes.

An example of a plot with coordinate systems of different sizes is given in Figure 1.

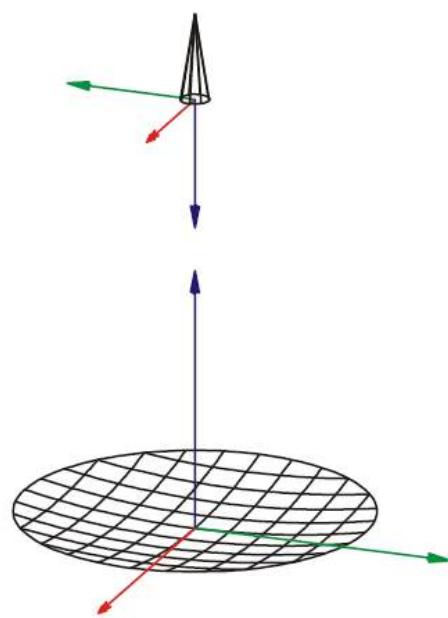


Figure 1

Examples of coordinate systems drawn with different lengths of the axes. For the reflector coordinate system the Axis Length is 30 m (the reflector diameter is 40 m) and the Relative Arrow Length is 0.08. For the feed coordinate system the Axis Length is 15 m and the Relative Arrow Length is 0.16. In this way the length of all arrows become 2.4 m.

SCATTERER PLOT

Purpose

By objects of the menu *Scatterer Plot* it is possible to control the illustration of the geometry of the modelled scatterers:

Reflector Plot

Reflector with Panels Plot

Triangular Plate Plot

Plate Plot

Circular Struts Plot

Polygonal Struts Plot

Box Plot

Load Plot

Tabulated Mesh Plot

DGR Intercostals Plot

Wires Plot

Aperture in Screen Plot

Lens Plot

Body of Revolution Plot

Links

Classes→*Plot Objects*→*Scatterer Plot*

REFLECTOR PLOT (reflector_plot)

Purpose

Objects of the class *Reflector Plot* define how objects of the class *Reflector* will be plotted.

Plotting of a *Reflector with Panels* is controlled from the class *Reflector with Panels Plot*.

Links

Classes→*Plot Objects*→*Scatterer Plot*→*Reflector Plot*

Remarks

Syntax

```
<object name> reflector_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    rim_plot              : <si>,
    surface_plot          : <si>,
    el_prop_plot          : <si>,
    x_lines               : <i>,
    y_lines               : <i>,
    obsolete_phi_facets  : <i>,
    obsolete_rho_facets  : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Select Reflectors (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Reflector*. The name of the reference may be 'all' which comprises references to all *Reflector* objects of the project. It is the specified (or all) objects, which are plotted.

Rim Plot (*rim_plot*) [item from a list of character strings], default: **on**.

Controls plotting of the rim of the reflector.

on

The rim is plotted.

off

The rim is not plotted.

Surface Plot (*surface_plot*) [item from a list of character strings], default: **on**.

Controls plotting of the surface of the reflector.

on

The surface is plotted.

off

The surface is not plotted.

Electrical Properties Plot (*el_prop_plot*) [item from a list of character strings], default: **on**.

Controls the plotting of defined *Electrical Properties* of the reflector or, more precisely, the directions of the wires in a defined grid or mesh. It is not all *Electrical Properties* that can be plotted.

on

Wires are plotted.

off

Wires are not plotted.

x-Lines (*x_lines*) [integer], default: **9**.

Applies to XYZ-plots and OpenGL plots, see the remarks below. The number of lines drawn on the reflector surface parallel to the *xz*-plane of the reflector coordinate system.

y-Lines (*y_lines*) [integer], default: **9**.

Applies to XYZ-plots and OpenGL plots, see the remarks below. The number of lines drawn on the reflector surface parallel to the *yz*-plane of the reflector coordinate system.

phi-Facets (Obsolete) (*obsolete_phi_facets*) [integer], default: **36**.

Applies to DXF-plots, see the remarks below. phi-Facets (Obsolete) is the number of facets on a full azimuth circle in ϕ .

rho-Facets (Obsolete) (*obsolete_rho_facets*) [integer], default: **10**.

Applies to DXF-plots, see the remarks below. `rho-Facets` (Obsolete) is the number of facets on a radius along ρ .

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

The plot of a reflector can show the reflector surface and symbolic grid lines illustrating the electrical properties of the surface:

Plot of reflector surface

For backward compatibility, a file with plot data for the reflector surface may be generated in three different ways, as XYZ-plots, as DXF-plots and as OpenGL plots, see below. These methods are today considered obsolete because the reflector – and most of the other geometric objects – are drawn in the 3D-VIEW window of GRASP as solids (corresponding to OpenGL plot) with a set of surface lines (corresponding to XYZ plot). The features of this 3D-VIEW window are controlled by right clicking the canvas of the window.

The three ways of plotting is explained and illustrated by examples in the following. Finally, plotting of electrical properties will be considered.

In the first way for generating a plot file (XYZ plot), the surface is illustrated by a grid of lines, the surface lines, and it will appear transparent. The lines are, when projected on the aperture plane, parallel to the x - and y -axes of the reflector coordinate system, see Figure 1. The number of lines parallel to the x -axis is $x\text{-Lines}$, and these lines are equidistantly spaced between y_{\min} and y_{\max} , the minimum and maximum y -values for the rim of the reflector. The line spacing along y is thus $(y_{\max} - y_{\min}) / ('x\text{-Lines}' + 1)$. The line spacing along x is determined in the same way.

The reflector surface is plotted as a grid of surface lines by the XYZ-commands of class *Plot Settings (Obsolete)*.

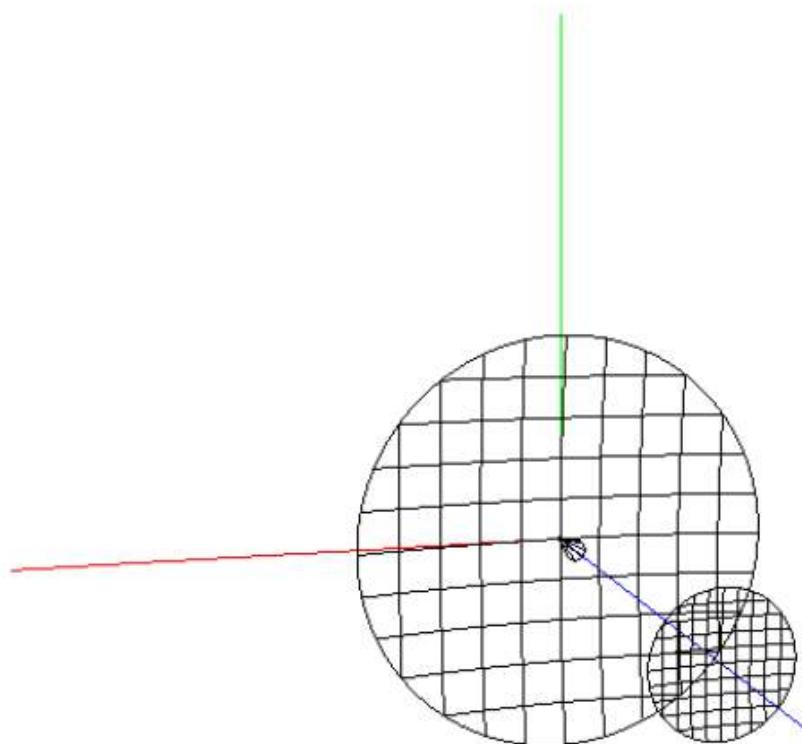


Figure 1

Dual reflector system drawn as a grid of surface lines. The lines are specified by $x\text{-Lines}: 9$ and $y\text{-Lines}: 9$ for both the main and the subreflector.

The second way in which the surface may be drawn, is as solid facets given in a polar $\rho\phi$ -grid when the reflector surface is projected on the xy -plane of the reflector coordinate system, see Figure 2. The number of facets along a circle of fixed radius is phi-Facets (Obsolete), and the number along a radius is rho-Facets (Obsolete). The facets are further subdivided by a diagonal into two triangles. The facets may be drawn as solids as it is illustrated in the figure.

The reflector surface is plotted with facets by the DXF-commands of class *Plot Settings (Obsolete)*.

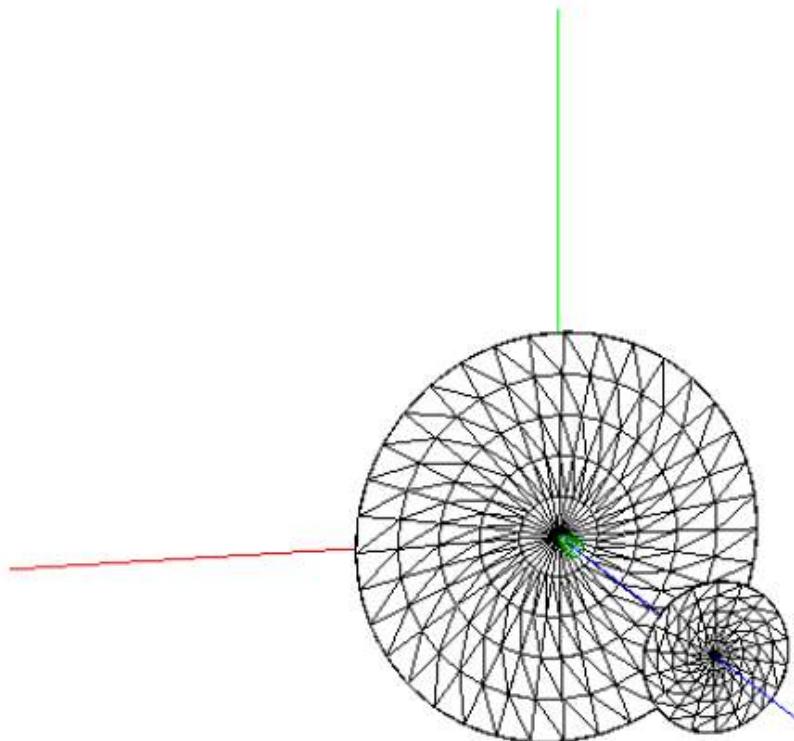


Figure 2

The same dual reflector system drawn with facets. The facets are specified by 36 phi-Facets (Obsolete) for the main reflector and 18 phi-Facets (Obsolete) for the subreflector. For both reflectors the radial specification is 5 rho-Facets (Obsolete).

Finally, the reflector may be drawn as solids by the OpenGL commands of class *Plot Settings (Obsolete)* and as shown in Figure 3. The drawing of the surface is controlled by the values of x-Lines and y-Lines as in the grid of surface lines for XYZ-plots.

Both x-Lines and y-Lines must be positive. The solid surface is not drawn if one of the attributes is zero.

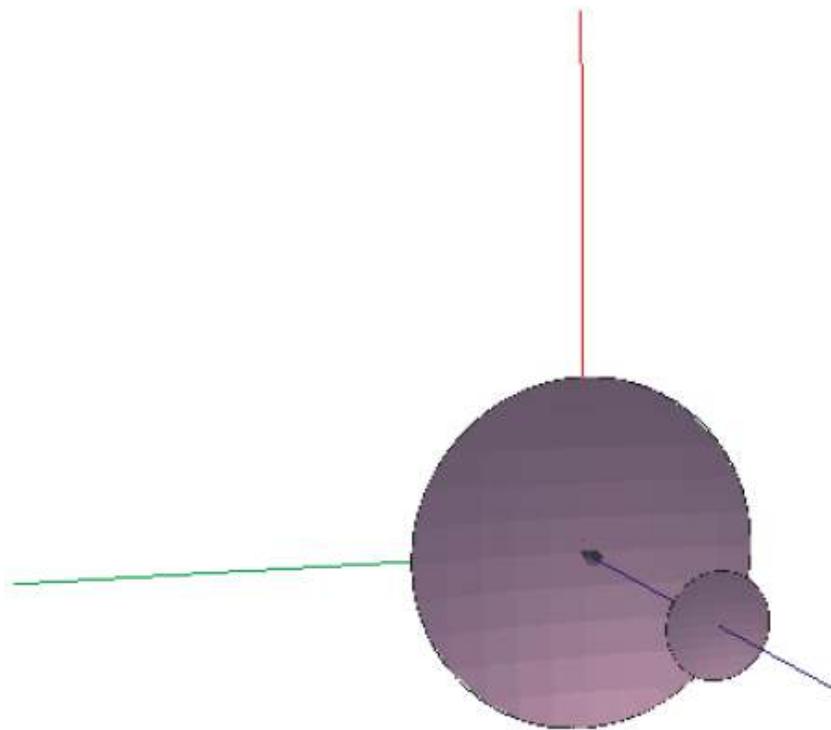


Figure 3

The reflector system drawn in OpenGL.

Plot of electrical properties

The electrical properties of a grid (and a mesh) may be illustrated in a plot by drawing lines parallel to lines of the grid (but with a much larger spacing than the true grid). The number of plotted lines are defined in the appropriate object of class *Electrical Properties* and the plot is activated by specifying Electrical Properties Plot to 'on'.

The illustration of the electrical properties may sporadically be hampered by the plot of the reflector surface as specified by x-Lines and y-Lines. Especially, when the reflector surface is drawn as a solid surface in OpenGL then the drawn grid may somewhere be hidden behind the solid reflector. In such cases x-Lines and y-Lines shall both be specified to zero. (In the GUI the tic-marks for showing surface lines and surfaces shall be removed in the menu appearing when the canvas is given a right click). The surface itself will then not be drawn and the grid will be visible from all viewing directions.

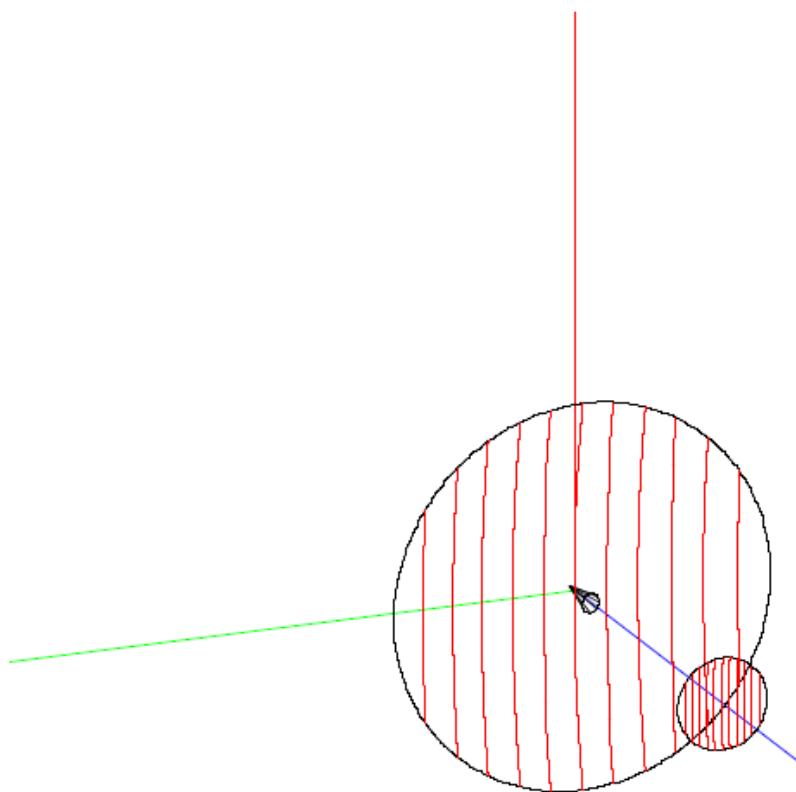


Figure 4

Dual reflector system drawn with electrical properties, here a vertical grid, on both reflectors. Both x-Lines and y-Lines are zero. The number of plotted lines are specified to 11 in the grid-specifying object (see the classes defining the *Electrical Properties*).

REFLECTOR WITH PANELS PLOT (reflector_with_panels_plot)

Purpose

Objects of the class *Reflector with Panels Plot* define how objects of the class *Reflector with Panels* (i.e. *Panels in Polar Grid* and *Individually Defined Panels*) are plotted.

Links

Classes→*Plot Objects*→*Scatterer Plot*→*Reflector with Panels Plot*

Remarks

Syntax

```
<object name> reflector_with_panels_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    panel_plot_specification : sequence(
        struct(
            start_panel:<i>,
            end_panel:<i>,
            rho_lines:<i>,
            phi_lines:<i>,
            rim_plot:<si>),
        ...))
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Select Reflectors (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Reflector with Panels*.
The name of the reference may be 'all' which comprises references

to all *Reflector with Panels* objects of the project. It is the specified (or all) objects, which are plotted.

Panel Plot Specification (*panel_plot_specification*) [sequence of structs].

Defines which panels should be included in the plot and how many lines or facets should be used for each panel. If two or more successive panels of the *Reflector with Panels* reflector have similar size, the lines and facets may be specified for all these panels by defining an interval of panels by the members Start Panel and End Panel. If there are no such identical panels, Panel Plot Specification must contain the same number of elements as there are panels which should be included in the plot. For each panel contained in the interval of panels, the number of lines and facets is defined by the members Rho-Lines and Phi-Lines.

Start Panel (*start_panel*) [integer], default: **-1**.

Number of the first panel in the interval of panels. Must be positive or -1 (all panels are included when both Start Panel and End Panel are specified to -1, see also the remarks below).

End Panel (*end_panel*) [integer], default: **-1**.

Number of the last panel in the interval of panels. Must be positive or -1 (see the remarks below).

Rho-Lines (*rho_lines*) [integer], default: **9**.

For each panel in the interval of panels, the number of plot lines on the panel surface drawn along the radial ρ -direction for reflectors of class *Panels in Polar Grid*, or along the x -direction of the reflector coordinate system for reflectors of class *Individually Defined Panels*. The number must be non-negative.

Phi-Lines (*phi_lines*) [integer], default: **9**.

For each panel in the interval of panels, the number of plot lines on the panel surface drawn along the azimuthal ϕ -direction for reflectors of class *Panels in Polar Grid*, or along the y -direction of the reflector coordinate system for reflectors of class *Individually Defined Panels*. The number must be non-negative.

Rim Plot (*rim_plot*) [item from a list of character strings], default: **on**.

For each panel in the interval of panels, determines if the rim of the panel is plotted:

on

The rim is plotted.

off

The rim is not plotted.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

This class, *Reflector with Panels Plot*, is used to plot objects of the type *Reflector with Panels* which consists of the two subclasses *Panels in Polar Grid* and *Individually Defined Panels*. The plots appear differently for the two subclasses as the reflectors are defined differently.

The first figure shows an example of a plot of an object defined by the class *Panels in Polar Grid*. The reflector consists of a central circular panel surrounded by 36 panels in three rings of 8, 12 and 16 panels, respectively.

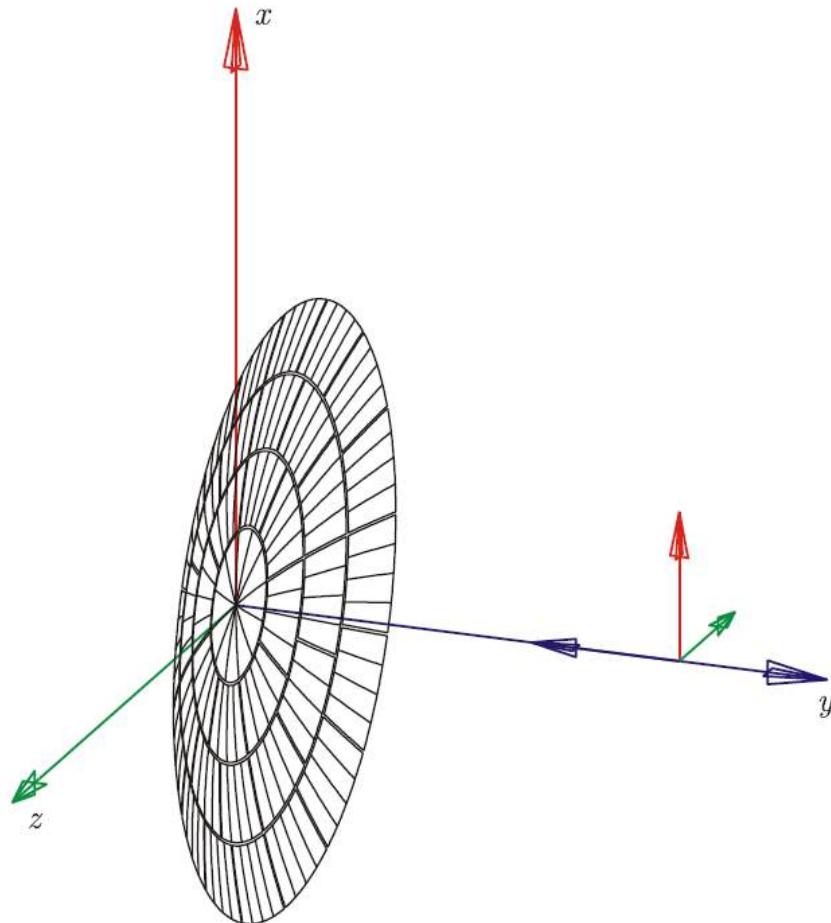


Figure 1

Paraboloidal reflector defined by *Panels in Polar Grid*. In the central panels 12 radial lines are drawn (*rho_lines* = 12) and in all the other panels 3 radial lines are drawn. There are not drawn lines in the ϕ -direction (*phi_lines* = 0).

The specification for the plot lines in the panels are given by

```
panel_plot_specification: sequence
(
  struct( start_panel: 1, end_panel: 1, rho_lines: 12,
  phi_lines: 0, rim_plot: on)
  struct( start_panel: 2, end_panel: 37, rho_lines: 3,
  phi_lines: 0, rim_plot: on),
)
```

The first struct defines the centre panel and the second struct defines the remaining panels of the rings.

In the second figure an example of a plot of an object defined by the class *Individually Defined Panels* is given. The example is the same as one of the examples given in the description of *Individually Defined Panels* apart from the number of lines along x and y (of the reflector coordinate system) over the panels.

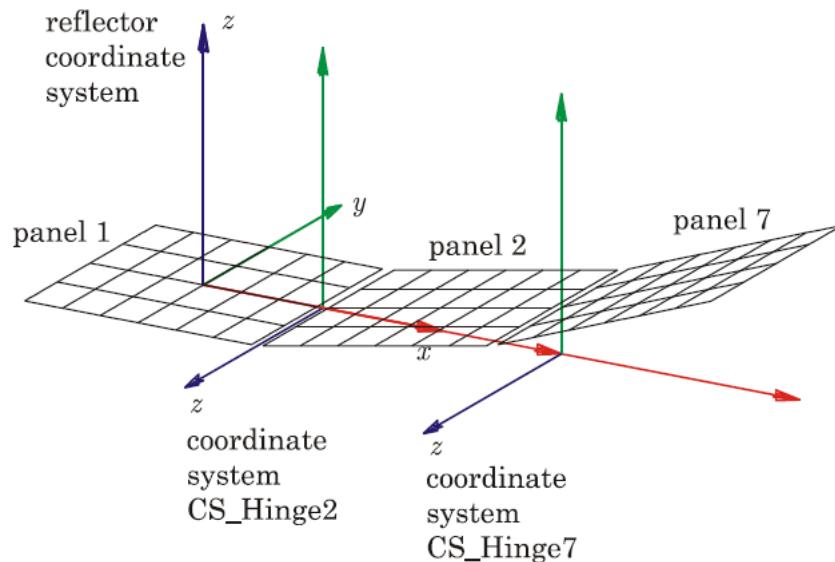


Figure 2

Three panels of a reflector defined by *Individually Defined Panels*. All panels have been drawn with 3 lines along x (ρ _lines = 3) and 5 lines along y (ϕ _lines = 5), x and y of the reflector coordinate system. The figure also shows two hinge coordinate system which are irrelevant for the plotting.

The specification for the plot lines in the panels are given by

```
panel_plot_specification: sequence
(
  struct( start_panel: -1, end_panel: -1,
    rho_lines: 3, phi_lines: 5, rim_plot: on)
)
```

All panels are included in the specification as *start_panel* and *end_panel* are both specified to -1.

TRIANGULAR PLATE PLOT (triangular_plate_plot)

Purpose

Objects of the class *Triangular Plate Plot* define how objects of the class *Triangular Plate* shall be plotted.

Links

Classes→*Plot Objects*→*Scatterer Plot*→*Triangular Plate Plot*

Remarks

Syntax

```
<object name> triangular_plate_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    rim_plot              : <si>,
    normal_plot           : <si>,
    el_prop_plot          : <si>,
    s1_lines              : <i>,
    s2_lines              : <i>,
    s3_lines              : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Triangular Plate*. The name of the reference may be 'all' which comprises references to all *Triangular Plate* objects of the project. It is the specified (or all) objects, which are plotted.

Rim Plot (*rim_plot*) [item from a list of character strings], default: **on**.

Controls plotting of the rim of the *Triangular Plate*.

on

The rim is plotted.

off

The rim is not plotted.

Normal Plot (*normal_plot*) [item from a list of character strings], default: **off**.

Controls the plotting of a normal to the plane of the *Triangular Plate*. The length of the normal depends on the size of the triangular plate. The normal is plotted from *corner_1* of the triangular plate, see Remarks below.

off

No normal is plotted

on

A normal is plotted

EI Prop Plot (*el_prop_plot*) [item from a list of character strings], default: **on**.

Controls the plotting of defined *Electrical Properties* of the triangular plate or, more precisely, the directions of the wires in a defined grid or mesh. It is not all Electrical Properties that can be plotted.

on

Wires are plotted.

off

Wires are not plotted.

S1 Lines (*s1_lines*) [integer], default: **9**.

Number of lines drawn on the surface parallel to side 1 of the triangular plate (the side between *corner_1* and *corner_2* in the specification of the *Triangular Plate*).

S2 Lines (*s2_lines*) [integer], default: **9**.

Number of lines drawn on the surface parallel to side 2 of the triangular plate (the side between *corner_2* and *corner_3* in the specification of the *Triangular Plate*).

S3 Lines (*s3_lines*) [integer], default: **9**.

Number of lines drawn on the surface parallel to side 3 of the triangular plate (the side between *corner_3* and *corner_1* in the specification of the *Triangular Plate*).

Command Types

The available commands for generating plots are described in [Plot Settings \(Obsolete\)](#).

Remarks

Objects of class [Triangular Plate Plot](#) are defined by the three corners (`corner_1`, `corner_2` and `corner_3`), which again denote the numbering of the sides from 1 to 3, as shown in Figure 1.

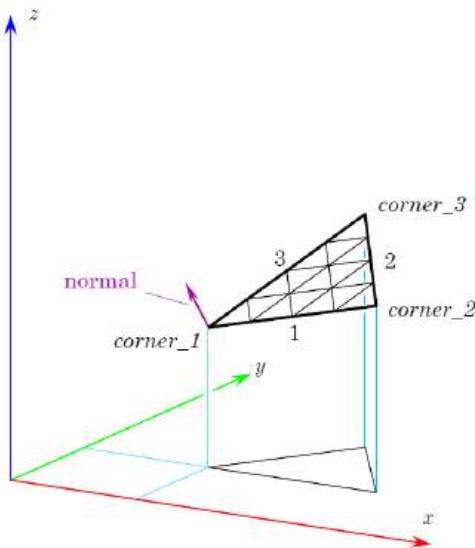


Figure 1

Triangular Plate defined by an object of class [Triangular Plate](#) in which the position of the 3 corners are given. The plot of the triangular plate is specified by an object of class [Triangular Plate Plot](#) as follows, `rim_plot` and `normal_plot` are on, `s1_lines`, `s2_lines` and `s3_lines` are all 3. The projection of the first triangular plate in the xy -plane defines another triangular plate, which is drawn with another object of class [Triangular Plate Plot](#) now with `normal_plot` specified to off, `s1_lines`, `s2_lines` and `s3_lines` are all 0, but `rim_plot` is still on.

An example on plotting of electrical properties are shown in Figure 2.

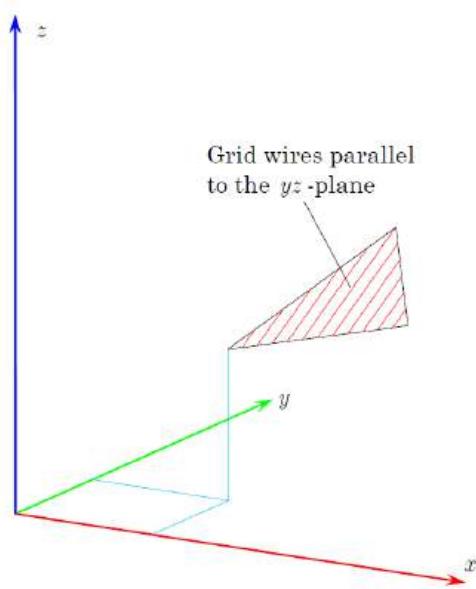


Figure 2

The same triangular plate as in Figure 1 but drawn with electrical properties: a wire grid with wires oriented parallel to the yz -plane. In the grid-specifying object `plot_lines` (the number of lines) are specified to 11 (see classes of [Electrical Properties](#)). In [Triangular Plate Plot](#) `s1_lines`, `s2_lines` and `s3_lines` are all specified to zero in order to avoid the black lines over the triangular plate as in Figure 1.

PLATE PLOT (plate_plot)

Purpose

Objects of the class *Plate Plot* define how objects of the class *Parallelogram*, class *Rectangular Plate*, or class *Circular Plate* shall be plotted.

Links

Classes→*Plot Objects*→*Scatterer Plot*→*Plate Plot*

Remarks

Syntax

```
<object name> plate_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    rim_plot              : <si>,
    normal_plot           : <si>,
    el_prop_plot          : <si>,
    s1_lines              : <i>,
    s2_lines              : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Parallelogram* or *Rectangular Plate*. The name of the reference may be 'all' which comprises references to all *Parallelogram* and *Rectangular Plate* objects of the project. It is the specified (or all) objects, which are plotted.

Rim Plot (*rim_plot*) [item from a list of character strings], default: **on**.

Controls plotting of the rim of the parallelogram.

on

The rim is plotted.

off

The rim is not plotted.

Normal Plot (*normal_plot*) [item from a list of character strings], default: **off**.

Controls the plotting of a normal to the plane of the parallelogram. The length of the drawn normal depends on the size of the parallelogram.

off

No normal is plotted.

on

A normal is plotted.

El Prop Plot (*el_prop_plot*) [item from a list of character strings], default: **on**.

Controls the plotting of defined **Electrical Properties** of the parallelogram or, more precisely, the directions of the wires in a defined grid or mesh. It is not all Electrical Properties that can be plotted.

on

Wires are plotted.

off

Wires are not plotted.

S1 Lines (*s1_lines*) [integer], default: **9**.

Number of lines drawn on the surface parallel to the 1'st edge of the parallelogram (specified by the attribute *vec_1* of the **Parallelogram** and by the attributes *corner_1* and *corner_2* of the **Rectangular Plate**). For **Circular Plate**, the lines are drawn on the surface parallel to the *x*-axis of the plate.

S2 Lines (*s2_lines*) [integer], default: **9**.

Number of lines drawn on the surface parallel to the 2'nd edge of the parallelogram (specified by the attribute *vec_2* of the **Parallelogram** and perpendicular to the 1'st edge of the **Rectangular Plate**). For **Circular Plate**, the lines are drawn on the surface parallel to the *y*-axis of the plate.

Command Types

The available commands for generating plots are described in **Plot Settings (Obsolete)**.

Remarks

The parameters for the parallelogram definition are illustrated in Figure 1.

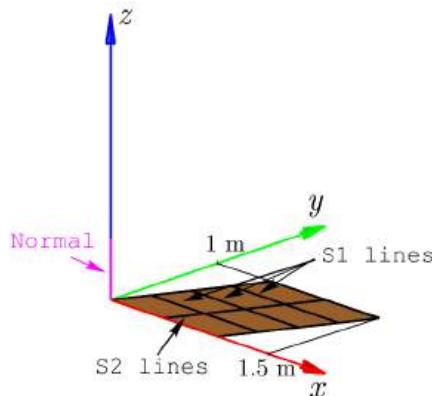


Figure 1

Parallelogram defined by an object of the class **Parallelogram**, with Vec 1 specified to (1 m, 0 m, 0 m) and Vec 2 specified to (0.5 m, 1 m, 0 m). In **Plate Plot** Rim Plot and Normal Plot are on, while S1 Lines is specified to 3 and S2 Lines to 1. The parallelogram is drawn with the coordinate system in which it is defined.

An example on plotting of electrical properties are shown in Figure 2.

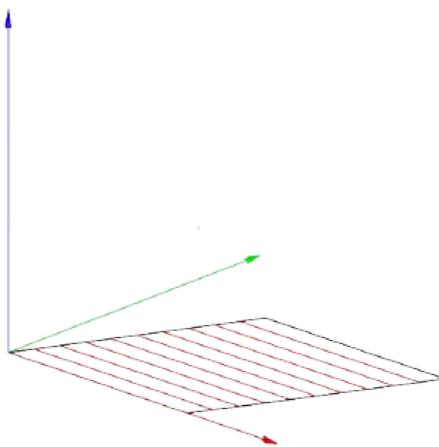


Figure 2

The same parallelogram as in Figure 1 shown with electrical properties, namely a wire grid with wires oriented parallel to the x -axis. In the grid-specifying object, the number of lines are specified to Plot Lines: 9 (see classes of **Electrical Properties**). In **Plate Plot** S1 Lines and S2 Lines are both specified to zero in order to avoid the black lines parallel to the edges in Figure 1.

CIRCULAR STRUTS PLOT (circular_struts_plot)

Purpose

Objects of the class *Circular Struts Plot* define how objects of the class *Circular Struts* shall be plotted.

Links

[Classes](#)→[Plot Objects](#)→[Scatterer Plot](#)→*Circular Struts Plot*

Syntax

```
<object name> circular_struts_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    lines                 : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Circular Struts*. The name of the reference may be 'all' which comprises references to all *Circular Struts* objects of the project. It is the specified (or all) objects, which are plotted.

Lines (*lines*) [integer], default: **8**.

Number of lines drawn parallel to the strut axis to show the surface of the strut.

Command Types

The available commands for generating plots are described in [Plot Settings \(Obsolete\)](#).

POLYGONAL STRUTS PLOT (polygonal_struts_plot)

Purpose

Objects of the class *Polygonal Struts Plot* define how objects of the class *Polygonal Struts* shall be plotted.

Links

Classes→*Plot Objects*→*Scatterer Plot*→*Polygonal Struts Plot*

Remarks

Syntax

```
<object name> polygonal_struts_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...)
)
where
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Polygonal Struts*. The name of the reference may be 'all' which comprises references to all *Polygonal Struts* objects of the project. It is the specified (or all) objects, which are plotted.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

An example of a plot of polygonal struts is given in Figure 1. The subreflector of a dual reflector system is supported by three rectangular struts, which are defined in the shown strut coordinate systems.

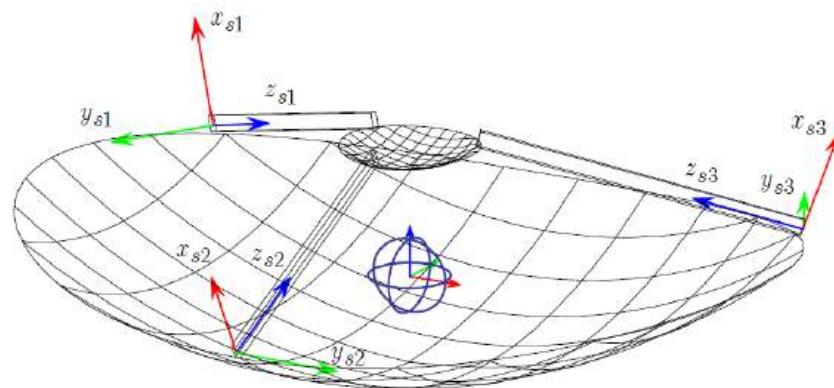


Figure 1

Illustration of a dual reflector system in which the subreflector is supported by three struts of rectangular cross-section. The cross-sections of the struts are given in the xy -planes of the respective strut coordinate systems, the z -axes of which also define the direction of the struts. Further, in this antenna model the feed is given by an expansion of spherical waves. This source, *Tabulated SWE Coefficients*, is by *Feed Plot* drawn as three great circles, here shown in blue.

BOX PLOT (box_plot)

Purpose

Objects of the class *Box Plot* define how objects of the class *Box* shall be plotted.

Links

Classes→*Plot Objects*→*Scatterer Plot*→*Box Plot*

Remarks

Syntax

```
<object name> box_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    s_lines               : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Box*. The name of the reference may be 'all' which comprises references to all *Box* objects of the project. It is the specified (or all) objects, which are plotted.

S Lines (*s_lines*) [integer], default: **0**.

Number of lines drawn parallel to each edge of the box.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

An example of a plot with a box is shown in Figure 1 showing a satellite with a deployed antenna on the front side. The satellite is modelled as a box and it is plotted with *s_lines* specified to 4.

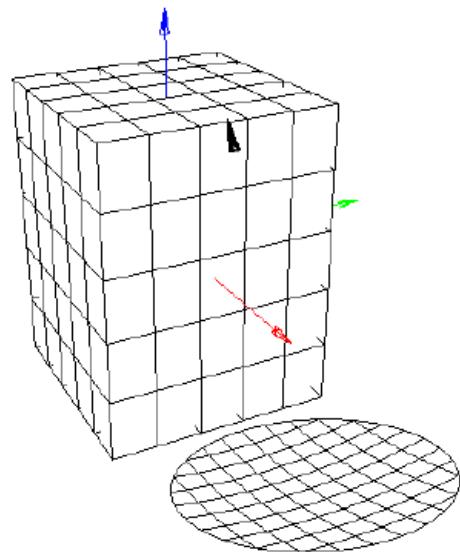


Figure 1

Illustration of an application of *Box Plot*.

LOAD PLOT (load_plot)

Purpose

Objects of the class *Load Plot* define how objects of the class *Load* shall be plotted.

The class *Load Plot* specifies the details for plotting of loads.

Links

Classes→*Plot Objects*→*Scatterer Plot*→*Load Plot*

Remarks

Syntax

```
<object name> load_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...)
)
where
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **plot_all**.

Controls the status of the plot. The following choices are possible:

plot_as_specified_in_objects

Obsolete options maintained for backward compatibility. All loads will be shown when this option is selected.

plot_all

All objects of class *Load* are plotted - independently of the plot settings (given by attribute *plot_status* of the individual *Load* objects).

plot_none

No *Load* objects are plotted.

plot_load_names

The *Load* objects specified in attribute *Objects* below are plotted.

Objects (*objects*) [sequence of names of other objects], default: **all**.

This attribute has only effect when attribute `load_selection` is specified to '`plot_load_names`'. Sequence of references to objects of the class `Load`. The name of the reference may be 'all' which comprises references to all `Load` objects of the project. Only the specified `Load` objects are plotted - and they are plotted independently of the plot settings (given by attribute `plot_status`) of the individual `Load` objects.

Command Types

The available commands for generating plots are described in [*Plot Settings \(Obsolete\)*](#).

Remarks

An example on the plot of a load is given in Figure 1. The load is drawn symbolically as a pyramidal absorber but the size of the pyramids is not related to any specific frequency.

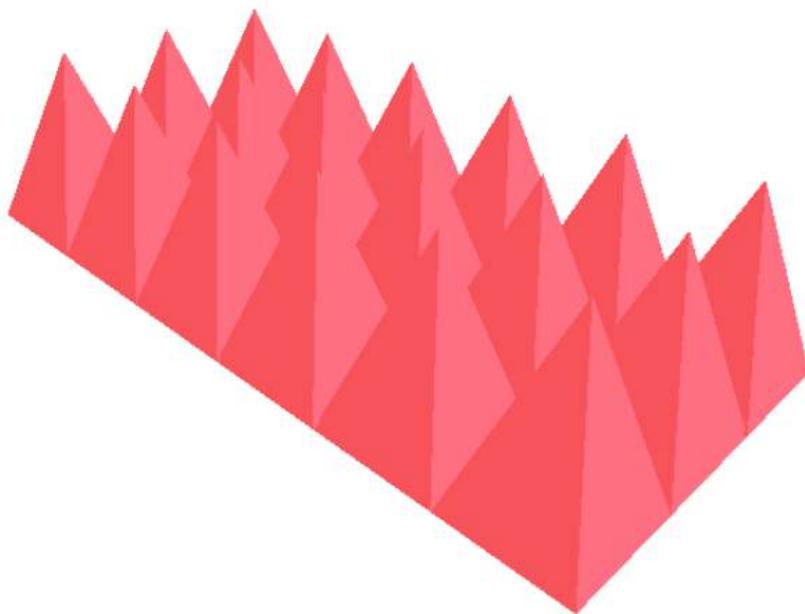


Figure 1

Illustration of a load which is 10 cm by 20 cm in size.

TABULATED MESH PLOT (tabulated_mesh_plot)

Purpose

Objects of the class *Tabulated Mesh Plot* define how scatterers of the class *Tabulated Mesh* are plotted.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Plot Objects](#)→[Scatterer Plot](#)→[Tabulated Mesh Plot](#)

Remarks

Syntax

```
<object name> tabulated_mesh_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    show_normals          : <si>,
    show_iso_lines        : <si>
)
where
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Tabulated Mesh*. The name of the reference may be 'all' which comprises references to all *Tabulated Mesh* objects of the project. It is the specified (or all) objects, which are plotted.

Show Normals (*show_normals*) [item from a list of character strings], default: **off**.

Controls if normal vectors are plotted at the centre of each patch. The following may be specified:

off

Normal vectors are not plotted.

on

Normal vectors are plotted.

Show Iso Lines (*show_iso_lines*) [item from a list of character strings], default: **on**.

The isoparametric lines connect the nodes defining the patch and include the outline of the patch. The outline is always plotted while the inner isoparametric lines on the patches may be included or not according to this attribute. There are no inner isoparametric lines for patches without inner nodes. See the Remarks below for an example. The following may be specified:

on

Isoparametric lines are plotted.

off

Isoparametric lines are not plotted.

Remarks

The *Tabulated Mesh Plot* specifies how to plot the mesh defined in the specified *Tabulated Mesh* objects. In a MoM analysis, the mesh must, patch by patch, fulfil a number of rules as listed in the description of the *Tabulated Mesh* class. The rules depend on the wavelength used in the MoM analysis. Therefore, the *Tabulated Mesh Plot* cannot be used to check if the patches fulfil these rules. Instead, this can be checked using the *MoM Plot* class.

It should be noted that the *Tabulated Mesh Plot* plots the user-defined mesh, whereas the *MoM Plot* plots the mesh used in the MoM calculations. The two meshes may be different.

The isoparametric lines

Each patch has an associated curvilinear parametric (u, v) coordinate system where u and v runs from -1 to +1. If the attribute *show_iso_lines* is specified to 'on', isoparametric lines will be drawn through the inner nodes (if any) of the patch.

An example of the parametric lines for a quadrilateral patch defined by 9 nodes is shown in the following figure. The patch has a single inner node for $(u, v) = (0, 0)$ and the isoparametric lines for $u = 0$ and $v = 0$ will be plotted if *show_iso_lines* is specified to 'on'.

The triangular patch is considered as a special case of a quadrilateral patch for which one edge is collapsed. The isoparametric lines will be plotted accordingly.

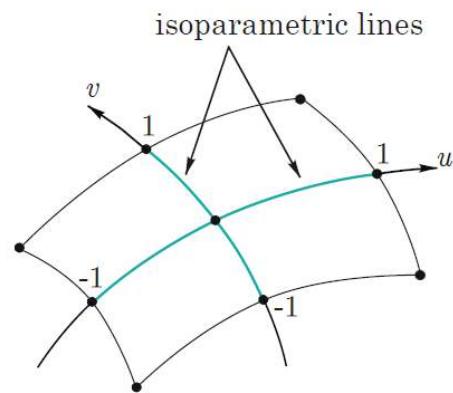


Figure 1

The isoparametric lines for a patch defined by 9 nodes (the dots) with the outline of the patch shown as black lines and the inner isoparametric lines shown in light blue.

WIRES PLOT (wires_plot)

Purpose

Objects of the class *Wires Plot* are used to plot objects belonging to the classes *Piecewise Straight Wire* and *Curved Wire*.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Plot Objects](#)→[Scatterer Plot](#)→[Wires Plot](#)

Remarks

Syntax

```
<object name> wires_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    accuracy_index        : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Piecewise Straight Wire* or *Curved Wire*. The name of the reference may be 'all' which comprises references to all *Piecewise Straight Wire* and *Curved Wire* objects of the project. It is the specified (or all) objects, which are plotted.

Accuracy Index (*accuracy_index*) [integer], default: **1**.

Specifies the plotting accuracy relative to the default. The default plotting accuracy is sufficient for most viewing purposes. If a very high quality plotting output is desired, the Accuracy Index can be increased beyond the default of 1. The number of flat facets used to plot the wire is then multiplied by Accuracy Index.

Remarks

An example of a plot of an object of class *Curved Wire* is shown in the following figure of a helix. The helix is defined by six points for each turn.

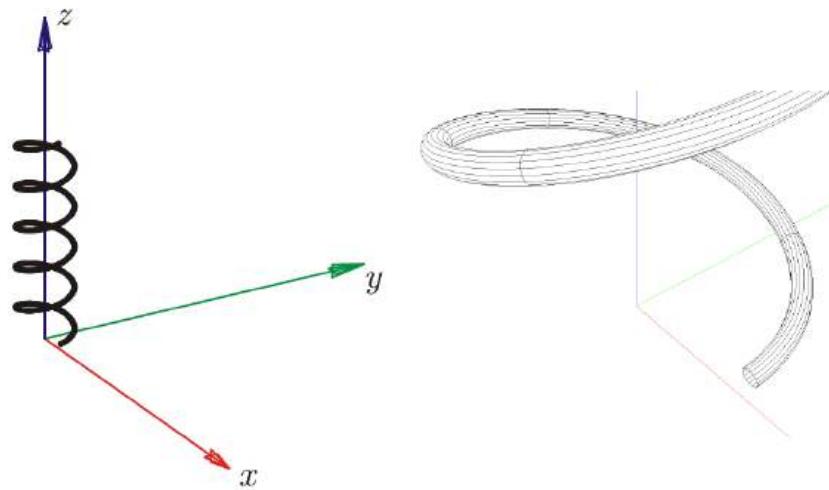


Figure 1

Example of a five turn helix (left) with a detail (right) showing the segments of the helix.

DGR INTERCOSTALS PLOT (dgr_intercostals_plot)

Purpose

Objects of the class *DGR Intercostals Plot* specify how intercostals for dual-gridded reflectors (DGR) are plotted.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Plot Objects](#)→[Scatterer Plot](#)→*DGR Intercostals Plot*

Remarks

Syntax

```
<object name> dgr_intercostals_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    plot_segments         : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of class *DGR Intercostals*. The intercostals defined by the specified (or all) objects of these classes are plotted.

Plot Segments (*plot_segments*) [integer], default: **1**.

Specifies the number of segments to use when generating the plot, see the Remarks belowfor details. The default value 1 is sufficient for most purposes but the number of segments may be increased to a higher value if desired.

Remarks

It is the geometry of the intercostals which is plotted by *DGR Intercostals Plot*. In MoM calculations an artificial gap is introduced at each intercostal/reflector-interface. The resulting, slightly smaller stiffeners (lower support ring) may be visualised using the *MoM Plot* class which shows the mesh of the MoM currents on the intercostals. In that case the *plot_status* in the *DGR Intercostals Plot* must be set to 'off' in order to the plot of the intercostals not to block for the MoM mesh.

As an examples on the applications of *DGR Intercostals Plot* consider first a double reflector with seven stiffeners, shown in Figure 1 and Figure 2.

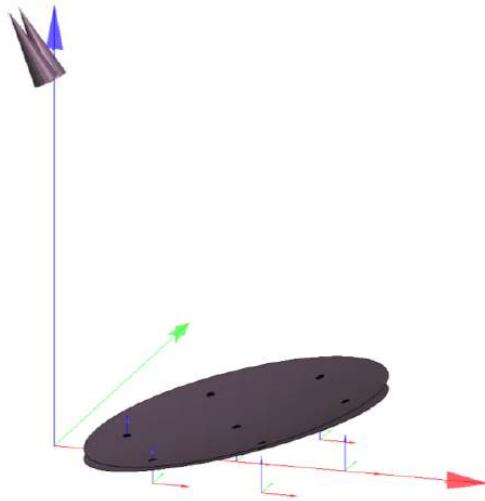


Figure 1

Illustration of a plot performed by *DGR Intercostals Plot*. A dual reflector system with a rear and a front reflector separated by seven circular dielectric stiffeners is shown. The attribute *plot_segments* has the value 1.

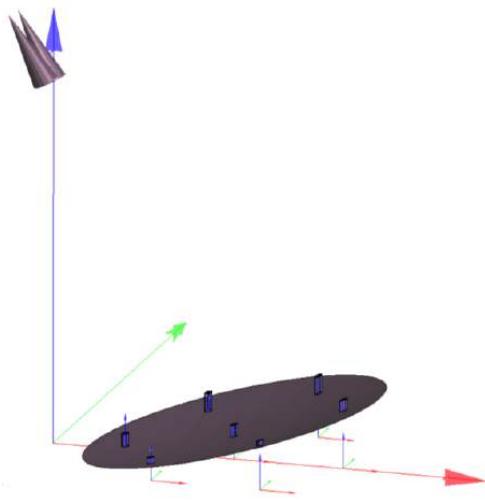


Figure 2

The same plot as in Figure 1 but with the front reflector removed in order to see the geometry of the stiffeners.

Another example is the plot of a thick polygonal stiffener between a parabolic and a plane reflector as shown below. The end-faces of a stiffener conform

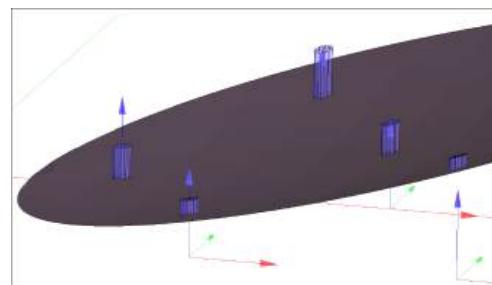


Figure 3

Details of Figure 2 with the stiffeners and the defining coordinate systems.

to the nearest reflector surface. Thus the edges of the end-faces are in general curved. Each of these edges is divided in an even number of segments which is *plot_segments* (or *plot_segments* + 1) and each segment is approximated by a cubic curve. The number of segments along the longitudinal edges of the stiffener is *plot_segments*. The faces of the stiffener are segmented according to the number of segments along the edges. By default *plot_segments* is 1 which is applied in the top figure below while *plot_segments* is 4 in the bottom figure.

For circular stiffeners the number of segments on the cylindrical face is again *plot_segments* along the axis while it is eight times *plot_segments* along the periphery of the end faces.

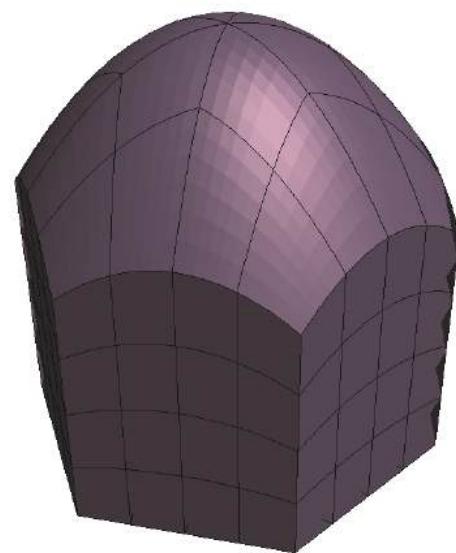
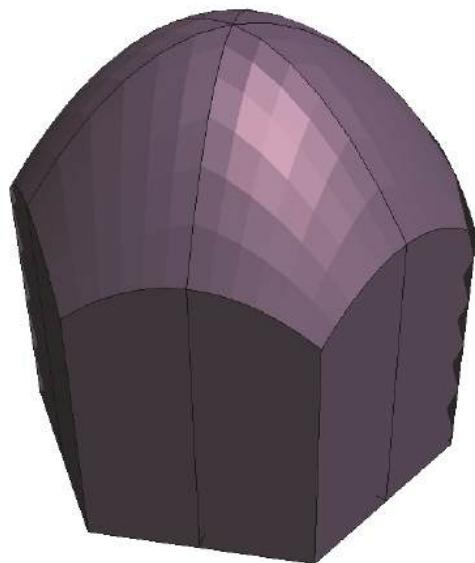


Figure 4

A very thick polygonal stiffener shown with *plot_segments*: 1 (top) and *plot_segments*: 4 (bottom)

APERTURE IN SCREEN PLOT (aperture_in_screen_plot)

Purpose

Objects of the class *Aperture in Screen Plot* define how objects of the class *Aperture in Screen* shall be plotted.

This class is only available with the Quast add-on.

Links

[Classes](#)→[Plot Objects](#)→[Scatterer Plot](#)→[Aperture in Screen Plot](#)

Remarks

Syntax

```
<object name> aperture_in_screen_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    lines                 : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Select Apertures (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Aperture in Screen*. The name of the reference may be 'all' which comprises references to all *Aperture in Screen* objects of the project. It is the specified (or all) objects, which are plotted.

Lines (*lines*) [integer], default: **9**.

Number of lines drawn diagonally over the surface of the screen (apart from the aperture).

Command Types

The available commands for generating plots are described in [Plot Settings \(Obsolete\)](#).

Remarks

The infinite screen is drawn as a square of about three times the diameter of the aperture. It is only in the plots that the screen is limited to this size. The conducting area of the screen is shown as solid in OpenGL plots and is drawn hatched in wire-grid plots with `Lines` being the number of diagonal lines. The aperture is drawn as a hole in the screen.

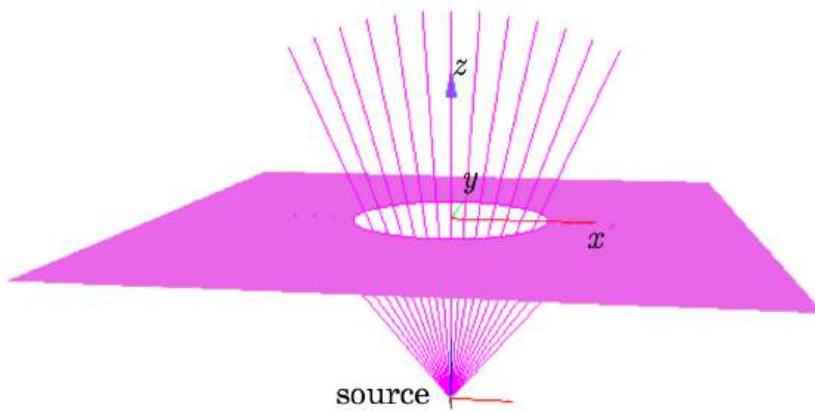


Figure 1 Plot of a source and an aperture in a screen plotted as a solid body.

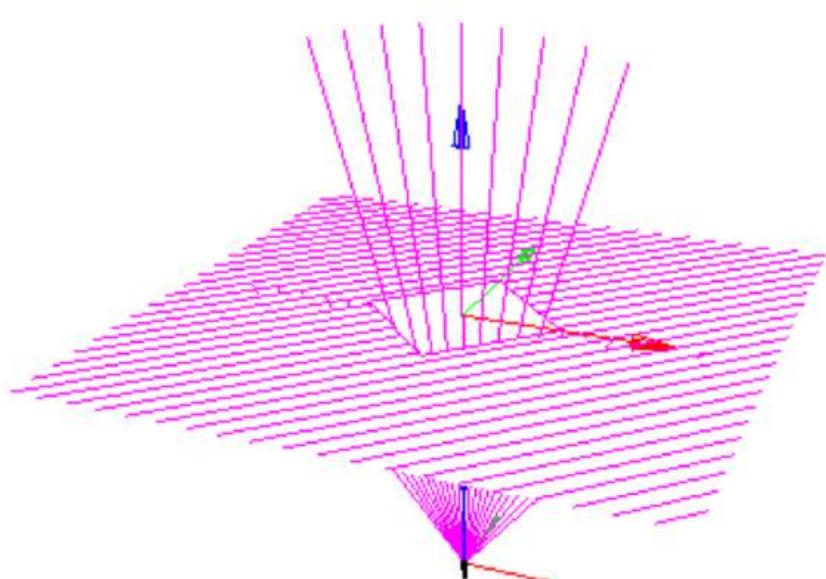


Figure 2 A similar plot of a screen with square aperture in which the screen is plotted as wire grid with `Lines` specified to 39.

LENS PLOT (lens_plot)

Purpose

Objects of the class *Lens Plot* define how objects of the class *Simple Lens* shall be plotted.

This class is only available with the Quast add-on.

Links

[Classes](#)→[Plot Objects](#)→[Scatterer Plot](#)→[Lens Plot](#)

Remarks

Syntax

```
<object name> lens_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    rim_plot              : <i>,
    phi_facets            : <i>,
    rho_facets            : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Simple Lens*. The name of the reference may be 'all' which comprises references to all *Simple Lens* objects of the project. It is the specified (or all) objects, which are plotted.

Rim Plot (*rim_plot*) [item from a list of character strings], default: **on**.

Controls plotting of the rim of the lens.

on

The rim is plotted.

off

The rim is not plotted.

phi Facets (*phi_facets*) [integer], default: 36.

phi Facets is the number of facets plotted on a full azimuth circle in ϕ . See the remarks below.

Rho Facets (*rho_facets*) [integer], default: 10.

Rho Facets is the number of facets on a radius along ρ . See the remarks below.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

The lens faces are drawn as facets given in a polar $\rho\phi$ -grid when the lens surface is projected on the xy -plane of the lens coordinate system, see the following figures. The number of facets along a circle of fixed radius is *phi Facets*, and the number along a radius is *Rho Facets*. The lens may be plotted either as a wire-grid drawing or as a solid OpenGL object. Though solid, the lens is drawn transparent.

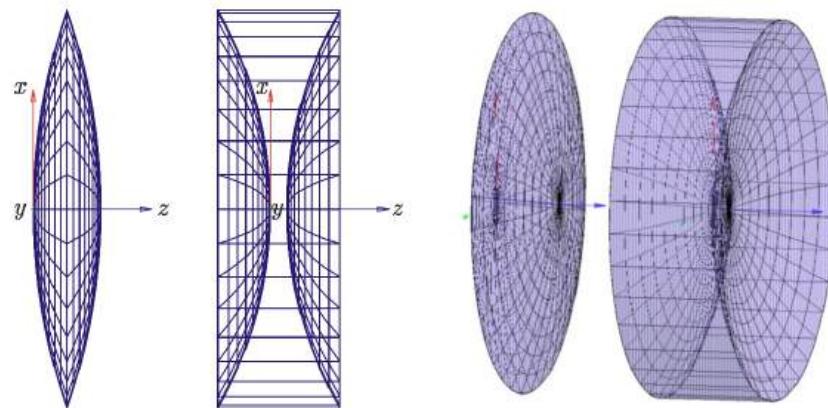


Figure 1

A biconvex lens and a concave lens drawn with facets 36 phi Facets and 10 Rho Facets. To the left the lenses are shown as a wire-grid drawing and to the right as a solid OpenGL drawing.

BODY OF REVOLUTION PLOT (bor_plot)

Purpose

Objects of the class *Body of Revolution Plot* define how objects of the scatterers of the type *Body of Revolution* are plotted.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Plot Objects](#)→[Scatterer Plot](#)→[Body of Revolution Plot](#)

Remarks

Syntax

```
<object name> bor_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    phi_lines             : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Body of Revolution*.

The name of the reference may be 'all' which comprises references to all *Body of Revolution* objects of the project. It is the specified (or all) objects, which are plotted.

phi-Lines (*phi_lines*) [integer], default: **0**.

The number of lines to display the generatrix at given ϕ -values.

Remarks

An example of a plot with a *Body of Revolution* scatterer is shown in Figure 1. Here, a *BoR Mesh* is used to define a closed dielectric cylinder. The cylinder is plotted with phi-Lines specified to 10.

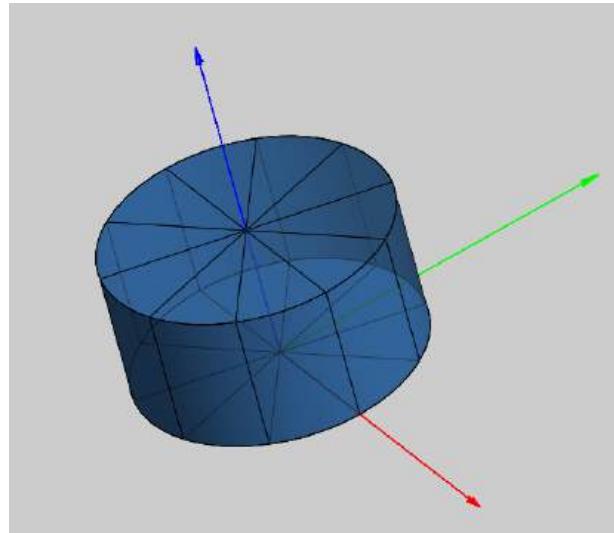


Figure 1 Illustration of an application of *Body of Revolution Plot* .

SOURCE PLOT

Purpose

By objects of the menu *Source Plot* it is possible to control the illustration of the geometry of the modelled sources:

Feed Plot

Plane Wave Plot

Currents Plot

GTD Plot

Tabulated Planar Source Plot

Plot Feed as Conical Horn

MoM Source Plot

Currents Colour Plot (only for MoM determined currents),

Finally, a plot of the mesh applied in MoM computations is given by

MoM Plot

Links

Classes→*Plot Objects*→*Source Plot*

FEED PLOT (feed_plot)

Purpose

The class *Feed Plot* specifies the plot of a feed.

Links

Classes→*Plot Objects*→*Source Plot*→*Feed Plot*

Remarks

Syntax

```
<object name> feed_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    spherical_mode_sphere : <si>,
    rayleigh_sphere       : <si>
)
where
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Feed* or class *Array*. The name of the reference may be 'all', which comprises references to all *Feed* objects and *Array* objects of the project. It is the specified (or all) objects, which are plotted. When an array is specified then all the feed elements of the array object are plotted. If 'all' is specified then feeds of an array will only be drawn as part of the array and not on their own. This is to avoid having these feeds drawn twice. If such feeds shall be drawn on their own, they must be specified individually in the sequence.

Spherical Mode Sphere (*spherical_mode_sphere*) [item from a list of character strings], default: **off**.

Controls the plot of a sphere related to the number of spherical modes used to express the field of the feed. The field determination may not be reliable inside the radius of this sphere (see the remarks below).

off

The spherical-mode sphere is not plotted.

on

Plot of the spherical-mode sphere.

Rayleigh Sphere (*rayleigh_sphere*) [item from a list of character strings], default: **off**.

Controls the plot of a sphere related to the far-field distance of the feed (see the remarks below).

off

The Rayleigh sphere is not plotted.

on

Plot of the Rayleigh sphere.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

If possible, the feed is drawn according to its *Feed* definition, i.e. the *Helix* is drawn as a helix and a *Rectangular Horn* is drawn as a rectangular, pyramidal horn. But if the aperture of the horn is too small to a flare to exist then only the aperture is drawn.

Some feeds such as the *Gaussian Beam*, *Pattern Def.* and the *Tabulated Pattern* do not, however, have a specific geometry and these feeds will be plotted as a conical horn illustrated by a circular symmetric cone. In case of *Tabulated Pattern*, the horn aperture is one wavelength and the length of the cone is three wavelengths.

An example of a plot with a simple conical horn representing the feed may be found under the remarks in the description of class *Rays from Point Sources*.

The feed is placed in the feed coordinate system as defined in the *Feed*.

A special feed model is the *Tabulated SWE Coefficients*, which is plotted as three perpendicular great circles illustrating a sphere. The radius, r , of the circles is given by

$$r = N/k \quad (1)$$

The source, *Tabulated SWE Coefficients*, is given by an expansion in spherical modes and N is the upper limit for the polar mode index of the modes

and k is the wavenumber, $2\pi/\lambda$. The sphere drawn here is the same as the spherical-mode sphere explained below.

An example on a plot of a source given by its *Tabulated SWE Coefficients* is found under remarks in class *Polygonal Struts Plot*.

A feed may also be drawn by *Plot Feed as Conical Horn* whereby various rotational symmetric structures may be drawn.

The plotted spheres

If requested, the spherical-mode sphere and the Rayleigh sphere of the *Feed* may be plotted. If an *Array* is specified the spheres will be plotted for the individual *Feeds* of the array. The spheres are plotted as the three perpendicular great circles.

The field of a *Feed* is expanded in spherical modes in order to represent the near-field effects accurately (unless *far_forced* is specified to 'on' in the definition of the *Feed*). The upper index in such an expansion is denoted N and the expansion will be valid outside a sphere having radius

$$r_{swe} = N/k \quad (2)$$

and centred at the origin of the coordinate system in which the *Feed* is specified; k is the wavenumber.

The spherical-mode sphere has importance when the field of a *Feed* shall be calculated at a short distance. In that case the field will be expressed as an expansion in spherical modes (or spherical waves). But this expansion will only be valid outside the spherical-mode sphere. If the user tries to calculate the field inside the sphere a warning is issued. By drawing the spherical-mode sphere the user has the possibility to check that the field points are all placed in the valid region outside the sphere. See also the section on *Near-Field Calculations*.

The Rayleigh sphere has the so-called Rayleigh distance, or far-field distance, as radius. The sphere is again centred at the origin of the feed coordinate system. Conventionally, the far-field expression (when *far_forced* is specified to 'on' in the definition of the *Feed*) is valid at distances larger than the Rayleigh distance and a plot of the Rayleigh sphere can thus help to estimate whether a given scatterer is in the far field of the source.

The Rayleigh distance, R , is defined as the distance from which a plane aperture of diameter D is seen with a phase error less than 22.5° peak to peak,

$$R = \frac{2D^2}{\lambda} \quad (3)$$

Applying the far field at this distance is thus still an approximation to the true field.

In general the spherical-mode sphere will be less than the Rayleigh sphere,

$$r_{swe} < R \quad (4)$$

(this is not the case for small antennas with diameter $D = 2r_o$ being less than the wavelength λ).

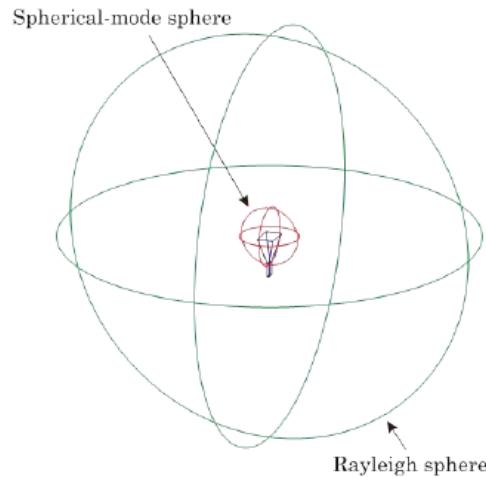


Figure 1 Spherical-mode sphere and Rayleigh sphere drawn for a rectangular horn.

The drawing of a feed with the spherical-mode sphere and the Rayleigh sphere included is illustrated in Figure 1.

The figure shows a rectangular horn. The aperture has the dimensions 3λ by 2λ whereby the diameter is

$$D = 2r_o = \sqrt{(3\lambda)^2 + (2\lambda)^2} \cong 3.6\lambda \quad (5)$$

and, according to the description of the *Rectangular Horn*, we obtain the radius of the spherical-mode sphere as

$$r_{swe} = N/k \cong r_o + 1.6\lambda = 3.4\lambda. \quad (6)$$

For the Rayleigh sphere we obtain

$$R = 2D^2/\lambda = 26\lambda. \quad (7)$$

PLANE WAVE PLOT (plane_wave_plot)

Purpose

The class *Plane Wave Plot* specifies the details for plotting of objects of the class *Plane Wave*.

Links

Classes→*Plot Objects*→*Source Plot*→*Plane Wave Plot*

Remarks

Syntax

```
<object name> plane_wave_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    rim_plot              : <si>,
    direction_plot        : <si>,
    lines                 : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Plane Wave*. The name of the reference may be 'all' which comprises references to all *Plane Wave* objects of the project. It is the specified (or all) *Plane Wave* objects, which are plotted.

Rim Plot (*rim_plot*) [item from a list of character strings], default: **on**.

Controls plotting of the rim of the circle representing the plane wave.

on

The rim is plotted.

off

The rim is not plotted.

Direction Plot (*direction_plot*) [item from a list of character strings], default: **on**.

Controls the plotting of the propagation direction of the plane wave.

on

The propagation direction is plotted.

off

No propagation direction is plotted.

Lines (*lines*) [integer], default: **4**.

The number of radial lines drawn from the centre of the circle representing the plane wave to the rim of the same circle.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

As a plane wave is infinitely large, it is drawn as a circular aperture parallel to a phase front of the wave, and a normal to this indicating the direction of the plane wave. The radius of the drawn aperture is given by the attribute *aperture_radius* of the *Plane Wave* object to be drawn.

An example is given in Figure 1, which also shows rays radiated from the plane wave and reflected in a spherical reflector. The reflector is defined in the reflector coordinate system $x_r y_r z_r$.

The plane wave is defined in the plane-wave coordinate system (with subscript *pw* in the figure). The plane wave is plotted as the blue circle in the x_{pw}, y_{pw} -plane (Rim Plot: on) with the 4 radii (Lines: 4).

The plane wave radiates in direction of the normal, which is plotted at the centre of the plane-wave aperture (Direction Plot: on).

The circle and the plane-wave coordinate system are defined in class *Plane Wave*. The radius of the circle is 25m in the applied scaling.

The rays of the plane wave, here drawn in purple, are defined in an object of the class *Rays from Plane Waves*. See that class for details.

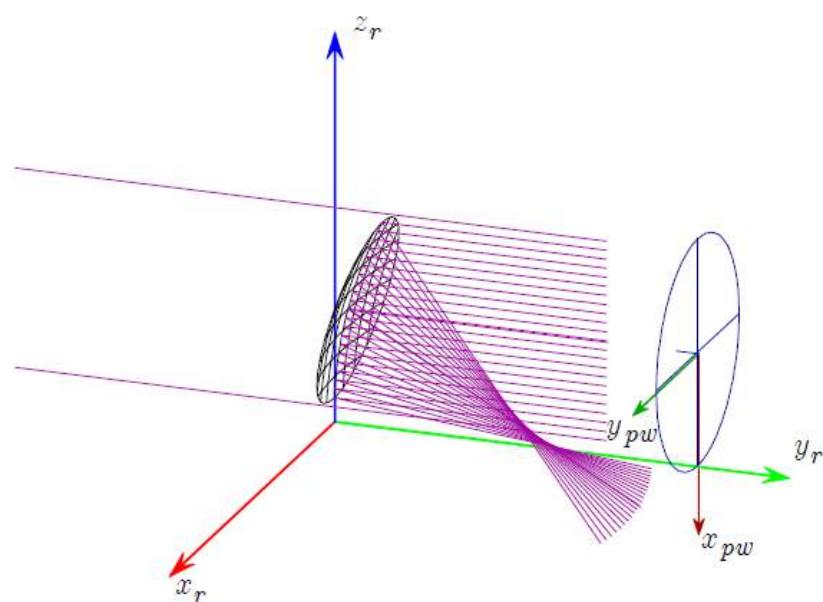


Figure 1

Illustration of plotting a plane wave. See the text for explanations.

CURRENTS PLOT (currents_plot)

Purpose

The class *Currents Plot* defines and plots currents specified in an object of a class under *PO Analysis*. The currents must have been computed by a *Get Currents* command in which the target object is of a class under *PO Analysis*. Furthermore, the attribute *file_name* in the *PO Analysis* object must specify the file where the computed currents are stored.

Links

Classes→*Plot Objects*→*Source Plot*→*Currents Plot*

Remarks

Syntax

```
<object name> currents_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    symbol_type          : <si>,
    ptd_symbol_type      : <si>,
    symbol_size          : <rl>,
    weight               : <si>
)
where
<n> = name of an object
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of a class under *PO Analysis*. The name of the reference may be 'all' which comprises references to all *PO Analysis* objects of the project. It is the current or power of the specified (or all) *PO Analysis* objects, which are plotted. The currents must have been calculated previously and stored on a file. If more than one frequency is specified in the *PO Analysis* object then the currents are plotted for the first frequency.

Symbol Type (*symbol_type*) [item from a list of character strings], default: **direction**.

It is here possible to indicate the type of symbol, which shall be applied to illustrate the physical optics (PO) surface currents (see also the Remarks below). The symbol is plotted at the positions of the currents.

direction

The currents directions and magnitudes (absolute value in x,y,z direction) are indicated by a scaled polarisation ellipse.

power

The power of the currents are shown by a scaled cross.

position_cross

The positions of the currents are shown by a cross in three dimensions.

off

The PO currents are not plotted.

Ptd Symbol Type (*ptd_symbol_type*) [item from a list of character strings], default: **off**.

The type of symbol, which shall be applied to illustrate the PTD currents, may be specified here.

off

The PTD currents are not plotted.

position_cross

The positions of the currents are shown by a cross in three dimensions.

Symbol Size (*symbol_size*) [real number with unit of length], default: **0**.

The maximum size of the symbol used to illustrate the current (or its position). The symbols are drawn on the plot of the scatterer and shall therefore be given a length comparable to the size of this.

Weight (*weight*) [item from a list of character strings], default: **off**.

Controls the use of the integration weights in the plot but only when the attribute Symbol Type is specified to 'power' or 'direction':

off

The plotted currents are not multiplied by the integration weights i.e. it is the amplitude of the current itself which is plotted.

on

The plotted currents are multiplied by the integration weights.

Command Types

The available commands for generating plots are described in [Plot Settings \(Obsolete\)](#).

Remarks

Examples on plots with [Currents Plot](#) applying the different possible values of the attribute *symbol_type* are shown in the following figures. The applied example is a simple front-fed reflector with a feed. Also plotted are the coordinate systems of the feed and of the reflector. The currents induced by the feed illumination on the reflector are illustrated in blue. The reflector radius is 20 m and the applied *symbol_size* is 2 m; the *weight* is specified to 'on'.

In the first figure *symbol_type* is specified to 'direction' and small polarisation ellipses are shown. These appear as lines as the applied feed is linearly polarised.

The integration weights consist of two factors. The first factor represents the integration area. A current at a position representing a small integration area will therefore appear small compared to a current representing a large area. The second factor of the integration weights concerns the integration itself whereby currents near the limits of the integration interval will have less weights than those centrally in the integration interval.

The influence of the integration weights is visible in the first plot and in the next plot where *symbol_type* is specified to 'power'. Near the centre of the reflector the integration weights are small (small areas and near the lower limit of the integration interval along radius) leading to small symbols. The weight are also small along the periphery (low incident illumination and near the upper limit of the integration interval along radius) leading to small symbols as well.

The drawings are plotted in a perspective projection. However, in general it is most favourable to plot the currents in an aperture plane, i.e. as seen from the top along the antenna axis, the z_r -axis.

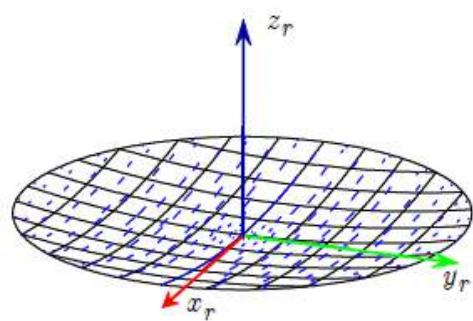
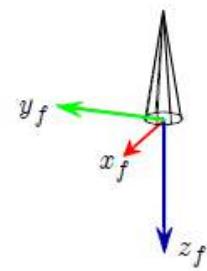


Figure 1 *symbol_type*: direction

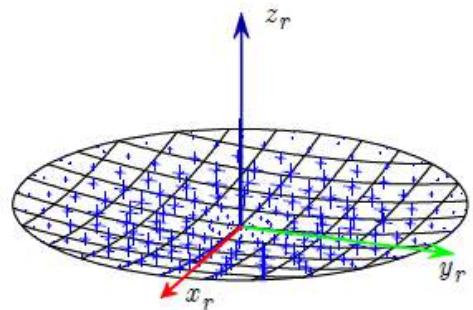
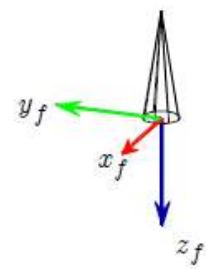


Figure 2 *symbol_type*: power

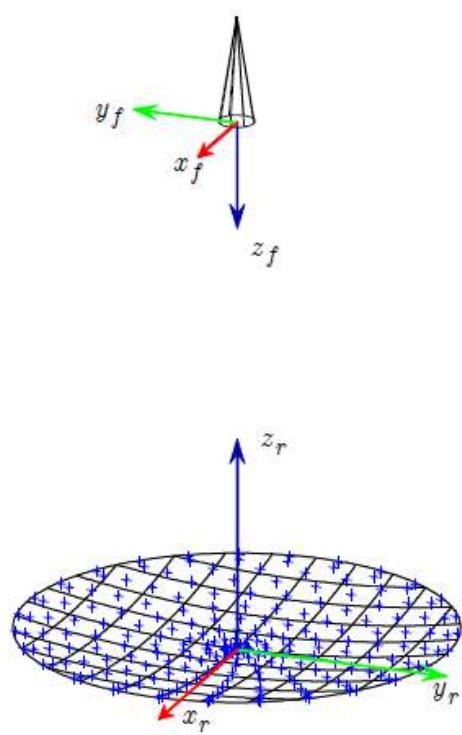


Figure 3 *symbol_type: position_cros*

GTD PLOT (gtd_plot)

Purpose

The class **GTD Plot** specifies the plotting of GO- and GTD-rays traced through a system of scatterers. Each ray originates at a source, passes a number of reflection or diffraction points on the system of scatterers, and terminates at a field point. The source, the system of scatterers and the type of rays considered are defined by an object of the class **GTD Analysis**. The field points are defined by a **Field Storage** object. The part of the ray to be plotted is specified by the user.

Links

[Classes](#)→[Plot Objects](#)→[Source Plot](#)→**GTD Plot**

Remarks

Syntax

```
<object name> gtd_plot
(
    plot_status           : <si>,
    field_points          : ref(<n>),
    ray_path_range        : struct(start:<rl>, end:<rl>),
    objects               : sequence(ref(<n>), ...),
    colour_shifts         : <i>
)
where
<i> = integer
<n> = name of an object
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Field Points (*field_points*) [name of an object].

Reference to a **Field Storage** object which specifies field points toward which the rays aim. Near-field points as well as far-field points may be specified.

Ray Path Range (*ray_path_range*) [struct].

Defines the part of the rays to be plotted (see the Remarks below).

Start (*start*) [real number with unit of length], default: **0**.

The distance measured along the rays from the source point to the point on each ray, where the plotting of the rays starts. The value must be positive or zero.

End (*end*) [real number with unit of length], default: **0**.

The distance along the rays from the source point to the point on each ray, where the plotting of the ray ends. The value must be greater than or equal to *start*.

Select GTD objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of a class under *GTD Analysis*. The name of the reference may be 'all' which comprises references to all *GTD Analysis* objects of the project. It is the rays determined by the specified (or all) *GTD Analysis* objects, which are plotted. If more than one frequency is specified in the *GTD Analysis* object then the rays are plotted for the first frequency.

Colour Shifts (*colour_shifts*) [integer], default: **-13**.

Defines the colour of the plotted ray paths. The actual colours depend on the operating system. if ≤ 0 : The same colour is used for the entire ray. if > 0 : The colour of the ray is changed at each reflection or diffraction. The number of colours used is the value of Colour Shifts.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

The attributes of the *GTD Plot* are in the GUI controlled from the 3D-View Settings (Visualization Objects) which is opened by right-clicking the 3D-View.

The values of the *ray_path_range* define the part of the rays to be plotted. Each ray is traced from the source specified in the *GTD Analysis* object through the scattering system also defined in the *GTD Analysis* object to the field points specified in the given *Field Storage* object.

The rays are traced but not plotted until the *start*-value of *ray_path_range* has been reached, and then traced and plotted until the field point or the *end*-value of *ray_path_range* has been reached. The rays are not plotted beyond a specified near-field point. This implies that if the ray-path length to a near-field point is shorter than specified by *start* then no part of that ray is plotted.

In the following figure an example on rays plotted by *GTD Plot* is given. A small horn illuminates a plane, 20λ by 20λ reflector at a distance of 50λ and a *Single-Reflector GTD* object describes diffraction in the right edge of

the reflector. We want to draw ray paths from the horn to near-field points upon a circle 70λ from the horn and 14° off the axis and separated 20° in azimuth.

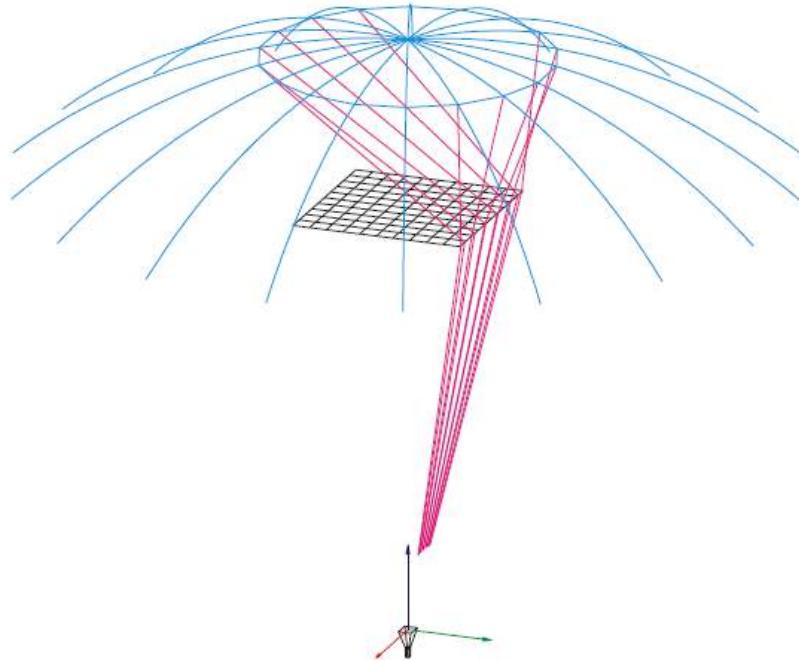


Figure 1 Example of GTD rays.

The near-field points are described in a *Field Storage* object as a *Spherical Cut* of type conical cut having a constant angle $\theta = 14^\circ$ from the common axis of symmetry. This cut is shown as a blue circle in the figure. In order to obtain a visual effect of the sphere of radius 70λ additional polar cuts with the spacing $\Delta\phi = 20^\circ$ are shown and the near-field points are the intersections between these cuts and the mentioned circle.

The ray paths are plotted with *GTD Plot* as shown in magenta. The plots starts 10λ from the feed, diffracts in the specified edge and ends at the field points (a large value of *end* is specified in attribute *ray_path_range*). It is seen that not all field points are reached by a ray simply because a ray of this type of diffraction does not exist for those field points with the actual length of the reflector edge. It is also observed that the points of diffraction for rays to the right part of the circle deviates slightly from those for rays to the left part of the circle.

TABULATED PLANAR SOURCE PLOT (tabulated_planar_source_plot)

Purpose

The class *Tabulated Planar Source Plot* specifies the details for plotting of objects of the class *Tabulated Planar Source*.

Links

Classes→*Plot Objects*→*Source Plot*→*Tabulated Planar Source Plot*

Remarks

Syntax

```
<object name> tabulated_planar_source_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    lines                 : <i>
)
where
<i> = integer
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Tabulated Planar Source*.

The name of the reference may be 'all' which comprises references to all *Tabulated Planar Source* objects of the project. It is the specified (or all) objects, which are plotted.

Lines (*lines*) [integer], default: **5**.

If a *Tabulated Planar Source* grid file is read, lines indicate the number of lines plotted along *x* and along *y* in a regular grid. If a cut file is read, lines indicate the number of radial lines drawn from the origin of the *Tabulated Planar Source* coordinate system. A circular periphery is also drawn.

Command Types

The available commands for generating plots are described in [Plot Settings \(Obsolete\)](#).

Remarks

The [Tabulated Planar Source](#) reads a file for which the *file_type* is either 'cut' or 'grid'.

When the *file_type* is 'cut', the source is plotted as radial lines in the *xy*-plane of the [Tabulated Planar Source](#) from the origin to a circular rim defined by the length of the cuts.

When the *file_type* is 'grid', the source is plotted as a rectangular grid of lines along *x* and *y* in the *xy*-plane of the coordinate system of the [Tabulated Planar Source](#). The limits of the grid are identical to the grid limits read in the file.

Note, that the plotted lines are given by the attribute *lines* and not necessary reflects the number or the positions of the field points given in the file.

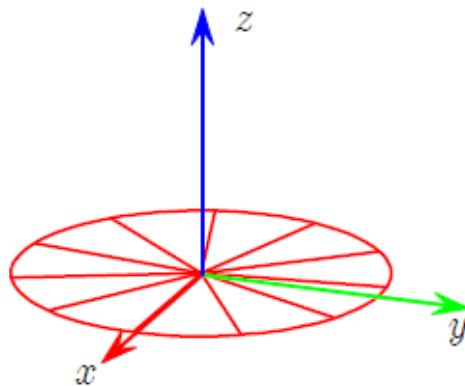


Figure 1

Plots of a [Tabulated Planar Source](#) based on *file_type* 'cut' as they appear in the coordinate system of the source. The attribute *lines* is specified to 11.

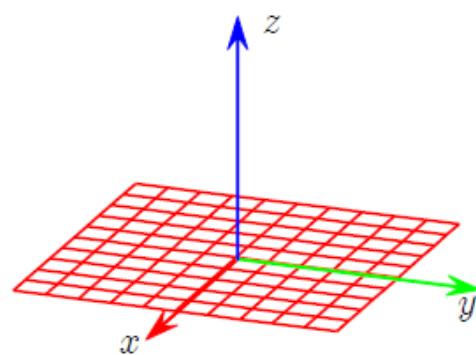


Figure 2

Plots of a *Tabulated Planar Source* based on *file_type* 'grid' as they appear in the coordinate system of the source. The attribute *lines* is specified to 11.

PLOT FEED AS CONICAL HORN (plot_feed_as_conical_horn)

Purpose

The class *Plot Feed as Conical Horn* specifies the plot of a general circular feed horn consisting of two connected cones of which the one usually will be specified as a circular cylinder representing the feed waveguide.

It is possible to draw a single cone only, or to draw a circular cylinder illustrating an open-ended waveguide.

Links

[Classes](#)→[Plot Objects](#)→[Source Plot](#)→*Plot Feed as Conical Horn*

Remarks

Syntax

```
<object name> plot_feed_as_conical_horn
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    radius_front          : <rl>,
    radius_middle          : <rl>,
    radius_back            : <rl>,
    length_front           : <rl>,
    length_back             : <rl>,
    length_centre           : <rl>,
    lines                  : <i>
)
where
<i> = integer
<n> = name of an object
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Feed* or class *Array*. The name of the reference may be 'all', which comprises references to all *Feed* objects and *Array* objects of the project. It is the specified (or all) objects, which are plotted. When an array is specified then all the feed elements of the array object are plotted as specified below. If 'all' is specified then feeds of an array will only be drawn as part of the array and not on their own. This is to avoid having these feeds drawn twice. If such feeds shall be drawn on their own, they must be specified individually in the sequence.

Radius Front (*radius_front*) [real number with unit of length], default: **0**.

Radius at the front end of the feed cone. This is usually the aperture of the feed horn; see the drawing below.

Radius Middle (*radius_middle*) [real number with unit of length], default: **0**.

Radius at the middle of the cone. This is typically at the throat of the feed.

Radius Back (*radius_back*) [real number with unit of length], default: **0**.

Radius at the back end of the cone. This radius is typically that of the feed waveguide.

Length Front (*length_front*) [real number with unit of length], default: **0**.

Length of the front cone (the flared horn).

Length Back (*length_back*) [real number with unit of length], default: **0**.

Length of the back cone (the feed waveguide).

Length Centre (*length_centre*) [real number with unit of length], default: **0**.

The feed cone is displaced on the plot by this amount in direction of the *z*-axis defined in the *Feed* object. For a usual feed horn, the phase centre is found some distance behind the aperture. If Length Centre is specified to this distance the feed will be positioned correctly in the plot, with its phase centre at the origin of the feed coordinate system.

Lines (*lines*) [integer], default: **8**.

Number of lines to be drawn from front to back along the surface of the cones.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

The attributes of the *Plot Feed as Conical Horn* are in the GUI controlled from the 3D-View Settings (Visualization Objects) which is opened by right-clicking the 3D-View.

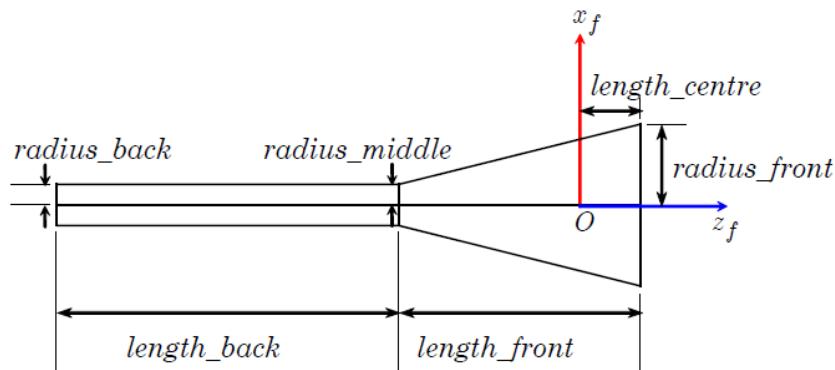


Figure 1

Illustration of *Plot Feed as Conical Horn* with the front cone, the tapered horn, at right and the back cone, the feed waveguide, at left. Here, Radius Back is the same as Radius Middle. The feed coordinate system illustrated by the axes x_f and z_f has origin at O, which is the phase centre of the feed horn.

The parameters for the definition of the cones are shown in Figure 1.

Special cases may be drawn by specifying some radii or lengths to zero. For example, an open-ended waveguide may be drawn by specifying Radius Front to the same value as Radius Middle and Length Front to zero.

The feed may also be drawn by *Feed Plot* whereby it will be plotted, if possible, according to its geometrical definition.

CURRENTS COLOUR PLOT (currents_colour_plot)

Purpose

Objects of class *Currents Colour Plot* are used to visualize computed currents in the plot of the antenna geometry. The surfaces of the scatterer with the currents are visualized with colours showing the intensity of the currents. The currents must have been computed by a *Get Currents* command in which the target object was of class *MoM* and where a file name was specified in the attribute *colour_plot_file*.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Plot Objects](#)→[Source Plot](#)→*Currents Colour Plot*

Remarks

Syntax

```
<object name> currents_colour_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    component             : <si>,
    component_coor_sys   : ref(<n>),
    current_type          : <si>,
    resolution            : <i>,
    scaling               : <si>,
    scale_min              : <r>,
    scale_max              : <r>,
    scale_type             : <si>,
    show_lines             : <si>,
    frequency_index        : <i>,
    beam_index              : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (objects) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class **MoM** which have generated the currents to be plotted. The name of the reference may be 'all' which comprises references to all **MoM** objects of the project. It is the currents of the specified (or all) **MoM** objects, which are plotted. The currents must have been calculated previously and stored on a colour plot file.

Component (component) [item from a list of character strings], default: **total_current**.

Specifies the current component to be plotted. The following may be specified:

total_current

The total currents are plotted.

x_component

The *x*-component of the currents is plotted.

y_component

The *y*-component of the currents is plotted.

z_component

The *z*-component of the currents is plotted.

Component Coor Sys (component_coor_sys) [name of an object], default: **blank**.

Reference to a **Coordinate Systems** object which specifies the coordinate system used in attribute component above. It has no effect if component is set to 'total_current'.

Current Type (current_type) [item from a list of character strings], default: **electric**.

Specifies which kind of currents is plotted:

electric

The electric currents are plotted.

magnetic

The magnetic currents are plotted.

Resolution (resolution) [integer], default: **1**.

Specifies the resolution of the plot. The default value 1 implies that the colours in the OpenGL plots are linearly interpolated between the nodes of each patch. This is a reasonable resolution sufficient for most purposes. If a value larger than 1 is specified, the currents are calculated at more points and the colours are linearly interpolated between these points. The value of Resolution specifies the number of interpolation segments between two nodes.

Scaling (*scaling*) [item from a list of character strings], default: **autoscale_all_components**

Specifies how the minimum and maximum amplitude values used in the colour scheme are found. The colours change from blue through green and yellow to red with increasing amplitude values. If the attribute `current_objects` references more than one object, the automatic scaling will be based on the global minimum and maximum values encountered in all the specified objects. The following values may be specified:

`autoscale_all_components`

The minimum and maximum amplitude values are determined automatically from the amplitude of all components independent on which individual component is selected by the attribute Component. This allows visualising different components, e.g., co- and cross-polarised currents, based on the same scaling. The minimum and maximum values are written in the standard output file.

`autoscale_selected_component`

The minimum and maximum amplitude values are determined automatically from the amplitude of the components selected by the attribute Component. The minimum and maximum values are written in the standard output file.

`user_defined`

The minimum and maximum amplitude values are specified by the user in the attributes Scale Min and Scale Max.

Scale Min (*scale_min*) [real number], default: **0**.

The minimum value used in the colour scheme when Scaling is specified to 'user_defined'.

Scale Max (*scale_max*) [real number], default: **0**.

The maximum value used in the colour scheme when Scaling is specified to 'user_defined'.

Scale Type (*scale_type*) [item from a list of character strings], default: **linear**.

Specification of the applied scale which may be:

linear

The colour coding is based on a linear scaling. If the attribute scaling is specified to 'user_defined' then Scale Min and Scale Max must be given in the standard GRASP unit of watt¹.

db

The colour coding is based on a logarithmic scaling (linear in dB). If the attribute Scaling is specified to 'user_defined' then Scale Min and Scale Max must be given in dB.

Show Lines (*show_lines*) [item from a list of character strings], default: **on**.

Determines if the mesh used in the MoM calculation is shown in the colour plot:

on

The mesh is shown.

off

The mesh is not shown.

Frequency Index (*frequency_index*) [integer], default: **1**.

Selects the desired frequency (or wavelength) of the **MoM** objects specified in attribute *current_objects* when the MoM currents have been computed for more than one frequency (wavelength). The currents which are plotted are for frequency (wavelength) number Frequency Index in the respective colour plot files determined by the specified **MoM** objects which are plotted. It is up to the user to ensure if these frequencies shall agree.

Beam Index (*beam_index*) [integer], default: **1**.

Selects the desired beam of the specified **MoM** objects (in attribute *current_objects*), when the MoM currents have been computed for more than one beam.

Remarks

The **Currents Colour Plot** shows the intensity of the currents on the surface of a scatterer. It is recommended to deactivate the plotting of the same scatterer as specified in other plot objects, e.g. a **Scatterer Plot** object or a **MoM Plot** object, as the colour plot is shown on top of the scatterer. The plotting of the scatterer is deactivated by setting the attribute *plot_status* to 'off' in the relevant objects.

Note that the MoM mesh of the scatterer can be included in the colour plot by specifying the attribute *show_lines* to 'on'.

The scaling

The currents, \vec{J} , to be plotted may be expressed in x -, y - and z -components in the coordinate system specified in attribute `component_coor_sys`:

$$\vec{J} = J_x \hat{x} + J_y \hat{y} + J_z \hat{z} \quad (1)$$

It is J_x , J_y and J_z , respectively, which is plotted when attribute `component` is specified to `x_component`, `y_component` or `z_component`. When the attribute `component` is specified to `total_current` it is the total current defined as

$$J = \sqrt{J_x^2 + J_y^2 + J_z^2} \quad (2)$$

which is plotted.

The scaling may in attribute `scale_type` be chosen to be ‘linear’ in which case the quantities J_x etc. are plotted in linear scale (in watt), or `scale_type` may be specified to ‘dB’ in which case

$$J_x^{dB} = 10 \log_{10}(J_x^2) = 20 \log_{10}(J_x) \quad (3)$$

etc. are plotted.

Examples

Examples on a *Currents Colour Plot* are shown in the following figure. A very thick polygonal stiffener (of class *Polygonal Stiffeners*) is exited by a nearby feed and the currents on the stiffener are calculated by MoM and visualised in two colour plots. The MoM mesh is shown as `show_lines` is specified to ‘on’.

In the plot to the left the default `resolution` value of 1 is applied. The patches are each defined by 16 nodes and the colours are linearly interpolated over the 3 by 3 sub-patches. If the currents vary fast over a sub-patch, the resolution is too crude and must be increased.

In the plot to the right, the `resolution` is increased to 8. The colours are now linearly interpolated over 24 by 24 sub-patches for each MoM patch, yielding a better visualization of the strong intensity of the currents close to the edge.

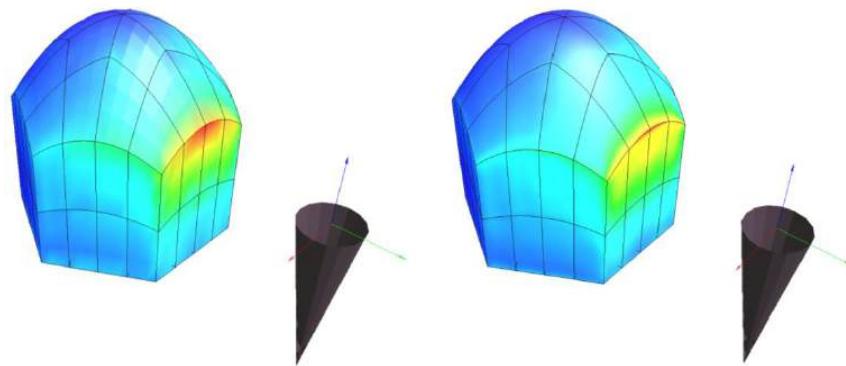


Figure 1

Two examples on current colour plots for a very wide polygonal stiffener illuminated by a horn to the right of the stiffener. In the left figure the value of *resolution* is 1, in the right figure the value is 8 giving a better impression of the fast varying currents near the edge closets to the horn.

MOM PLOT (mom_plot)

Purpose

Objects of the class **MoM Plot** specify how meshes used in MoM computations are visualised.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Plot Objects](#)→[Source Plot](#)→[MoM Plot](#)

Remarks

Syntax

```
<object name> mom_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    show_edges            : <si>,
    show_iso_lines        : <si>
)
where
<n> = name of an object
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Select MoM Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class **MoM**. The meshes generated by the specified (or all) objects of this class are plotted.

Show Edges (*show_edges*) [item from a list of character strings], default: **on**.

External edges of the mesh may be shown in red which is used to check for proper mesh connectivity. See the Remarks below. The following may be specified:

on

External edges are shown in red, other edges in black.

off

External edges are shown in black as any other edges

Show Iso Lines (*show_iso_lines*) [item from a list of character strings], default: **off**.

The isoparametric lines connect the nodes defining the patch and include the outline of the patch. The outline is always plotted while the inner isoparametric lines on the patches may be included or not according to this attribute. There are no inner isoparametric lines for patches without inner nodes. The following may be specified:

on

Isoparametric lines are plotted.

off

Isoparametric lines are not plotted.

Remarks

The *MoM Plot* shows the mesh used in the MoM calculations. It may be necessary to deactivate the plotting of the scatterer specified by the MoM object as the mesh is shown on top of the scatterer. The plotting of the scatterers is deactivated by setting the attribute *plot_status* to 'off' in the relevant *Scatterer Plot* object.

An example of plotting the MoM mesh is shown in the following two figures of a subreflector illuminated by a horn. In the top figure also the reflector is plotted while this is switched off in the bottom figure. The MoM mesh is plotted in black lines with the inner isoparametric lines plotted as light blue lines (the patches are each defined by 16 nodes, see class *MoM* for a description of nodes and patches).

Mesh connectivity

In a MoM analysis, the mesh must, patch by patch, fulfil a number of rules. The rules are listed in the description of the *MoM* class. If the scatterer analysed by MoM is a simple scatterer, e.g. a reflector, a plate, or a strut, the mesh is automatically generated by GRASP and will obey the rules. For complex scatterers of the classes *Scatterer Cluster* or *Tabulated Mesh*, it is up to the user to ensure that the rules are fulfilled.

The *MoM Plot* is useful for detecting and correcting improper mesh connectivity. If two patches in the mesh share an edge but the edge is not shared along its entire length (within a tolerance of $10^{-5}\lambda$), the MoM algorithm will detect the entire edge as being an external edge implying that the current flowing across the edge is forced to zero. The *MoM Plot* will, when *show_edges* is specified to 'on', show external edges in red and internal edges shared by two or more patches in black. This can be used to

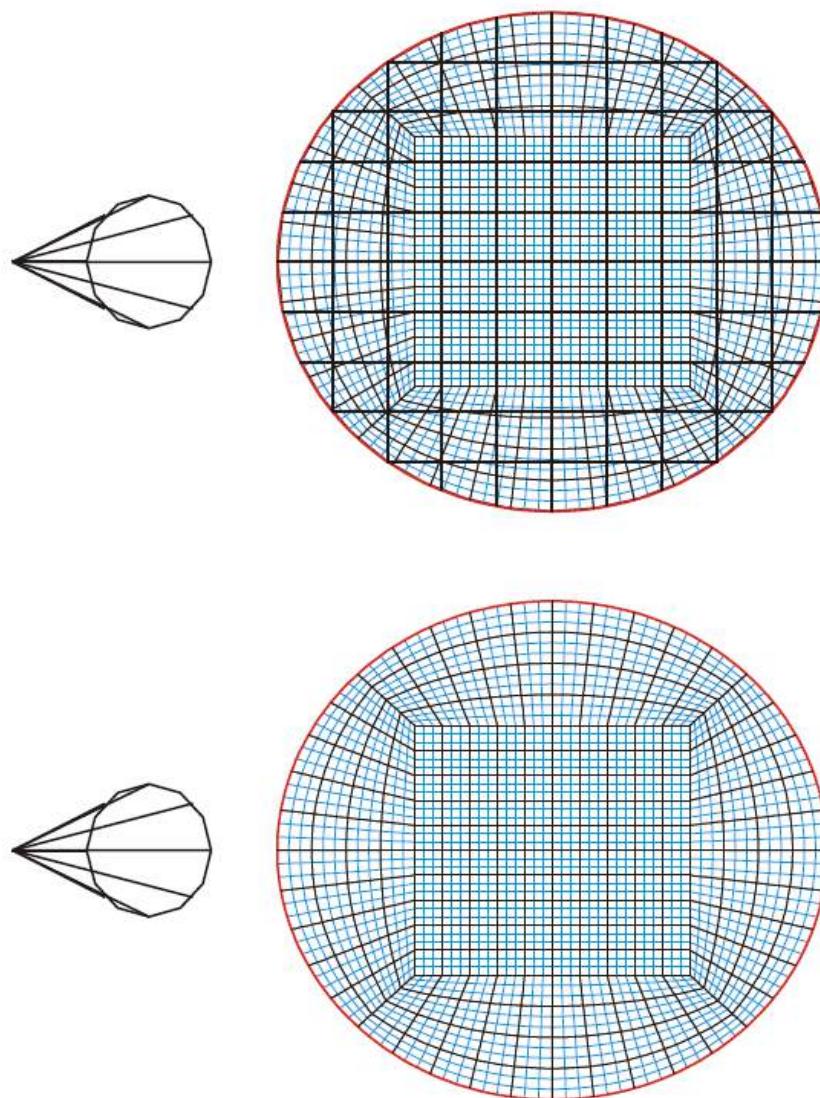


Figure 1 Two examples on the mesh on a subreflector. In the top figure both the MoM mesh and the scatterer itself are plotted, in the bottom figure only the MoM mesh is plotted.

identify edges that are misinterpreted as being external edges. An example is given in the following figure.

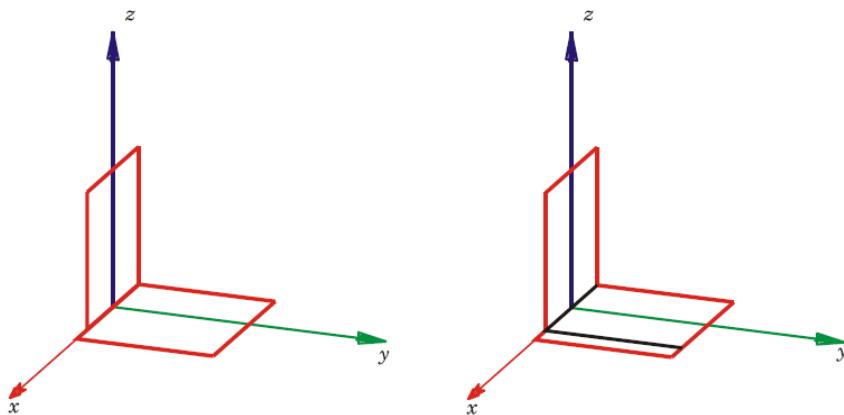


Figure 2

(left) Example on an incorrect mesh of two patches. The edge between the two patches is only partly shared which is illegal. The entire edge will be detected as an external edge and the currents flowing across the edge will be forced to zero. External edges are shown in red and the present example contains only external edges. (right) A correct mesh of the same structure as (left), constructed by three patches. Edges that are shared between two patches are shared along the entire length of the edge. The external edges are shown in red, internal edges in black. In contrast to the figure on the left, only edges that are truly external are red.

MOM SOURCE PLOT (mom_source_plot)

Purpose

Objects of the class *MoM Source Plot* define how objects of the type *MoM Source* are plotted.

This class is only available with the MoM add-on.

Links

[Classes](#)→[Plot Objects](#)→[Source Plot](#)→*MoM Source Plot*

Remarks

Syntax

```
<object name> mom_source_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    mom_source_plot_size : <rl>
)
where
<n> = name of an object
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

MoM Source Plot Size (*mom_source_plot_size*) [real number with unit of length], default: **0**.

Size of the drawn sphere indicating the source. If MoM Source Plot Size is specified to zero, a value based on the structure dimensions will be applied.

Remarks

An example of a plot of a *MoM Source* is shown in Figure 1. A helix antenna positioned on a circular plate is excited by a *Voltage Generator*. The plate

and helix are tilted with respect to the global coordinate system. A sphere divided into red and blue halves is used to indicate the location of the voltage generator. The sphere is always oriented with respect to the global coordinate system, regardless of the voltage generator's orientation. Thus both halves of the sphere are visible in this case.

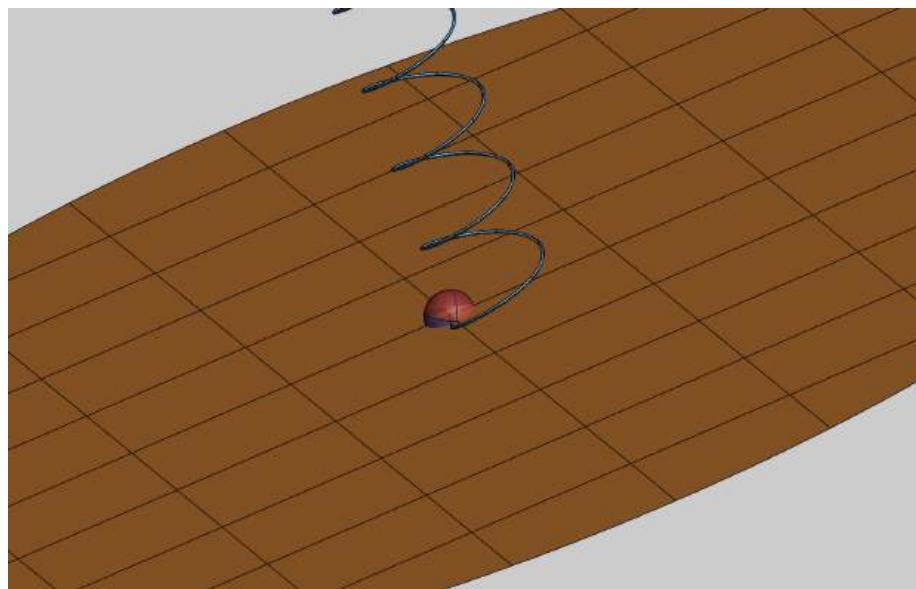


Figure 1

Illustration of an application of *MoM Source Plot*.

OUTPUT POINTS PLOT (output_points_plot)

Purpose

The class *Output Points Plot* defines how objects of the *Field Storage* classes (except of the class *Tabulated uv-Points*) shall be plotted.

The output points where the field shall be calculated may be plotted to control that they are positioned correctly with respect to the geometry. Near-field points are shown at their true position while far-field points are positioned at a finite distance selected by the user.

Links

Classes→*Plot Objects*→*Output Points Plot*

Remarks

Syntax

```
<object name> output_points_plot
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    distance              : <rl>
)
where
<n> = name of an object
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Field Storage* (except of the class *Tabulated uv-Points*). The name of the reference may be 'all' which comprises references to all *Field Storage* objects (except of the class *Tabulated uv-Points*) of the project. It is the specified (or all) objects, which are plotted.

Distance (*distance*) [real number with unit of length], default: **0**.

This attribute is needed when far-field points are plotted, since these per definition are located infinitely far away. The attribute determines the distance at which far-field points are shown in the plot. The distance is measured from the origin of the output coordinate system. The value is not used for near-field points which are drawn at their true position.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

Examples of figures drawn by *Output Points Plot* may be found under the different output objects described in the class *Field Storage*.

RAY PLOT

Purpose

By objects of the menu *Ray Plot* it is possible to control the illustration of GO rays scattered in the modelled geometry:

Rays from Point Sources

Rays from Plane Waves

Rays from Array Elements

Rays from Coordinate Systems

Note that ray tracing generated by a *GTD Analysis* object may be visualised by the class *GTD Plot* which is listed in the menu *Source Plot*.

For Gaussian feeds (*Gaussian Beam*, *Pattern Def.* and *Gaussian Beam, Near-Field Def.*) only a central ray is drawn together with a ray tube with a specified power content. The ray tube is drawn by

Gaussian Beam Tube

The here mentioned classes are in the GUI invoked from the menu 3D-View Settings (Visualization Objects) opened by a right click in the 3D-View frame.

Links

Classes→*Plot Objects*→*Ray Plot*

RAYS FROM POINT SOURCES (`rays_from_point_sources`)

Purpose

The class *Rays from Point Sources* specifies the details for plotting of rays radiated from one or more point sources given by a *Feed*. The rays are radiated in a spherical grid.

See also class *Rays from Coordinate Systems* for plotting of rays radiated from the origin of a coordinate system.

Links

[Classes](#)→[Plot Objects](#)→[Ray Plot](#)→*Rays from Point Sources*

Remarks

Syntax

```
<object name> rays_from_point_sources
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    theta_range           : struct(start:<r>, end:<r>, np:<i>),
    phi_range             : struct(start:<r>, end:<r>, np:<i>),
    ray_path_range        : struct(start:<rl>, end:<rl>),
    colour_shifts         : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Select Point Sources (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Feed*. The name of the reference may be 'all' which comprises references to all *Feed*s of the project. It is rays radiated by the specified (or all) *Feed*s, which are plotted.

theta-Range (*theta_range*) [struct].

Defines the range of the polar angle θ of the spherical grid in which the rays are emitted.

Start (*start*) [real number], default: **0**.

Start value of the polar coordinate θ .

End (*end*) [real number], default: **0**.

End value of the polar coordinate θ .

Np (*np*) [integer], default: **1**.

Number of θ -values in the grid.

phi-Range (*phi_range*) [struct].

Defines the range of the azimuthal angle ϕ of the polar grid from which the rays are emitted.

Start (*start*) [real number], default: **0**.

Start value of the azimuthal coordinate ϕ .

End (*end*) [real number], default: **0**.

End value of the azimuthal coordinate ϕ .

Np (*np*) [integer], default: **1**.

Number of ϕ -values in the grid.

Ray Path Range (*ray_path_range*) [struct].

Defines the part of the rays to be plotted (see the remarks below).

Start (*start*) [real number with unit of length], default: **0**.

The distance measured along the rays from the source point to the starting point on each ray, where the plotting of the rays is started. The value must be positive or zero.

End (*end*) [real number with unit of length].

The distance along the rays from the source point to the end-point on each ray, where the plotting of the rays ends. The value must be greater than the value of *start*.

Colour Shifts (*colour_shifts*) [integer], default: **-13**.

Defines the colour of the plotted ray paths.

if 0 : all rays are given a nominal colour.

if < 0 : another fixed colour instead of the nominal colour is used.

if > 0 : the colour of the ray is changed at each reflection. The colour of the first ray section is determined by the value of Colour Shifts.

Command Types

The available commands for generating plots are described in [Plot Settings \(Obsolete\)](#).

Remarks

The attributes of the [Rays from Point Sources](#) are in the GUI controlled from the 3D-View Settings (Visualization Objects) which is opened by right-clicking the 3D-View.

The rays from the source point (the origin of the coordinate system in which the feed is given) are defined in usual spherical coordinates by the directions

$$\hat{r} = \hat{x} \sin \theta \cos \phi + \hat{y} \sin \theta \sin \phi + \hat{z} \cos \theta \quad (1)$$

The spherical grid is defined when θ runs through the values

$$\theta_i = \theta_{start} + \Delta\theta \cdot (i - 1), \quad i = 1, 2, \dots, n_\theta \quad (2)$$

with

$$\Delta\theta = (\theta_{end} - \theta_{start}) / (n_\theta - 1) \quad (3)$$

θ_{start} and θ_{end} are the members `start` and `end`, respectively, of the attribute `theta-Range`, and n_θ is the member `np` of the same attribute.

Correspondingly, ϕ runs through the values

$$\phi_j = \phi_{start} + \Delta\phi \cdot (j - 1), \quad j = 1, 2, \dots, n_\phi \quad (4)$$

with

$$\Delta\phi = (\phi_{end} - \phi_{start}) / (n_\phi - 1) \quad (5)$$

where ϕ_{start} and ϕ_{end} are the members `start` and `end`, respectively, of the attribute `phi-Range`, and n_ϕ is the member `np` of the same attribute.

The rays are radiated from the source point or, more precisely, from the origin of the coordinate system in which the source (the feed) is defined. The ray paths may therefore be incorrect if the origin does not coincide with the phase centre of the feed.

The values of the `ray_path_range` define the length of the plotted rays. The length is measured from the origin of the coordinate system. The rays are traced but not plotted until the length has reached the `start`-value of Ray Path Range. The rays are then traced and plotted until the length has reached the `end`-value of Ray Path Range. If a ray hits an object, it is reflected following the law of specular reflection. A reflection will not interrupt the plotting of the ray. This will be continued until the ray has propagated to the path length value of `end`.

An example is given in Figure 1. The feed is defined in the feed coordinate system denoted $x_f y_f z_f$ in the figure, and the z_f -axis points towards the centre of the reflector. The feed is specified as a **Gaussian Beam, Pattern Def.**. This kind of feed is plotted by **Feed Plot** as a conical horn. The reflector is specified in the $x_r y_r z_r$ coordinate system.

The two coordinate systems are drawn by two different objects of class **Coordinate System Plot**. In the first, the `axis_length` is 100 m; in the second the `axis_length` is 10 m.

The rays are plotted for θ in the range from 0° to 15° in steps of 3° for $\phi = 0^\circ$ and $\phi = 180^\circ$ resulting in a vertical fan of rays. Further, the rays are plotted for a ray path from 10 m to 140 m. The specifications are:

```
theta-Range struct(start:0, end:15, np:5),
phi-Range: struct(start:0, end:180, np:2)
```

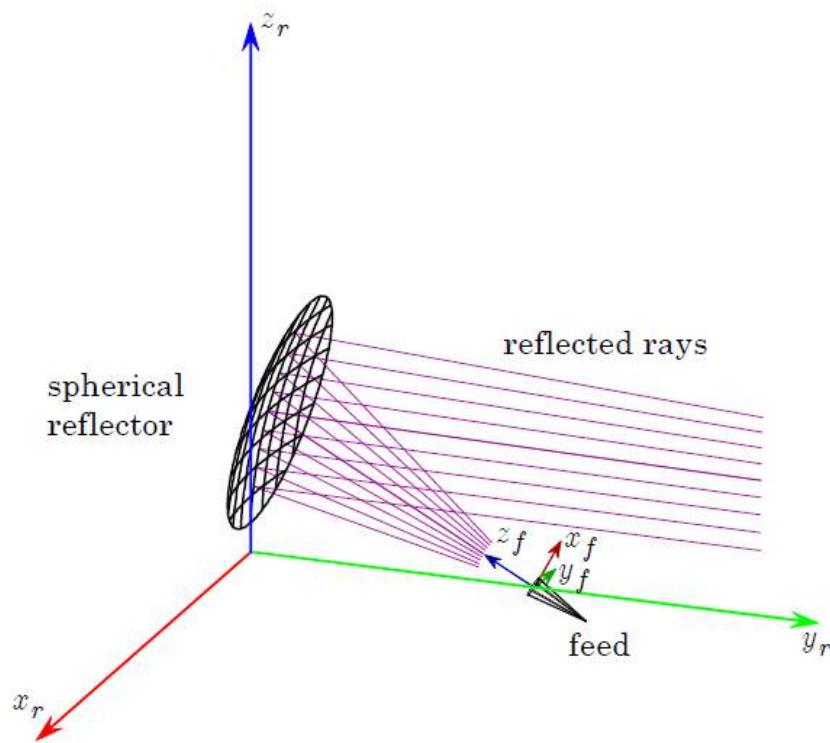


Figure 1 Illustration of rays of a feed illustrated by a simple feed horn. See the text for details.

Ray Path Range: struct(start:10 m, end:140 m)

The rays, here drawn in purple, are defined in an object of the present class *Rays from Point Sources*.

The plotted rays hit a reflector and are reflected back in direction of the positive y_r -axis. As the reflector is a spherical mirror, the reflected rays are not parallel.

As all rays start and end at the same distance from the source point, the start and the end of the rays represents phase fronts for the spherical wave (at start) and for the reflected wave (at end). More phase fronts may be included in the plot by defining more objects of class *Rays from Point Sources*, the objects having different Ray Path Ranges (e.g. start:175 m, end:200 m).

RAYs FROM PLANE WAVES (`rays_from_plane_waves`)

Purpose

The class *Rays from Plane Waves* defines how to plot rays coming from one or more objects of class *Plane Wave*. Rays are emitted in the direction of the propagation in a plane-polar grid defined in a plane perpendicular to the direction of propagation, with its centre at the centre of the circular boundary defining the plane wave.

Links

[Classes](#)→[Plot Objects](#)→[Ray Plot](#)→*Rays from Plane Waves*

Remarks

Syntax

```
<object name> rays_from_plane_waves
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    rho_range             : struct(start:<rl>, end:<rl>, np:<i>),
    phi_range              : struct(start:<r>, end:<r>, np:<i>),
    ray_path_range        : struct(start:<rl>, end:<rl>),
    colour_shifts         : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Select Plane Waves (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Plane Wave*. The name of the reference may be 'all' which comprises references to all *Plane Waves* of the project. It is rays radiated by the specified (or all) *Plane Waves*, which are plotted.

rho-Range (*rho_range*) [struct].

Defines the range of the radial coordinate ρ of the spherical grid in which the rays are emitted.

Start (*start*) [real number with unit of length], default: **0**.

Start value of the radial coordinate ρ .

End (*end*) [real number with unit of length], default: **0**.

End value of the radial coordinate ρ .

Np (*np*) [integer], default: **1**.

Number of ρ -values in the grid.

phi-Range (*phi_range*) [struct].

Defines the range of the azimuthal angle ϕ of the polar grid from which the rays are emitted.

Start (*start*) [real number], default: **0**.

Start value of the azimuthal coordinate ϕ .

End (*end*) [real number], default: **0**.

End value of the azimuthal coordinate ϕ .

Np (*np*) [integer], default: **1**.

Number of ϕ -values in the grid.

Ray Path Range (*ray_path_range*) [struct].

Defines the part of the rays to be plotted (see the remarks below).

Start (*start*) [real number with unit of length], default: **0**.

The distance measured along the rays from the reference plane of the plane wave to the starting point on each ray, where the plotting of the rays is started. The value must be positive or zero.

End (*end*) [real number with unit of length].

The distance along the rays from the reference plane of the plane wave to the end-point on each ray, where the plotting of the rays ends. The value must be greater than the value of *start*.

Colour Shifts (*colour_shifts*) [integer], default: **-13.**

Defines the colour of the plotted ray paths

if 0 : all rays are given a nominal colour.

if < 0 : another fixed colour instead of the nominal colour is used.

if > 0 : the colour of the ray is changed at each reflection.

The colour of the first ray section is determined by the value of Colour Shifts.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

The attributes of the *Rays from Plane Waves* are in the GUI controlled from the 3D-View Settings (Visualization Objects) which is opened by right-clicking the 3D-View.

The values of the `ray_path_range` are given along z of the coordinate system of the plane wave (attribute `coor_sys` in class *Plane Wave*). Thus, the rays are radiated from $z = 0$ in the direction of the plane wave. The rays are traced but not plotted until the `start`-value of Ray Path Range has been reached, and then traced and plotted until the `end`-value of Ray Path Range has been reached. If a ray hits an object, it is reflected according to the law of specular reflection. A reflection will not interrupt the plotting of the ray. This will be continued until the ray has propagated to the path length value of `end`.

This is illustrated in the following figure which shows rays of a plane wave being reflected in a spherical reflector of diameter 40 m. The reflector is defined in the shown $x_r y_r z_r$ coordinate system, the plane wave in the $x_{pw} y_{pw} z_{pw}$ coordinate system.

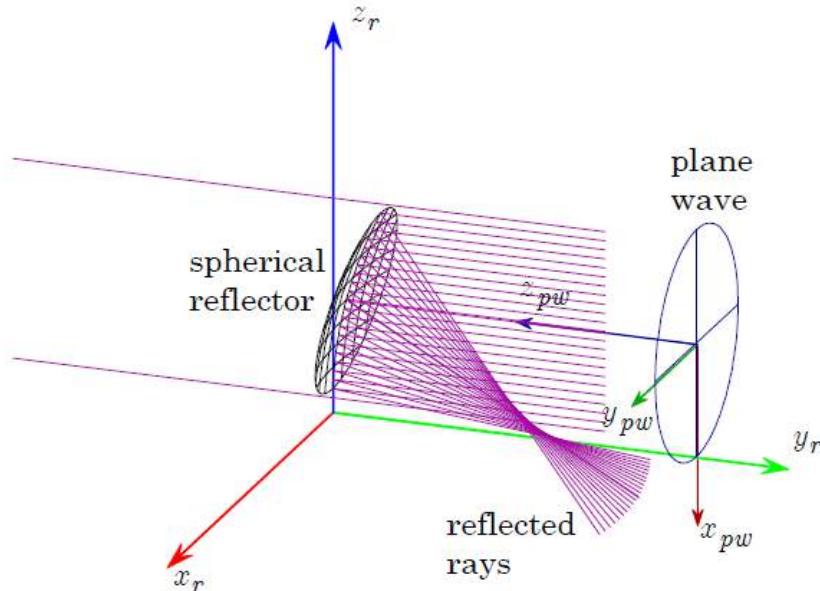


Figure 1

Illustration of rays of a plane wave being reflected in a spherical reflector. See the text for explanations.

The two coordinate systems in the figure are drawn by two different objects of class *Coordinate System Plot*. In the first, the `axis_length` is 100 m; in the second the `axis_length` is 40 m.

The rays of the plane wave, here drawn in purple, are defined in an object of the present class *Rays from Plane Waves*. 12 rays are plotted for ρ in the range from 0 to 22 m, for $\phi = 0^\circ$ and 180° resulting in a vertical fan of parallel rays. Further, the rays are plotted for a ray path from 20 m to 150 m. The specifications are:

```
rho-Range struct(start:0 m, end:22 m, np:12),
phi-Range: struct(start:0, end:180, np:2)
```

Ray Path Range: struct(start:20 m, end:150 m)

Apart from the outmost rays, the rays hit the reflector and are reflected back towards a focal region. The rays do not focus at a single point because the surface of the reflector is not paraboloidal but spherical.

As all rays start and end at the same distance from the plane-wave reference plane, the start and the end of the rays represent phase fronts for the plane wave (at start) and for the reflected wave (at end). More phase fronts may be included in the plot by defining more objects of class *Rays from Plane Waves*, the objects having different Ray Path Range (e.g. start:175 m, end:200 m).

RAYS FROM ARRAY ELEMENTS (rays_from_array_elements)

Purpose

The class *Rays from Array Elements* specifies the details for plotting of rays radiated from element sources defined in an *Array* object. The rays are radiated in a spherical grid with angles related to the element direction and orientation.

Links

Classes→*Plot Objects*→*Ray Plot*→*Rays from Array Elements*

Remarks

Syntax

```
<object name> rays_from_array_elements
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    element_ids          : sequence(<s>, ...),
    theta_range           : struct(start:<r>, end:<r>, np:<i>),
    phi_range             : struct(start:<r>, end:<r>, np:<i>),
    ray_path_range        : struct(start:<rl>, end:<rl>),
    colour_shifts         : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<s> = character string
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Select Array Elements (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Array*. The name of the reference may be 'all' which comprises references to all *Arrays* of the project.

Element IDs (*element_ids*) [sequence of character strings], default: **all**.

Sequence of names of elements in the *Arrays*. The name of the element may be 'all' which comprises references to all elements of the *Arrays* specified by Select Array Elements. It is rays radiated by the specified (or all) elements, which are plotted. In case the same element name exists in more than one *Array*, rays will be plotted from each of these elements.

theta-Range (*theta_range*) [struct].

Defines the range of the polar angle θ of the spherical grid in which the rays are emitted.

Start (*start*) [real number], default: **0**.

Start value of the polar coordinate θ .

End (*end*) [real number], default: **0**.

End value of the polar coordinate θ .

Np (*np*) [integer], default: **1**.

Number of θ -values in the grid.

phi-Range (*phi_range*) [struct].

Defines the range of the azimuthal angle ϕ of the polar grid from which the rays are emitted.

Start (*start*) [real number], default: **0**.

Start value of the azimuthal coordinate ϕ .

End (*end*) [real number], default: **0**.

End value of the azimuthal coordinate ϕ .

Np (*np*) [integer], default: **1**.

Number of ϕ -values in the grid.

Ray Path Range (*ray_path_range*) [struct].

Defines the part of the rays to be plotted (see the remarks below).

Start (*start*) [real number with unit of length], default: **0**.

The distance measured along the rays from the source point to the starting point on each ray, where the plotting of the rays is started. The value must be positive or zero.

End (*end*) [real number with unit of length].

The distance along the rays from the source point to the end-point on each ray, where the plotting of the rays ends. The value must be greater than the value of *start*.

Colour Shifts (*colour_shifts*) [integer], default: **-13**.

Defines the colour of the plotted ray paths.

if 0 : all rays are given a nominal colour.

if < 0: another fixed colour instead of the nominal colour is used.

if > 0 : the colour of the ray is changed at each reflection. The colour of the first ray section is determined by the value of Colour Shifts.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

The attributes of the *Rays from Array Elements* are in the GUI controlled from the 3D-View Settings (Visualization Objects) which is opened by right-clicking the 3D-View.

Each element of an array is given in an element coordinate system defined by the element position (x, y, z) and orientation (θ, ϕ, ψ) in the array coordinate system as specified in the Array object.

The rays are radiated from the origin of the element coordinate system of each specified element. The direction of the rays are defined in usual spherical coordinates in the same individual element coordinate systems (having axes \hat{x} , \hat{y} and \hat{z}) by the directions

$$\hat{r} = \hat{x} \sin \theta \cos \phi + \hat{y} \sin \theta \sin \phi + \hat{z} \cos \theta \quad (1)$$

The spherical grid is defined when θ runs through the values

$$\theta_i = \theta_{start} + \Delta\theta \cdot (i - 1), i = 1, 2, \dots, n_\theta \quad (2)$$

with

$$\Delta\theta = (\theta_{end} - \theta_{start}) / (n_\theta - 1) \quad (3)$$

θ_{start} and θ_{end} are the members `start` and `end`, respectively, of the attribute `theta-Range`, and n_θ is the member `np` of the same attribute.

Correspondingly, ϕ runs through the values

$$\phi_j = \phi_{start} + \Delta\phi \cdot (j - 1), j = 1, 2, \dots, n_\phi \quad (4)$$

with

$$\Delta\phi = (\phi_{end} - \phi_{start}) / (n_\phi - 1) \quad (5)$$

where ϕ_{start} and ϕ_{end} are the members `start` and `end`, respectively, of the attribute `phi-Range`, and n_ϕ is the member `np` of the same attribute.

As the rays are radiated from the origin of the element coordinate system the ray paths may be incorrect if the origin does not coincide with the phase centre of the feed.

The values of the `ray_path_range` define the length of the plotted rays. The length is measured from the origin of the coordinate system. The rays are traced but not plotted until the length has reached the `start`-value of Ray Path Range. The rays are then traced and plotted until the length has reached the `end`-value of Ray Path Range. If a ray hits an object, it is reflected following the law of specular reflection. A reflection will not interrupt the plotting of the ray. This will be continued until the ray has propagated to the path length value of `end`. For illustration see class *Rays from Point Sources* for plotting of rays radiated from a single feed position.

RAYS FROM COORDINATE SYSTEMS (rays_from_coordinate_systems)

Purpose

The class *Rays from Coordinate Systems* specifies the details for plotting of rays radiated from a point, namely the origin of one or more specified coordinate systems. The rays are radiated in a spherical grid.

See also class *Rays from Point Sources* for plotting of rays radiated from a source point given by a *Feed*.

Links

[Classes](#)→[Plot Objects](#)→[Ray Plot](#)→*Rays from Coordinate Systems*

Remarks

Syntax

```
<object name> rays_from_coordinate_systems
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    theta_range           : struct(start:<r>, end:<r>, np:<i>),
    phi_range             : struct(start:<r>, end:<r>, np:<i>),
    ray_path_range        : struct(start:<rl>, end:<rl>),
    colour_shifts         : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Select Coordinate Systems (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Coordinate Systems*. The name of the reference may be 'all' which comprises references to all *Coordinate Systems* objects of the project. Rays are radiated and

plotted from the origins of the specified *Coordinate Systems* objects, possibly all. The rays are radiated in the spherical angles θ and ϕ in the respective coordinate systems.

theta-Range (*theta_range*) [struct].

Defines the range of the polar angle θ of the spherical grid in which the rays are emitted.

Start (*start*) [real number], default: **0**.

Start value of the polar coordinate θ .

End (*end*) [real number], default: **0**.

End value of the polar coordinate θ .

Np (*np*) [integer], default: **1**.

Number of θ -values in the grid.

phi-Range (*phi_range*) [struct].

Defines the range of the azimuthal angle ϕ of the polar grid from which the rays are emitted.

Start (*start*) [real number], default: **0**.

Start value of the azimuthal coordinate ϕ .

End (*end*) [real number], default: **0**.

End value of the azimuthal coordinate ϕ .

Np (*np*) [integer], default: **1**.

Number of ϕ -values in the grid.

Ray Path Range (*ray_path_range*) [struct].

Defines the part of the rays to be plotted (see the remarks below).

Start (*start*) [real number with unit of length], default: **0**.

The distance measured along the rays from the origin to the starting point on each ray, where the plotting of the rays are started. The value must be positive or zero.

End (*end*) [real number with unit of length].

The distance along the rays from the origin to the end-point on each ray, where the plotting of the rays ends. The value must be greater than the value of *start*.

Colour Shifts (*colour_shifts*) [integer], default: **-13**.

Defines the colour of the plotted ray paths.

if 0 : all rays are given a nominal colour.

if < 0 : another fixed colour instead of the nominal colour is used.

if > 0 : the colour of the ray is changed at each reflection. The colour of the first ray section is determined by the value of Colour Shifts.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

The attributes of the *Rays from Coordinate Systems* are in the GUI controlled from the 3D-View Settings (Visualization Objects) which is opened by right-clicking the 3D-View.

The rays from the origin are defined in usual spherical coordinates by the directions

$$\hat{r} = \hat{x} \sin \theta \cos \phi + \hat{y} \sin \theta \sin \phi + \hat{z} \cos \theta \quad (1)$$

The spherical grid is defined when θ runs through the values

$$\theta_i = \theta_{start} + \Delta\theta \cdot (i - 1), i = 1, 2, \dots, n_\theta \quad (2)$$

with

$$\Delta\theta = (\theta_{end} - \theta_{start}) / (n_\theta - 1) \quad (3)$$

θ_{start} and θ_{end} are the members `start` and `end`, respectively, of the attribute `theta-Range`, and n_θ is the member `np` of the same attribute.

Correspondingly, ϕ runs through the values

$$\phi_j = \phi_{start} + \Delta\phi \cdot (j - 1), j = 1, 2, \dots, n_\phi \quad (4)$$

with

$$\Delta\phi = (\phi_{end} - \phi_{start}) / (n_\phi - 1) \quad (5)$$

where ϕ_{start} and ϕ_{end} are the members `start` and `end`, respectively, of the attribute `phi-Range`, and n_ϕ is the member `np` of the same attribute.

The rays are radiated from the origin of a coordinate system. The ray paths may therefore be incorrect if the origin does not coincide with the phase centre of the source the radiation of which the plotted rays shall represent.

The values of the `ray_path_range` define the length of the plotted rays. The length is measured from the origin of the coordinate system. The rays are traced but not plotted until the length has reached the `start`-value of Ray Path Range. The rays are then traced and plotted until the length has reached the `end`-value of Ray Path Range. If a ray hits an object, it is reflected following the law of specular reflection. A reflection will not interrupt the plotting of the ray. This will be continued until the ray has propagated to the path length value of `end`.

An example is given in the figure showing rays reflected in a paraboloidal reflector given in a reflector coordinate system (x_r, y_r, z_r). The reflector diameter is 40m, focus is at F , focal length 50m. A focal coordinate system denoted (x_f, y_f, z_f) is defined with origin at F . The z_f -axis points towards the centre of the reflector, the y_f -axis points in opposite direction of x_r .

The two coordinate systems are drawn by two different objects of class *Coordinate System Plot*. In the first, the `axis_length` is 100m; in the second the `axis_length` is 10m.

The rays are defined to be radiated from the origin of the focal coordinate system, which is listed in the attribute `coor_sys_names`. The rays shall

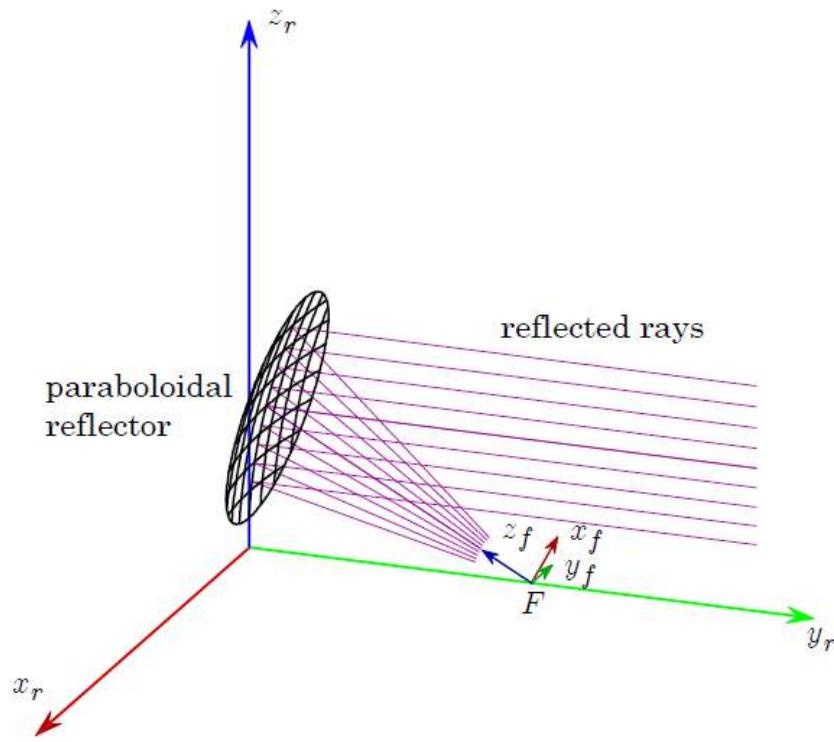


Figure 1 Illustration of rays radiated from the origin of the $x_f y_f z_f$ -coordinate system. See the text for details.

radiate in a polar θ -range from 0° to 15° in steps of 3° for $\phi = 0^\circ$ and $\phi = 180^\circ$ resulting in a fan of rays in the $z_f x_f$ -plane. The rays are plotted for a ray path from 10m to 140m. The specifications are:

```
theta-Range struct(start:0, end:15, np:5),
phi-Range: struct(start:0, end:180, np:2)
Ray Path Range: struct(start:10 m, end:140 m)
```

The plotted rays hit the reflector and are reflected back in direction of the positive y_r -axis.

As all rays start and end at the same distance from the origin of the focal coordinate system, the start and the end of the rays represents phase fronts for the spherical wave (at start) and for the reflected wave (at end). More phase fronts may be illustrated in the plot by defining more objects of class *Rays from Coordinate Systems*, the objects having different Ray Path Range (e.g. start:175 m, end:200 m).

GAUSSIAN BEAM TUBE (gaussian_beam_tube)

Purpose

The class *Gaussian Beam Tube* specifies the details for plotting of a beam tube radiated from one or more sources given by a Gaussian feed definition, i.e. one of the two *Feed* types *Gaussian Beam*, *Pattern Def.* and *Gaussian Beam, Near-Field Def.*

The beam which is plotted consists of a central ray with a surrounding beam tube. The diameter of the plotted tube follows the power level within the tube and is thus most narrow at the beam waist.

Links

Classes→*Plot Objects*→*Ray Plot*→*Gaussian Beam Tube*

Remarks

Syntax

```
<object name> gaussian_beam_tube
(
    plot_status           : <si>,
    objects               : sequence(ref(<n>), ...),
    beam_path_range      : struct(start:<rl>, end:<rl>,
                                    number_of_cross_sections:<i>),
    plot_ray_tube        : <si>,
    phase_front_cuts     : <i>,
    power_level          : <r>,
    obsolete_ray_tube_facets : <i>
)
where
<i> = integer
<n> = name of an object
<r> = real number
<rl> = real number with unit of length
<si> = item from a list of character strings
```

Attributes

Plot Status (*plot_status*) [item from a list of character strings], default: **on**.

Controls the status of the plot.

on

The specified plot is generated.

off

The specified plot is not generated.

Objects (*objects*) [sequence of names of other objects], default: **all**.

Sequence of references to objects of the class *Gaussian Beam, Pattern Def.* or *Gaussian Beam, Near-Field Def.*. The name of the reference may be 'all' which comprises references to all *Gaussian Beam, Pattern Def.* and *Gaussian Beam, Near-Field Def.* objects of the project. It is the specified (or all) objects, which are plotted.

Beam Path Range (*beam_path_range*) [struct].

Defines the range along which the beam tubes shall be plotted. The beam tube is plotted along the positive *z*-axis of the coordinate system in which the Gaussian feed is defined. A reflection occurs when a scatterer is reached (see the remarks below).

Start (*start*) [real number with unit of length], default: **0**.

The *z*-value at which the plot of the beam starts. The value must be positive or zero.

End (*end*) [real number with unit of length].

The *z*-value at which the plot of the beam ends. The value must be greater than the value of start.

Number of Cross Sections (*number_of_cross_sections*) [integer].

Number of points at which cross sections of the beam tube are plotted. The first point is at Start and the last point is at End. The value must be positive or zero.

The cross sections are visualized as phase-front cuts according to the value of attribute Phase Front Cuts which then shall be positive.

Plot Ray Tube (*plot_ray_tube*) [item from a list of character strings], default: **on**.

Controls the plot of the beam tube.

on

Beam tubes will be plotted. The number of lines suspending the tube is proportional to the Resolution Index specified in the 3D-View Settings window in the GUI (opened by right clicking the 3D-View).

off

Beam tubes will not be plotted

Phase Front Cuts (*phase_front_cuts*) [integer], default: **6**.

A phase front is plotted at each cross section of the tube as specified by Number of Cross Sections in attribute Beam Path Range. The phase fronts are illustrated by radial cuts (like spokes in a wheel) from the centre ray to the radial extent of the tube. The number of radial cuts in the cross sections is specified by this attribute. Examples are given in the Remarks below. The value should be at least 1. Otherwise no phase fronts are plotted.

Power Level (*power_level*) [real number], default: **-8.68589**.

The beam tube is plotted with a width such that the power level at the beam wall is Power Level (dB) relative to the level at the centre ray.

Ray Tube Facets (Obsolete) (*obsolete_ray_tube_facets*) [integer], default: **16**.

This attribute is no longer used. Please use Plot Ray Tube to turn on/off the plot of the beam tube. Use the Resolution Index in 3D-View Settings of the GUI to increase the resolution of the plot.

Command Types

The available commands for generating plots are described in *Plot Settings (Obsolete)*.

Remarks

The attributes of the *Gaussian Beam Tube* are in the GUI controlled from the 3D-View Settings which is opened by right-clicking the 3D-View.

The central ray

The beam tube is plotted as following the central ray of the Gaussian beam. It is not possible to draw other rays. See *Ray Plot* for drawing other rays types.

Width of the tube

The width of the tube is given by the radial distance from the centre ray at which the field amplitude is decreased to a level Power Level dB relative the on-axis level. The amplitude is decreased to $1/e$ of the on-axis field for Power Level

$$20 \log(1/e) = -8.68589 \text{ dB} \quad (1)$$

which is the default value of Power Level. The corresponding beam radius is denoted w . Satisfactorily results are achieved when scatterers are designed with an aperture diameter of $D = 4w$ (c.f. P.F. Goldsmith, 'Quasioptical Systems', IEEE Press, 1998, Section 11.2.6.). It is then useful to plot the beam tubes with a corresponding diameter to see if the design goal is fulfilled. An aperture diameter of $D = 4w$ corresponds to a beam radius $\rho = D/2 = 2w$ and as the field amplitude in a cross section of the Gaussian beam is given by

$$e^{-(\rho/w)^2} \quad (2)$$

we find the value of Power Level to

$$20 \log(e^{-(2w/w)^2}) = 20 \log(e^{-4}) = -34.7436 \text{ dB}, \quad (3)$$

i.e. the edge illumination of the aperture is -35 dB.

Reflections in scatterers

The Gaussian beam reflects into a new Gaussian beam for which the parameters are determined on basis of the incoming beam and the surface

parameters at the reflection point for the central ray. As illustrated in the following, the plotting of the Gaussian beam continues after the reflection with unchanged plot-parameters until the specified End-value of the Beam Path Range has been reached.

It is the parameters for the central ray which determines the tube of the beam. The tube is thus drawn in full width until and as well as after the reflection point and may thus intercept the scatterer.

The beam tube plot

The beam tube may be illustrated as a transparent tube or a set of phase fronts, or both.

The tube is plotted when Plot Ray Tube is set to 'on'. The tube is plotted as transparent which means that objects inside or behind the tube are visible.

The phase fronts are plotted when the Number of Cross Sections in attribute Beam Path Range is set to a positive value. The first phase front is at the start of the ray tube and the last at the end. The phase fronts are plotted with a number of radial cuts (as spokes in a wheel) according to the value of Phase Front Cuts. This attribute shall thus be positive for the phase fronts to be visible.

Figure 1 shows a beam tube of a beam waveguide for which Plot Ray Tube is specified to 'on'. The beam tube is plotted with Number of Cross Sections set to 0 in the attribute Beam Path Range.

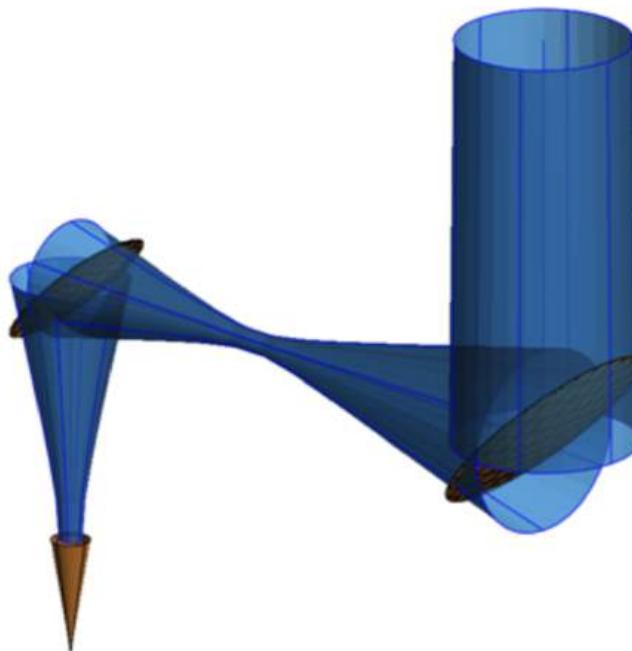


Figure 1 Beam tube for a beam waveguide radiated from a feed and reflected in two mirrors.

In the next figure, Figure 2, the same beam waveguide is plotted but now with only the phase fronts shown as cross sections of the tube. This is done by setting Plot Ray Tube to 'off', Number of Cross Sections to 31 in the attribute Beam Path Range, and Phase Front Cuts to 12.

This kind of illustrating the beam tube is suited for plane projections as shown in Figure 3 in which Phase Front Cuts is specified to 2.

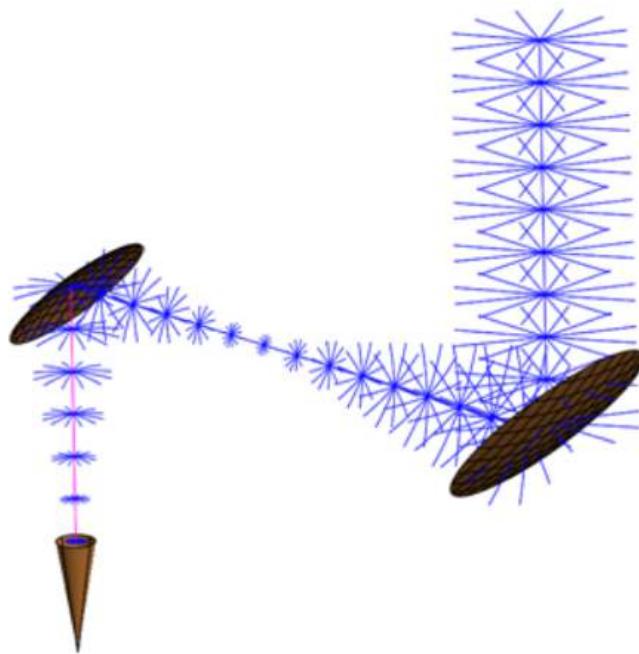


Figure 2

The same beam tube as in Figure 1, but here the beam tube is illustrated by its phase fronts.

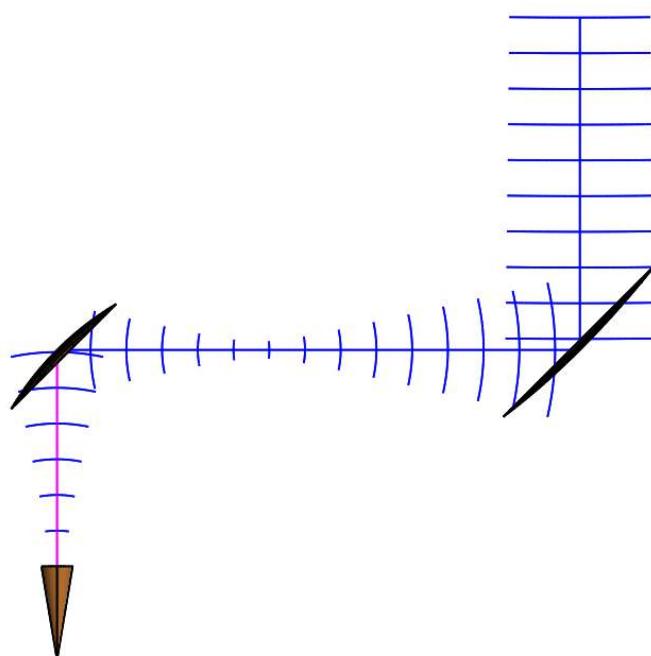


Figure 3

The phase fronts in plane projection illustrating how they change from plane at the start in the waist at the feed (in the aperture of the illustrated horn) towards spherical before the reflector, concave after the reflection towards the secondary focus of the reflector and finally, after reflection in the second reflector, to plane-wave phase fronts.

9.4 Command Types

Purpose

Commands are used in GRASP to activate calculations. A command must be of one of the available command types listed below.

The command types described here follow the syntax:

COMMAND OBJECT <target> *command* (*attributes*)

where <target> is the object which is the target for the command (i.e. the object upon which the command operates), *command* is the actual command type and *attributes* is a list of attributes which specifies the input to the command type.

Command type(s) for current computations:

Get Currents (get_currents)

Command type(s) for field and pattern computations:

Get Field (get_field)

Add Field (add_field)

Subtract Field (subtract_field)

Get Pattern (get_pattern)

Add Pattern (add_pattern)

Subtract Pattern (subtract_pattern)

Replace Pattern (replace_pattern)

Command type(s) for output of reflector parameters:

Get Reflector Data (get_reflector_data)

Command type(s) for coordinate systems:

Get Coordinate System (get_coor_sys)

Change Coordinate System Base (coor_sys_change_base)

Command type(s) for movement of geometrical objects:

Activate Movement (activate_movement)

Command type(s) for coupling determination:

Get Coupling (get_coupling)

Command type(s) for determination of blockage in a scattering system:

Check Blockage (check_blockage)

Command type(s) for exporting the MoM mesh generated by GRASP:

Write MoM Mesh (write_mom_mesh)

Command type(s) for exporting geometry to CAD files:

Export to STEP File (export_step)

Export to IGES File (export_iges)

Command type(s) for field expansion in spherical waves:

Get SWE (get_swe)

Command type(s) for field expansion in plane waves:

Get Plane Wave Expansion (get_plane_wave_expansion)

Command type(s) for executing external commands or scripts:

Execute External Command (execute_external_command)

Command type(s) for field computation through a chain of components:

Compute Chain (compute_chain)

Command types for creating, editing and deleting objects:

Create Object (create)

Set Attribute (set)

Delete Object (delete)

Command type(s) for plot of geometry:

These command types are obsolete but maintained for compatibility with older GRASP-versions. The commands are used for drawing the defined geometry, a task which is now automatically performed by the GUI.

Get XYZ Plot (Obsolete) (get_xyz_plot)

Add XYZ Plot (Obsolete) (add_xyz_plot)

Get All XYZ Plot (Obsolete) (get_all_xyz_plot)

Get OpenGL Plot (Obsolete) (get ogl_plot)

Add OpenGL Plot (Obsolete) (add ogl_plot)

Get All OpenGL Plot (Obsolete) (get_all ogl_plot)

Batch commands

When GRASP is operated in batch mode, by running a tci-file from a command prompt, several batch commands are available – commands which are not available from the GUI. When a tci-file containing such commands are opened in the GUI a warning is issued (the commands cannot be executed from the GUI) and the commands are listed as ‘General’. By double clicking a command it is possible to view and edit its content under ARGUMENTS.

Running batch commands are described in Chapter 7.

Batch command types for manipulating objects:

Rename (rename)

Copy (copy)

Files Read All (files read all)

Files Write All (files write all)

Batch command types for checking completeness of objects:

Browse Objects (browse objects)

Browse Ghosts (browse ghosts)

Browse Incomplete (browse incomplete)

Batch command type for stopping the execution of the job:

Quit (quit)

Links

Alphabetical List of Classes and Command Types

Classes

Applicable Units

File Extensions

File Formats

Appendices

Reference Section

Contents

GET CURRENTS (get_currents)

Purpose

A command of the type *Get Currents* activates the computation of the PO and PTD currents on a scatterer specified in a *PO Analysis* object or of the MoM currents on a scatterer specified in a *MoM* object.

The sources, which illuminate the scatterer and hereby generate the currents, are specified as attributes to the command. The calculated currents are stored in the *PO Analysis* object (respectively the *MoM* object) which is the target of the command. This *PO Analysis* object (*MoM* object) may be used as source for subsequent current or field computations.

It is for *PO Analysis* calculations possible to let the command determine an integration grid fine enough to obtain a specified accuracy over the regions of interest. For *MoM* calculations the user must evaluate the accuracy by varying the parameters of the *MoM* object.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> get_currents
(
    source                  : sequence(ref(<n>), ...),
    auto_convergence_of_po   : <si>,
    convergence_on_scatterer : sequence(ref(<n>), ...),
    convergence_on_output_grid : sequence(ref(<n>), ...),
    convergence_on_expansion_grid : sequence(ref(<n>), ...),
    field_accuracy           : <r>,
    max_bisections          : <i>,
    integration_grid_limit   : <si>,
    obsolete_conv_on_po_grid : sequence(ref(<n>), ...)
)
```

where

<target> = name of an object of type *BoR-MoM*; *MoM*; *Plane Wave Expansion*; *PO, Aperture in Screen*; *PO, Lens*; *PO, Multi-Face Scatterer*; *PO, Panels in Polar Grid*; *PO, Polygonal Struts (Obsolete)*; *PO, Single-Face Scatterer*; *Strut Analysis, Arbitrary Cross Section*; *Strut Analysis, Circular Cross Section*

<i> = integer

<n> = name of an object

<r> = real number

<si> = item from a list of character strings

Target

<target> [name of an object].

Reference to an object of a class under *PO Analysis* (except the obsolete class *PO Convergence (Obsolete)*) or to an object of class *MoM*. This object defines the scatterer on which the currents to be calculated flow as well as a possible file (the currents file) for storage of the currents. It is the sum of the currents induced by all the specified sources, which is calculated and stored. Any currents already contained in the object will be overwritten, and if a currents file is specified and it already exists, it will be overwritten as well.

Attributes

Source (*source*) [sequence of names of other objects].

A sequence of one or more references to objects of class *Source* shall be specified (except objects of the obsolete class *PO Convergence (Obsolete)*). It is the sources defined in this sequence, which illuminate the scatterer defined in the <target>.

Automatic Convergence of PO (*auto_convergence_of_po*) [item from a list of character strings], default: **off**.

This attribute can be used to switch off the automatic PO/PTD convergence calculations without changing the settings of the other attributes. See the Remarks section below for details.

off

Convergence tests are not carried out. The computation uses the number of PO and/or PTD currents specified in the <target>. The currents are stored in the <target>.

on

Convergence tests are carried out to determine the number of PO and/or PTD currents necessary to achieve convergence as specified below. The currents are subsequently calculated and stored in the <target>. At least one scatterer or output grid must be specified when *auto_convergence_of_po* is 'on'.

Convergence on Scatterer (*convergence_on_scatterer*) [sequence of names of other objects], default: **blank**.

A sequence of one or more references to objects of class *Scatterer* defining the scatterers on which the calculation of the field from the <target> must converge. (This attribute is not used in MoM currents calculations).

Convergence on Output Grid (*convergence_on_output_grid*) [sequence of names of other objects], default: **blank**.

A sequence of one or more references to objects of class *Field Storage* defining field points at which the calculation of the field from the <target> must converge. (This attribute is not used in MoM currents calculations).

Convergence on Expansion Grid (*convergence_on_expansion_grid*) [sequence of names of other objects].

A sequence of one or more references to objects of class *Plane Wave Expansion* defining field points at which the calculation of the field from the <target> must converge. (This attribute is not used in MoM currents calculations).

Field Accuracy (*field_accuracy*) [real number], default: **-80**.

The PO integration has converged when a small change in *po1* and *po2* will cause a pattern change less than the specified Field Accuracy. The pattern change is calculated as the complex field difference between two successive pattern calculations based on two sets of *po1* and *po2*. The Field Accuracy shall be given in dB relative to the maximum power level in the pattern. See also the Remarks below. (This attribute is not used in MoM currents calculation).

Maximum Number of Bisections (*max_bisections*) [integer], default: **5**.

The attribute Maximum Number of Bisections controls how precise *po1* and *po2* are determined. The value must be a positive integer giving the maximum number of bisections in the integration grid evaluation. If zero, no bisections are performed. See the remarks below. (This attribute is not used in MoM currents calculation).

Integration Grid Limit (*integration_grid_limit*) [item from a list of character strings], default: **on**.

This attribute is used to stop the optimisation if the field accuracy limit can not be obtained.

on

The maximum values of *po1* and *po2* are given by simple estimates for the different scatterers and then multiplied by 10. See the remarks below.

off

No maximum values for *po1* and *po2*.

Convergence on PO Grid (Obsolete) (*obsolete_conv_on_po_grid*) [sequence of names of other objects], default: **blank**.

The attribute is obsolete and retained for the reason of backward compatibility only. Instead the attribute Convergence on Scatterer shall be applied. (This attribute is not used in MoM currents calculations).

Remarks

The only attribute needed for *MoM* currents calculation is the attribute *source*. The following remarks only concern *PO Analysis* (PO and PTD) calculation.

The PO/PTD current integration grid on a specified scatterer is given by *po1* times *po2* points over the surface of the scatterer and by *ptd* points along

the rim as defined in the *PO Analysis* objects for the respective scatterers. The values of these parameters must be chosen with care as too small values will give incorrect results and too large values may give unacceptable long time of computation.

Optimum values of the parameters *po1*, *po2* and *ptd* of the target *PO Analysis* object are automatically determined when *auto_convergence_of_po* is specified to 'on'. In this case, the corresponding values specified in the <target> are not used.

The obtained converged values of *po1*, *po2* and *ptd* are stored in the currents file of the *PO Analysis* object.

When initially specifying *auto_convergence_of_po* to 'off' the user may control the convergence accuracy by changing the values of *po1*, *po2* and *ptd* of the target *PO Analysis* object, cf. the description of the method below. Estimated values are given in the respective *PO Analysis* objects.

The values of the parameters *po1*, *po2* and *ptd* depend on the specified *field_accuracy* but will also depend on the position of the points where field computations are requested. These field points shall therefore be specified. Thus, if the actual target *PO Analysis* object illuminates one or more *Scatterers* (from which the scattering shall be determined in a following step) then these shall be specified in the sequence of the attribute *convergence_on_scatterer*. Correspondingly, if the field of the actual target *PO Analysis* object shall be determined at field points defined in one or more *Field Storage* objects these shall be specified in the sequence of the attribute *convergence_on_output_grid*.

If the specified *Field Storage* object is of the *Cut* class then the field points used in the convergence process are derived in the following way with a set of polar spherical cuts as example: Between the first and the last cut (minimum and maximum ϕ -value) five convergence cuts are defined. Hence, if the user has specified cuts from $\phi = 0^\circ$ to $\phi = 90^\circ$ the program will generate convergence cuts at 0° , 22.5° , 45° , 67.5° and 90° . In each of these convergence cuts 11 points will be defined between the minimum and the maximum θ -value. In case conical cuts are defined in the object the same procedure is used, however with θ and ϕ interchanged.

If the specified *Field Storage* object is of the *Grid* class the convergence version of the grid will have five points between the minimum and maximum value in each dimension, (X, Y).

In conclusion it is not necessary to create a convergence version of the *Cut* or *Grid* object with sparse points as the program does this automatically.

Note, that at least one object must be specified in the attribute *convergence_on_scatterer* or in *convergence_on_output_grid* in order to activate the convergence. The sequence of references in the other attribute may then be empty.

Example

Consider for example a dual reflector system for which we want to determine the currents on the subreflector such that we from these currents can

calculate the illumination of the main reflector, the illumination of some struts and also the far field of the subreflector. The *PO Analysis* object describing the subreflector is now the target of the command as it is the currents on the subreflector that shall be determined. Further, the *Scatterer* objects specifying the main reflector and the struts, respectively, shall be specified in the attribute *convergence_on_scatterer* and the *Field Storage* object defining the far-field points shall be specified in the attribute *convergence_on_output_grid*.

Important

In several geometries it is possible to construct pitfalls for the convergence procedure whereby the desired accuracy is not obtained (though it looks like) or the computation time becomes unnecessary long:

Is the accuracy obtained?

In general, the auto-convergence results in a good estimate of the values of *po1*, *po2* and *ptd* for the requested accuracy. It is, however, for special geometries possible that the number of field points for the convergence test (see the above sections) is insufficient. To ensure that the desired accuracy is obtained it is therefore recommended to apply the following check.

First, a case with *auto_convergence_of_po* specified to 'on' shall be run. The hereby determined values of *po1*, *po2* and *ptd* shall be noted and the field pattern shall be determined. Next, a run shall be performed with *auto_convergence_of_po* specified to 'off' and *po1*, *po2* and *ptd* shall be specified to the noted values each added by one, and a second field pattern shall be determined. The two determined patterns shall then be subtracted (in the Results section of the GRASP GUI or by a *Subtract Pattern* command).

The procedure of introducing a slight increase of the values for the numbers of integration points will change the position of all integration points and result in a quite different integration which will only result in the same pattern (within the requested accuracy) if the integration has converged.

Is the computation time extremely long?

This may happen when a source point accidentally falls close to a field point. Consider for example the case in which a strut is illuminated by a reflector field and the strut is specified so a part of the one end of the strut becomes very close to the reflector surface. It is then possible that the source point (a PO point on the reflector surface) becomes very close to a field point (a PO point on the strut). The obtained field may then be very dependent on the integration grid whereby a very fine grid is laid out resulting in a corresponding long computation time.

To avoid such a case the geometry shall be specified with sense; in the above example the entire end of the strut should be specified to be a fraction of a wavelength from the reflector surface. Alternatively, the user may actively control the grid spacing and check the field dependence on the grid when *auto_convergence_of_po* is 'off'.

Description of the method

In the following, the applied method for determining *po1* and *po2* when *auto_*

convergence_of_po is set to ‘on’ shall be explained. Hereby also the controlling attributes are presented. Next, the method for determining the value of *ptd* is presented.

po1 and *po2* are determined independently. First, *po1* is converged with trial values 10,20,40,80,... and so forth while *po2* is fixed at the value 10. The doubling of the value of *po1* is stopped when the pattern variations between the patterns calculated for the last two tried values of *po1* deviates less than the specified *field_accuracy*. The smallest acceptable value for *po1* will be located in the interval between the second to last tried value and the third to last tried value. This interval is therefore bisected a number of times in order to determine the lowest value of *po1* satisfying the *field_accuracy* criteria, see the example below. The number of bisections is limited by the attribute *max_bisections*.

When the trial value (of the series of 10,20,40,80,...) passes above the limit given by specifying *integration_grid_limit* to ‘on’ then this limit is included as a trial value, and one more value of the series may also be tried order to evaluate the accuracy at the limit.

(Some objects have other starting points than *po1* = *po2* = 10, but the principle is the same).

During the evaluation process the field is determined at a set of points, the convergence grid. The considered field is denoted the test field

$$\vec{E}_i^{test}, \quad i = 1, 2, \dots, N \quad (1)$$

where *N* is the number of points in the convergence grid. In order to evaluate the accuracy, the test field is compared to a reference field, given at the same points

$$\vec{E}_i^{ref}, \quad i = 1, 2, \dots, N. \quad (2)$$

We will determine the maximum power level found at a point in the convergence grid, *P_{max}*

$$P_{\max} = \max_{i=1,N} (20 \log |\vec{E}_i^{test}|) \quad (3)$$

where the function *max(...)* takes the maximum value.

The accuracy is determined as the largest field deviation found in the convergence grid

$$A = \max_{i=1,N} \left(20 \log \left\{ \left| \vec{E}_i^{test} - \vec{E}_i^{ref} \right| \right\} - P_{level} \right). \quad (4)$$

The field deviation is taken relative to *P_{level}* where *P_{level}* = *P_{max}* (both are in dB) until convergence has occurred the first time, after which *P_{level}* takes the largest value of *P_{max}* met so far. At the end of the convergence *P_{level}* is thus the maximum level of the converged pattern.

This method is easiest demonstrated by an example in which we have specified the *field_accuracy* to -80 dB.

First an integration test grid with *po1*/*po2* = 10/10 is applied (this means *po1* = 10 and *po2* = 10). The maximum power level in the convergence grid is *P_{max}* = 40.15 dBi. These results are summarised in the following table.

Test-grid <i>po1/po2</i>	Ref-grid <i>po1/po2</i>	P_{\max} [dBi]	P_{level} [dBi]	A [dB]
10/10		40.15		

We know nothing about the accuracy but store the determined field values as a reference, \vec{E}_{ref} .

Next *po1* is doubled and an integration test grid with $po1/po2 = 20/10$ is applied and a new test field, \vec{E}_{test} , is calculated. The maximum power level in the output grid is unchanged, $P_{\max} = 40.15$ dBi. The accuracy, A , is determined to -49.19 according to Eq. (4).

We add the new results to the previous table and get the following:

Test-grid <i>po1/po2</i>	Ref-grid <i>po1/po2</i>	P_{\max} [dBi]	P_{level} [dBi]	A [dB]
10/10		40.15		
20/10	10/10	40.15	40.15	-49.19

The accuracy request of -80 dB has not been reached and *po1* is doubled again. The last calculation gave the best field determined so far and this is now stored as reference field \vec{E}_{ref} . A new test field is calculated and a new line is added to the table:

Test-grid <i>po1/po2</i>	Ref-grid <i>po1/po2</i>	P_{\max} [dBi]	P_{level} [dBi]	A [dB]	
10/10		40.15			
20/10	10/10	40.15	40.15	-49.19	
40/10	20/10	40.15	40.15	-209.91	< -80 dB

The two last test fields agree within the specified accuracy and we can deem them both to be correct with that accuracy. The last test field is now applied as \vec{E}_{ref} .

The smallest acceptable value for *po1* shall thus be found in the interval between the second to last tried value and the third to last tried value, here $10 \leq po1 \leq 20$. This interval is therefore bisected a number of times in order to determine the lowest value of *po1* satisfying the *field_accuracy* criteria. The number of bisections is limited by the attribute *max_bisections* which here is 5.

A test is therefore carried out for $po1 = 15$, which converges, then for $po1 = 13$ (mid point between 10 and 15 rounded to an integer), which also converges, and for $po1 = 12$, which does not converge:

Test-grid <i>po1/po2</i>	Ref-grid <i>po1/po2</i>	P_{\max} [dBi]	P_{level} [dBi]	A [dB]	
10/10		40.15			
20/10	10/10	40.15	40.15	-49.19	
40/10	20/10	40.15	40.15	-209.91	< -80 dB
15/10	40/10	40.15	40.15	-113.05	< -80 dB
13/10	40/10	40.15	40.15	-82.22	< -80 dB
12/10	40/10	40.15	40.15	-69.13	

The correct value to be used for *po1* is then $po1 = 13$. The above table is found in the output file of GRASP and so are the following tables.

Next, *po2* is converged in the same way while *po1* is kept fixed at the value 10. The convergence process runs as shown in the following table. The first

reference field is taken from the case $po1/po2 = 10/10$ in the convergence for $po1$ and is therefore not in the table.

Test-grid $po1/po2$	Ref-grid $po1/po2$	P_{max} [dBi]	P_{level} [dBi]	A [dB]	
10/20	10/10	40.15	40.15	-13.93	
10/40	10/20	40.15	40.15	-56.94	
10/80	10/40	40.15	40.15	-111.45	< -80 dB
10/30	10/80	40.15	40.15	-103.92	< -80 dB
10/25	10/80	40.15	40.15	-94.33	< -80 dB
10/23	10/80	40.15	40.15	-82.47	< -80 dB
10/22	10/80	40.15	40.15	-73.87	

$po2$ is doubled until it gets the value 80 and convergence has been reached. The value of $po2$ shall then be found in the interval between 20 and 40 and the bisection finds that the value of $po2$ shall be $po2 = 23$.

The value of ptd is derived correspondingly and the steps of convergence are shown in the following table:

Test-grid ptd	Ref-grid ptd	P_{max} [dBi]	P_{level} [dBi]	A [dB]	
20	20	-17.99	40.15	-61.55	
40	40	-18.99	40.15	-77.87	
80	80	-18.99	40.15	-131.00	< -80 dB
30	80	-18.99	40.15	-127.61	< -80 dB
25	80	-18.99	40.15	-112.85	< -80 dB
23	80	-18.99	40.15	-97.43	< -80 dB
22	80	-18.99	40.15	-90.37	< -80 dB
21	80	-18.99	40.15	-83.76	< -80 dB

The value of ptd to be used is thus $ptd = 21$.

Note, that the level to which the accuracy is compared is unchanged compared to the level applied for determining $po1$ and $po2$, $P_{level} = 40.15$ dBi. The PTD-field is thus determined to same accuracy as the PO-fields to which it will be added. If, on the other hand, the PTD-field was determined independent of the PO-field then P_{level} would equal $P_{max} = -18.99$ dBi resulting in a higher final value for ptd .

It is the experience that when the convergence has been reached then a small increase in $po1$ or $po2$ will not change the result significantly. But as long as the convergence has not been reached a small change in $po1$ or $po2$ will change the result considerably. In other words, as $po1$ (or $po2$) is increased the result fluctuate until the convergence is reached and a good result is obtained. An additional small increase in $po1$ (or $po2$, respectively) will not change the result drastically (the result is still good) but its accuracy will be improved. This effect can be exploited in a manual convergence test.

If a given accuracy can not be reached the geometry should be evaluated carefully as some unrealistic requirements may have been set up.

It is thus possible that the field accuracy may be specified to a level that cannot be achieved. In order to stop the optimisation in such a case, the integration grid parameters $po1$ and $po2$ can be limited by setting the attribute *integration_grid_limit* to 'on'. The applied maximum limits for the

grid parameters are found using the following simple estimates from the *PO Analysis* object descriptions and multiplying these by 10:

If the scatterer is a reflector with a polar integration grid, the number of points for convergence over the whole sphere can be estimated by

$$po2 = \frac{2\pi D}{\lambda} \quad (5)$$

$$po1 = \frac{po2}{2.4} \quad (6)$$

where D is the maximum reflector dimension and λ is the wavelength.

If the scatterer is a rectangular reflector, $po1$ and $po2$ give the number of points along x and y , respectively. For a non-focused aperture the integration limits are

$$po1 = 3.5 \frac{D_x}{\lambda} \quad (7)$$

$$po2 = 3.5 \frac{D_y}{\lambda} \quad (8)$$

where D_x and D_y are the reflector dimensions along x and y .

If the scatterer is a triangle, $po2$ signifies the number of points along the longest edge and $po1$ the number of points along the height perpendicular to the longest edge,

$$po2 = 3.5 \frac{L_{max}}{\lambda} \quad (9)$$

$$po1 \equiv po2 \quad (10)$$

where L_{max} is the length of the longest edge.

The accuracy of the field, given by the level *field_accuracy* in dB, corresponds to a power accuracy ε (maximum error in power),

$$\varepsilon = 10^{\text{field_accuracy}/10} \quad (11)$$

and to the requirement

$$|E_1 - E_2|^2 / P_{\max} < \varepsilon \text{ at all field points}$$

Here, E_1 and E_2 are the field values at the same field point determined by the two last integration grids. P_{\max} is here the maximum power level in the converged pattern. An accuracy requirement given as a maximum error of $\pm X$ dB at Y dB below peak ($Y > 0$) can be related to a *field_accuracy* by

$$\begin{aligned} \text{field_accuracy} &= 20 \log \left\{ [10^{(-Y+X)/20} - 10^{(-Y-X)/20}] / 2 \right\} \\ &= -Y + 20 \log \left\{ [10^{X/20} - 10^{-X/20}] / 2 \right\} \\ &\cong -Y + 20 \log \left\{ \left[1 + \frac{X}{20 \log e} - \left(1 - \frac{X}{20 \log e} \right) \right] / 2 \right\} \\ &= -Y + 20 \log \left\{ \frac{X}{20 \log e} \right\} \\ &= -Y - 18.8 \text{dB} + 20 \log \{X\} \\ \text{field_accuracy} &\cong -Y - 20 \text{dB} + 20 \log \{X\} \end{aligned} \quad (12)$$

Some examples are given in the following table.

X	Y	field_accuracy (dB)
1 dB	40 dB	-60
0.1 dB	40 dB	-80
1 dB	60 dB	-80

The value of *field_accuracy* can be derived from an accuracy requirement of $\pm X$ dB at Y dB below peak.

In the following example we assume that P_{\max} is the maximum level in dB in the calculated pattern. The pattern of a reflector with diameter $D = 40\lambda$ shall be determined with an accuracy of ± 1 dB at -60 dB. The *field_accuracy* is therefore specified to -80 dB. The directivity of the reflector antenna is, say, 38 dBi which means that P_{\max} is 38 dBi and the accuracy specification will then allow inaccuracies at a level of $38 \text{ dBi} - 80 \text{ dB} = -42 \text{ dBi}$.

The field for a side-lobe region will be determined to the same accuracy if it is calculated in the same cut as the field calculation of the main beam. But if the side-lobe region is calculated separately then the *field_accuracy* of -80 dB will result in a more accurate (and time consuming) calculation as P_{\max} within the side-lobe region has a lower value than P_{\max} of the main beam.

GET FIELD (get_field)

Purpose

A command of the type **Get Field** activates the computation of the field from a given **Source** at the points specified in a **Field Storage** object.

Links

[Command Types](#)

Syntax

```
COMMAND OBJECT <target> get_field
(
    source           : sequence(ref(<n>), ...)
)
where
<target> = name of an object of type Cylindrical Cut; Cylindrical Grid;
Planar Cut; Planar Grid; Spherical Cut; Spherical Grid; Surface Cut; Surface Grid; Tabulated uv-Points
<n>      = name of an object
```

Target

<target> [name of an object].

Reference to an object of one of the classes in **Field Storage**. This object defines the field points and the field to be calculated. Any field already contained in the object will be overwritten as will existing files.

Attributes

Source (source) [sequence of names of other objects].

References to objects of the class **Source** from which the field shall be calculated and stored. The field from all sources in the sequence will be added.

ADD FIELD (add_field)

Purpose

A command of the type *Add Field* activates the computation of the field from a given *Source* at the points specified in a *Field Storage* object. The computed field is added to the field already contained in the *Field Storage* object.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> add_field
(
    source                      : sequence(ref(<n>), ...)
)
where
<target> = name of an object of type Cylindrical Cut; Cylindrical Grid;
Planar Cut; Planar Grid; Spherical Cut; Spherical Grid; Surface Cut; Surface Grid; Tabulated uv-Points
<n>      = name of an object
```

Target

<target> [name of an object].

Reference to an object of one of the classes in *Field Storage*. This object defines the field points and the field components to be calculated. The calculated field will be added to the field already contained in the object, and it is the resulting field which will be stored. The object must exist and it must have been referred to in an earlier *Get Field* command such that it is not empty.

Attributes

Source (*source*) [sequence of names of other objects].

References to objects of the class *Source* from which the field shall be calculated. The calculated field is added to the field already stored in the target object.

SUBTRACT FIELD (subtract_field)

Purpose

A command of the type *Subtract Field* activates the computation of the field from a given *Source* at the points specified in a *Field Storage* object. The computed field is subtracted from the field already contained in the *Field Storage* object.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> subtract_field
(
    source : sequence(ref(<n>), ...)
)
where
<target> = name of an object of type Cylindrical Cut; Cylindrical Grid;
Planar Cut; Planar Grid; Spherical Cut; Spherical Grid; Surface Cut; Surface Grid; Tabulated uv-Points
<n>      = name of an object
```

Target

<target> [name of an object].

Reference to an object of one of the classes in *Field Storage*. This object defines the field points and the field components to be calculated. The calculated field will be added to the field already contained in the object, and it is the resulting field which will be stored. The object must exist and it must have been referred to in an earlier *Get Field* command such that it is not empty.

Attributes

Source (*source*) [sequence of names of other objects].

References to objects of the class *Source* from which the field shall be calculated. The calculated field is subtracted from the field already stored in the target object.

GET PATTERN (get_pattern)

Purpose

A command of the type *Get Pattern* retrieves and interpolates the radiated field within one *Field Storage* object (the attribute) to points specified in another *Field Storage* object (the target). Any field, already contained in the target *Field Storage* object will be overwritten.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> get_pattern
(
    input_field           : ref(<n>),
    factor                : struct(db:<r>, deg:<r>)
)
where
<target> = name of an object of type Cylindrical Cut; Cylindrical Grid;
Planar Cut; Planar Grid; Spherical Cut; Spherical Grid; Surface Cut; Surface Grid; Tabulated uv-Points
<n>      = name of an object
<r>      = real number
```

Target

<target> [name of an object].

Reference to an object of one of the classes in *Field Storage*. This object defines the field points and the field components to be calculated by interpolation. Any field already contained in the object will be overwritten as will existing files.

Attributes

Input Field (*input_field*) [name of an object].

Reference to another object of the class *Field Storage* from which the field shall be retrieved.

Factor (*factor*) [struct].

The retrieved field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Remarks

A command of the type *Get Pattern* retrieves and interpolates the field specified within one *Field Storage* object (given by the attribute `Input Field`) to the points specified in another *Field Storage* object (the target).

The interpolation is carried out as a cubic interpolation (see the GRASP Technical Description for details). If the target range specifies field points outside the range defined in the attribute `Input Field`, those field values will be zero.

It is thus possible by combination of commands of type *Get Pattern*, *Add Pattern* and *Subtract Pattern* to combine patterns of adjacent but not overlapping ranges. Patterns of overlapping ranges may be combined using a command of type *Replace Pattern*.

For example, PO calculation is accurate for the main beam and the near-in side lobes of a reflector antenna, but time consuming for far-out side lobes. For large angles, a fast GTD determination of the side lobes will have sufficient accuracy. Two *adjacent* patterns are then calculated by the two different methods. A final pattern is constructed by first applying *Get Pattern* to the PO calculated pattern and then adding the GTD pattern by *Add Pattern*. As the original patterns do not overlap and as values outside the ranges are inserted as zero, the resulting pattern will be the two original patterns “pasted” together.

Alternatively, a GTD pattern may be calculated over the full range. This is a fast procedure but it gives poor results over the central region. A PO pattern is then calculated here and inserted in the GTD pattern by a *Replace Pattern* command. This command replaces the GTD pattern by the PO pattern over the PO region while the GTD pattern is unchanged over the remaining region.

ADD PATTERN (add_pattern)

Purpose

A command of the type *Add Pattern* retrieves and interpolates the radiated field within one *Field Storage* object (the attribute) to points specified in another *Field Storage* object (the target). The interpolated field is added to the field already contained in the target object.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> add_pattern
(
    input_field
    factor
)
where
```

<target> = name of an object of type *Cylindrical Cut*; *Cylindrical Grid*; *Planar Cut*; *Planar Grid*; *Spherical Cut*; *Spherical Grid*; *Surface Cut*; *Surface Grid*; *Tabulated uv-Points*

<n> = name of an object
<r> = real number

Target

<target> [name of an object].

Reference to an object of one of the classes in *Field Storage*. This is the field to which another will be added. The resulting field will be stored in this target object. It is the target object which defines the field points and the field components of the resulting total field. The target *Field Storage* object must exist and it must have been referred to in an earlier *Get Pattern* or *Get Field* command such that it is not empty.

Attributes

Input Field (*input_field*) [name of an object].

Reference to another object of the class *Field Storage* from which the field shall be retrieved, interpolated and added to the field already stored in the target object. The field is read from the file specified in this attribute object. The user must ensure that the parameters for the attribute object do not conflict with those of the target object (e.g. frequency and near field distance should agree).

Factor (*factor*) [struct].

The field to be added will be multiplied by a complex factor before it is added.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Remarks

The command type may, together with the command type [Get Pattern](#), be used to add patterns from different sources or patterns of different ranges.

If the target range specifies field points outside the range defined in the attribute `Input Field`, those field values at such points will remain unchanged. This is utilised when patterns of different ranges are combined as illustrated by the following example.

PO calculation is accurate but time consuming for far-out side lobes of a reflector antenna. However, for large angles, a fast GTD determination of the side lobes will have sufficient accuracy. Two adjacent patterns are then calculated. A central pattern calculated by PO and a peripheral pattern determined by GTD. The final pattern is then constructed first by applying [Get Pattern](#) to e.g. the PO calculated pattern and next to add the GTD pattern by [Add Pattern](#). As the original patterns do not overlap and as values outside the ranges are zero, then the resulting added pattern will be the two original patterns pasted together.

SUBTRACT PATTERN (subtract_pattern)

Purpose

A command of the type *Subtract Pattern* retrieves and interpolates the radiated field within one *Field Storage* object (the attribute) to points specified in another *Field Storage* object (the target). The interpolated field is subtracted from the field already contained in the target object.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> subtract_pattern
(
    input_field           : ref(<n>),
    factor                : struct(db:<r>, deg:<r>)
)
where
<target> = name of an object of type Cylindrical Cut; Cylindrical Grid;
Planar Cut; Planar Grid; Spherical Cut; Spherical Grid; Surface Cut; Surface Grid; Tabulated uv-Points
<n>      = name of an object
<r>      = real number
```

Target

<target> [name of an object].

Reference to an object of one of the classes in *Field Storage*. This object defines the field points and the field components to be calculated by interpolation. The interpolated field will replace the field already contained in the object and the new field will be stored. The target *Field Storage* object must exist and it must have been referred to in an earlier *Get Pattern* or *Get Field* command such that it is not empty.

Attributes

Input Field (*input_field*) [name of an object].

Reference to another object of the class *Field Storage* from which the field shall be retrieved, interpolated and subtracted from the field already stored in the target object. The field is read from the file specified in this attribute object. The user must ensure that the parameters for the attribute object do not conflict with those of the target object (e.g. frequency and near field distance should agree).

Factor (*factor*) [struct].

The retrieved field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Remarks

Commands of the type *Subtract Pattern*, together with commands of the type *Get Pattern*, may be used to subtract patterns from different sources, see the remarks under *Get Pattern*.

If the target range specifies field points outside the range defined in the attribute Input Field, then the field values at such points will remain unchanged.

REPLACE PATTERN (replace_pattern)**Purpose**

A command of the type *Replace Pattern* retrieves and interpolates the radiated field within one *Field Storage* object (the attribute) to points specified in another *Field Storage* object (the target). The field is interpolated to the grid of the target object and replaces the field already contained there.

Links*Command Types***Syntax**

```
COMMAND OBJECT <target> replace_pattern
(
    input_field           : ref(<n>),
    factor                : struct(db:<r>, deg:<r>)
)
where
<target> = name of an object of type Cylindrical Cut; Cylindrical Grid;
Planar Cut; Planar Grid; Spherical Cut; Spherical Grid; Surface Cut; Surface Grid; Tabulated uv-Points
<n>      = name of an object
<r>      = real number
```

Target

<target> [name of an object].

Reference to an object of one of the classes in *Field Storage*. This object defines the field points and the field components to be calculated by interpolation. The interpolated field will replace the field already contained in the object and the new field will be stored. The target *Field Storage* object must exist and it must have been referred to in an earlier *Get Pattern* or *Get Field* command such that it is not empty.

Attributes

Input Field (*input_field*) [name of an object].

Reference to another object of the class *Field Storage* from which the field is retrieved and interpolated before it is replaced in the target object. The field is read from the file specified in this attribute object. The user must ensure that the parameters for the attribute object do not conflict with those of the target object (e.g. frequency and near field distance should agree).

Factor (*factor*) [struct].

The retrieved field will be multiplied by a complex factor.

Amplitude in dB (*db*) [real number], default: **0**.

Amplitude of the factor, in dB.

Phase in degrees (*deg*) [real number], default: **0**.

Phase of the factor, in degrees.

Remarks

A command of the type *Replace Pattern* may be used to replace a part of a pattern with another pattern.

If the target range specifies field points outside the range defined in the attribute Input Field, then the field values at such points will remain unchanged. This is utilised when patterns of different ranges are combined as illustrated by the following example.

PO calculation is accurate but time consuming for far-out side lobes of a reflector antenna. However, for large angles, a fast GTD determination of the side lobes will have sufficient accuracy. Two patterns are then calculated. A central pattern calculated by PO (because the GTD calculations here are too inaccurate) and a total pattern determined by GTD covering both the peripheral side lobes as well as the central pattern. The final pattern is then constructed by applying first *Get Pattern* to the GTD calculated pattern for the total range and next to replace the central region by the PO pattern using *Replace Pattern*. This will replace the field values of the central pattern by the PO field values while the peripheral pattern outside the inserted pattern is unchanged.

GET SWE (get_swe)

Purpose

A command of the type *Get SWE* activates the computation of the spherical wave coefficients for a field of an object of class *Source*. The expansion shall be specified in a *Spherical Wave Expansion (SWE)* object.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> get_swe
(
    source           : ref(<n>)
)
where
<target> = name of an object of type Spherical Wave Expansion (SWE)
<n>      = name of an object
```

Target

<target> [name of an object].

Reference to an object of the class *Spherical Wave Expansion (SWE)*. This object defines how the spherical wave coefficients shall be calculated as well as a possible file for storage of the coefficients. If an output file is specified and it already exists, it will be overwritten.

Attributes

Source (source) [name of an object], default: **blank**.

A reference to a *Source* radiating the field to be expanded in spherical modes.

Remarks

The field of the specified *Source* is calculated in the grid defined in the *Spherical Wave Expansion (SWE)* object. The spherical wave expansion is next carried out on basis of these field values and the coefficients are stored in the file defined in the *Spherical Wave Expansion (SWE)* object.

GET PLANE WAVE EXPANSION (get_plane_wave_expansion)

Purpose

A command of the type *Get Plane Wave Expansion* activates the computation of a *Plane Wave Expansion* of the field radiated by a *Source*.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> get_plane_wave_expansion
(
    source                  : sequence(ref(<n>), ...),
    auto_convergence        : <si>,
    convergence_on_scatterer : sequence(ref(<n>), ...),
    convergence_on_near_field_grid : sequence(ref(<n>), ...),
    field_accuracy          : <r>,
    max_bisections         : <i>,
    automatic_convergence_limit : <si>
)
where
<target> = name of an object of type Plane Wave Expansion
<i>      = integer
<n>      = name of an object
<r>      = real number
<si>     = item from a list of character strings
```

Target

<target> [name of an object].

Reference to an object of the class *Plane Wave Expansion*. This object defines how the plane wave coefficients shall be calculated as well as a possible file for storage of the coefficients. If an output file is specified and it already exists, it will be overwritten.

Attributes

Source (source) [sequence of names of other objects].

A sequence of one or more references to objects of class *Source* shall be specified.

Auto Convergence (auto_convergence) [item from a list of character strings], default: **off**.

This attribute can be used to switch off the automatic convergence calculations without changing the settings of the other attributes.

off

Convergence tests are not carried out. The computation uses the numbers of plane waves specified in the attribute `spectral_integration_grid` in the target object.

on

Convergence tests are carried out to determine the number of plane waves necessary to achieve convergence as specified below. The plane waves are subsequently calculated and stored in the target object. At least one scatterer or near-field grid must be specified when `auto_convergence` is 'on'. It is recommended to use the automatic convergence test. Otherwise, the user must do a manual convergence test to determine the numbers `n_theta` and `n_phi` in the attribute `spectral_integration_grid` in the target object.

Convergence On Scatterer (`convergence_on_scatterer`) [sequence of names of other objects].

A sequence of one or more references to objects of class `Scatterer` defining the scatterers on which the plane-wave expansion in the target object must converge.

Convergence On Near Field Grid (`convergence_on_near_field_grid`) [sequence of names of other objects].

A sequence of one or more references to objects of class `Field Storage` defining near-field points at which the calculation of the field from the target object must converge. The objects must refer to near fields since the field from a plane-wave expansion is only valid in the near field.

Field Accuracy (`field_accuracy`) [real number], default: **-80**.

The spectral integration has converged when a small change in `n_theta` and `n_phi` will cause a pattern change less than the specified Field Accuracy. The pattern change is calculated as the complex field difference between two successive pattern calculations based on two sets of `n_theta` and `n_phi`. The Field Accuracy shall be given in dB relative to the maximum power level in the pattern. See also the Remarks below.

Max Bisections (`max_bisections`) [integer], default: **5**.

The attribute Max Bisections controls how precise `n_theta` and `n_phi` are determined. The value must be a positive integer giving the maximum number of bisections in the integration grid evaluation. If zero, no bisections are performed. See the Remarks below for details.

Automatic Convergence Limit (`automatic_convergence_limit`) [item from a list of character strings], default: **on**.

This attribute is used to stop the optimization if the Field Accuracy limit cannot be obtained.

on

The maximum values of *n_theta* and *n_phi* are given by simple rules. See the Remarks below.

off

No maximum values for *n_theta* and *n_phi*.

Remarks

The integration grid of the plane-wave expansion is given as *n_theta* by *n_phi* far-field points as defined in the *Plane Wave Expansion* specified as the target. The values of these parameters must be chosen with care because too small values will give incorrect results and too large values may give unacceptable long time of computation.

Optimum values of the parameters *n_theta* and *n_phi* in the *Plane Wave Expansion* target are automatically determined when Auto Convergence is specified to 'on'. In this case, the corresponding values specified in the *Plane Wave Expansion* target are not used.

When initially specifying Auto Convergence to 'off' the user may control the integration accuracy by manually setting the values of *n_theta* and *n_phi* of the target *Plane Wave Expansion* object, but since the values depend on both the source and the output points it is difficult to give general guidelines. It is thus recommended to use the Auto Convergence facility. If GRASP program is run many times and if the necessary values of *n_theta* and *n_phi* are known from previous runs some computation time can be saved by specifying Auto Convergence 'off' and insert the known values in the *Plane Wave Expansion* target object.

GET COUPLING (get_coupling)

Purpose

A command of the type *Get Coupling* activates the computation of the coupling quotients between a given set of transmitter *Sources* and the receiver *Sources* specified in the referenced *Coupling System* object.

For backward compatibility the command type may also be applied for objects of the obsolete class *Coupling (Obsolete)*. The following mentioned references to objects of class *Coupling System* may thus also apply to objects of class *Coupling (Obsolete)*.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> get_coupling
(
    source           : sequence(ref(<n>), ...)
)
where
<target> = name of an object of type Coupling (Obsolete); Coupling System
<n>      = name of an object
```

Target

<target> [name of an object].

Reference to an object of the class *Coupling System*. This object defines how the coupling quotients shall be calculated as well as a possible file for storage of the quotients. If an output file is specified and it already exists, it will be overwritten.

Attributes

Source (*source*) [sequence of names of other objects], default: **blank**.

References to objects of the class *Source*. It is the field of the sources (transmitters) defined in these object which is coupled to the field from the receiver sources specified in the *Coupling System* object.

Remarks

More than one source (transmitter) may be specified in the attribute *source*. The sum of the coupling quotients due to all the transmitter sources will be superimposed in the object according to the rule defined in the specified object of class *Coupling System*.

EXPORT TO STEP FILE (export_step)

Purpose

A command of the type *Export to STEP File* exports the geometry of a scatterer to a CAD file. The following types of Scatterers can be exported: *Reflector*, *Plate*, *Struts*, *Box*, *Tabulated Mesh* and *Scatterer Cluster*. The produced CAD file will be in the STEP format and the recommended extension is *.stp* (or *.step*). The surface of the exported scatterer will have zero thickness.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> export_step
(
    output_file_name          : <f>,
    tolerance                 : <rl>,
    group_bodies               : <si>,
    coor_sys                  : ref(<n>)
)
where
<target> = name of an object of type Reflector; Rectangular Plate;
Parallelogram; Triangular Plate; Circular Plate; Polygonal Struts; Circular Struts; Box; Tabulated Mesh; Scatterer Cluster
<n>      = name of an object
<rl>     = real number with unit of length
<f>       = file name
<si>     = item from a list of character strings
```

Target

<target> [name of an object].

Reference to an object of one of the *Scatterer* classes: *Reflector*, *Plate*, *Struts*, *Box*, *Tabulated Mesh* or *Scatterer Cluster* (the scatterers referenced in the *Scatterer Cluster* shall also be restricted to these classes). The referenced object defines the geometry to be exported.

Attributes

Output File Name (*output_file_name*) [file name].

The name of the file on which the geometry is written in STEP format. The recommended file extension is *.stp* (or *.step*).

Tolerance (*tolerance*) [real number with unit of length].

The value of tolerance expresses the tolerance of the representation in the CAD file, i.e. the maximum allowed deviation between the

GRASP scatterer definition and the representation. The tolerance will be written to the CAD file. The tolerance is an upper bound on the error. The actual error might be smaller, and in some cases several orders of magnitudes smaller. For reflectors, the time to create the CAD file and the size of the file will increase when tolerance is decreased.

Group Bodies (*group_bodies*) [item from a list of character strings], default: **off**.

When the scatterer object contains more than one separate body (this may be the case for *Struts* and for *Scatterer Cluster*), these bodies may be grouped or not in the CAD file.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The surface of the scatterer will be expressed in the referenced *Coordinate Systems* when written to the CAD file.

Remarks

The Export to the CAD file works for all reflector *Surfaces* and nearly all reflector *Rims*. However, reflectors with *Tabulated Rims*, and reflectors with a *Triangular Rim* and a circular hole, cannot be exported. The reflector will be represented as a single surface shell in the CAD file. The surface shell will consist of a number of connected sub-surfaces. The thickness of the reflector is always zero.

The output CAD file is written with the length unit 'mm'.

EXPORT TO IGES FILE (export_iges)

Purpose

A command of the type *Export to IGES File* exports the geometry of a scatterer to a CAD file. The following types of *Scatterers* can be exported: *Reflector*, *Plate*, *Struts*, *Box*, *Tabulated Mesh* and *Scatterer Cluster*. The produced CAD file will be in the IGES format and the recommended extension is *.igs* (or *.iges*). The surface of the exported scatterer will have zero thickness.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> export_iges
(
    output_file_name          : <f>,
    tolerance                 : <rl>,
    group_bodies               : <si>,
    coor_sys                  : ref(<n>)
)
where
<target> = name of an object of type Reflector; Rectangular Plate;
Parallelogram; Triangular Plate; Circular Plate; Polygonal Struts; Circular Struts; Box; Tabulated Mesh; Scatterer Cluster
<n>      = name of an object
<rl>     = real number with unit of length
<f>       = file name
<si>     = item from a list of character strings
```

Target

<target> [name of an object].

Reference to an object of one of the *Scatterer* classes: *Reflector*, *Plate*, *Struts*, *Box*, *Tabulated Mesh* or *Scatterer Cluster* (the scatterers referenced in the *Scatterer Cluster* shall also be restricted to these classes). The referenced object defines the geometry to be exported.

Attributes

Output File Name (*output_file_name*) [file name].

The name of the file on which the geometry is written in IGES format. The recommended file extension is *.igs* (or *.iges*).

Tolerance (*tolerance*) [real number with unit of length].

The value of tolerance expresses the tolerance of the representation in the CAD file, i.e. the maximum allowed deviation between the GRASP scatterer definition and the representation.

Group Bodies (*group_bodies*) [item from a list of character strings], default:
off.

When the scatterer object contains more than one separate body (this may be the case for *Struts* and for *Scatterer Cluster*), these bodies may be grouped or not in the CAD file.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes. The surface of the scatterer will be expressed in the referenced *Coordinate Systems* when written to the CAD file.

Remarks

The Export to the CAD file works for all reflector *Surfaces* and nearly all reflector *Rims*. However, reflectors with *Tabulated Rims*, and reflectors with a *Triangular Rim* and a circular hole, cannot be exported. The reflector will be represented as a single surface shell in the CAD file. The surface shell will consist of a number of connected sub-surfaces. The thickness of the reflector is always zero.

The output CAD file is written with the length unit 'mm'.

GET REFLECTOR DATA (get_reflector_data)

Purpose

A command of the type *Get Reflector Data* activates the computation of the reflector data for a specified *Reflector* object and stores the results on one or more files as specified in the referred object of the *Derived Geometry Data* class.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> get_reflector_data
(
    object : ref(<n>)
)
where
<target> = name of an object of type Electrical Properties Data Output;
Reflector Data Output; Rim Data Output; Surface Data Output
<n>      = name of an object
```

Target

<target> [name of an object].

Reference to an object of one of the classes in *Derived Geometry Data*. This object defines the reflector parameters to be calculated and the files for storage of the parameters.

When the reference is to an object of class *Reflector Data Output* then reflector data are calculated and stored as specified in this object.

When the reference is to an object of class *Surface Data Output* (either directly or through an object of class *Reflector Data Output*) then surface geometrical data are calculated at the points specified in the object of class *Surface Data Output*.

When the reference is to an object of class *Rim Data Output* (either directly or through an object of class *Reflector Data Output*) then rim data are calculated as specified in the object of class *Rim Data Output*.

When the reference is to an object of class *Electrical Properties Data Output* (either directly or through an object of class *Reflector Data Output*) then the reflection and transmission coefficients for the reflector are calculated as specified in the object of class *Electrical Properties Data Output*.

Attributes

Object (*object*) [name of an object].

Reference to an object of the class *Reflector* for which data are determined and stored as specified in the *Derived Geometry Data* class.

GET COORDINATE SYSTEM (get_coor_sys)

Purpose

A command of the type *Get Coordinate System* informs on the position and orientation of a coordinate system relative to the global coordinate system or relative to the coordinate system given as attribute.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> get_coor_sys  
(  
    base : ref(<n>)  
)
```

where

<target> = name of an object of type *Coordinate System*; *Coordinate System*, *Euler Angles*; *Coordinate System*, *GRASP Angles*; *Tabulated Coordinate System*

<n> = name of an object

Target

<target> [name of an object].

Reference to a *Coordinate Systems* object. The origin and the axes defined in the target-object will be given relative to the specified Base. The *Coordinate Systems* object is not changed.

Attributes

Base (base) [name of an object], default: **blank**.

Reference to another object of *Coordinate Systems* class, specifying the coordinate system in which the <target> shall be expressed. If *base* and its reference are omitted the <target> will be given relative to the global coordinate system.

Remarks

The data of the coordinate system are written in the standard output file and in the log file. The position of the origin is given in metres.

CHANGE COORDINATE SYSTEM BASE (coor_sys_change_base)

Purpose

A command of the type *Change Coordinate System Base* transforms a coordinate system given in one base into another coordinate system, which is then used as base.

The position and orientation of the coordinate system are unchanged but the definition of its origin and axes becomes different because the definition is given relative to a new base.

In the GUI the command may be invoked when a *Coordinate Systems* object is opened in the Object Editor. The base of the coordinate system is then changed directly, while issuing the command from the Command Tab will not change the base until a job is submitted.

Links

[Command Types](#)

Syntax

COMMAND OBJECT <target> *coor_sys_change_base*

```
(  
    base : ref(<n>)  
)
```

where

<target> = name of an object of type *Coordinate System*; *Coordinate System*, *Euler Angles*; *Coordinate System*, *GRASP Angles*; *Tabulated Coordinate System*

<n> = name of an object

Target

<target> [name of an object].

Reference to a *Coordinate Systems* object. The origin and the axes defined in the target-object will be changed and given relative to the specified Base.

Attributes

Base (*base*) [name of an object], default: **blank**.

Reference to another object of a *Coordinate Systems* class, specifying the coordinate system in which the <target> shall be expressed. If Base and its reference are omitted the <target> will be given relative to the global coordinate system.

When the command is invoked from the Object Editor in the GUI this attribute is specified as 'New Base'.

Remarks

Note that the coordinate system is changed for the subsequent commands only.

This command is very useful if a large structure consisting of several objects shall be moved. A simple example is given.

Consider a single reflector antenna system as shown in the figure below. It consists of:

- a feed horn defined in the feed coordinate system, CSfeed,
- a reflector defined in the reflector coordinate system, CSrefl, and finally,
- the output (pattern cuts) is defined in an output coordinate system, CSout. This system has origin at the centre of the reflector.

The coordinate systems CSfeed and CSout are defined in CSrefl.

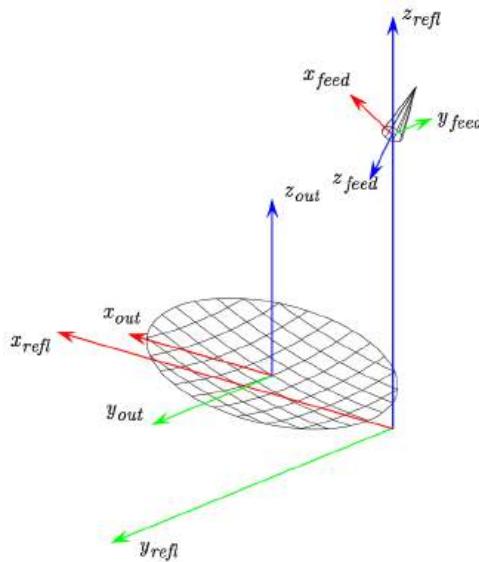


Figure 1 Single reflector antenna consisting of feed, reflector and reference frame for output.

When the first field calculations have been carried out, it is desired next to determine the pattern when the feed horn is scanned. The scanning shall be carried out by rotating the complete feed assembly 10° around the y -axis of CSout (y_{out} in the figure).

We therefore define a coordinate system to which all the objects to be rotated refer. We denote this coordinate system CSrot, and initially it is identical to CSout.

The objects to be rotated are the feed and its coordinate system CSfeed. CSfeed is defined (has base) in the coordinate system CSrefl and we change this base to CSrot by the command

```
COMMAND OBJECT CSfeed coor_sys_change_base( base: ref(CSrot) )
```

The rotation is then carried out by redefining CSrot with respect to CSout. This will rotate CSfeed and the feed, and we get the required configuration as shown in the next figure.

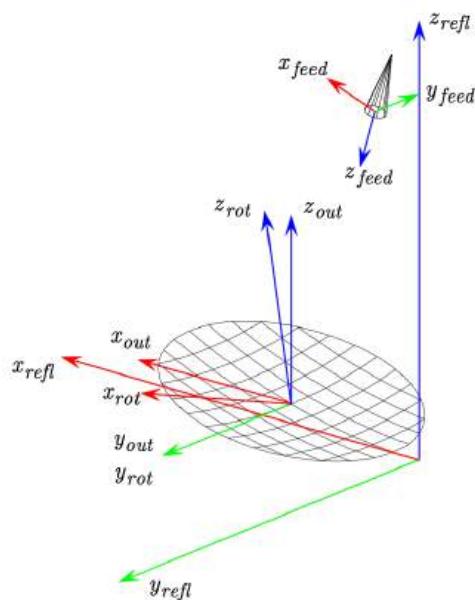


Figure 2

The same antenna as in the previous figure. Now the feed and its coordinate system has base in CS_{rot} which is rotated 10° with respect to CS_{out} around y_{out} .

WRITE MOM MESH (write_mom_mesh)

Purpose

A command of the type *Write MoM Mesh* is used to export the MoM mesh generated by the *MoM* class to a file. The format of the file is described in *Meshed Geometries*. GRASP features third-order meshing in terms of 16-node quadrilaterals and 4-node wire segments and can be used as a high-accuracy meshing tool for other numerical codes.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> write_mom_mesh
(
    output_file_name          : <f>,
    frequency_index           : <i>,
    body_index                : <i>,
    max_order                 : <i>,
    coor_sys                  : ref(<n>),
    length_unit               : <si>
)
where
<target> = name of an object of type MoM
<i>      = integer
<n>      = name of an object
<f>      = file name
<si>     = item from a list of character strings
```

Target

<target> [name of an object].

Reference to an object of the class *MoM* which generates the MoM mesh to be written.

Attributes

Output File Name (*output_file_name*) [file name].

The name of the file where the mesh is written. The format of the file is described in Section *Meshed Geometries*. The recommended file extension is *.msh*.

Frequency Index (*frequency_index*) [integer], default: **1**.

If the target *MoM* object contains more than one frequency this attribute selects the frequency by its number in the list of specified frequencies. The generated mesh will contain edges that are no longer than specified by the *max_mesh_length* attribute (in wavelengths) in the *MoM* object.

Body Index (*body_index*) [integer], default: **0**.

This attribute selects the desired body if the scatterer object specified by the *MoM* object contains more than one body. As an example, a *Polygonal Stiffeners* object may define several stiffeners that can be selected independently by specifying a nonzero value of Body Index. See the Remarks below for details. Specifying the Body Index to zero implies that the mesh file contains all bodies of the scatterer.

Max Order (*max_order*) [integer], default: **4**.

The maximum order of the patches in the mesh file. The internal mesher of GRASP uses 4-node patches (order 1), 9-node patches (order 2), or 16-node patches (order 3), as illustrated in Section *Mesched Geometries*. In addition, a MoM mesh may also include 25-node patches (order 4) if a *Tabulated Mesh* is used. However, most numerical codes are not able to use the high-order patches employed by GRASP and the mesh output can therefore be limited to a specified order. The resulting mesh may still contain patches of orders lower than specified if the scatterer object contains e.g. plane faces.

Coordinate System (*coor_sys*) [name of an object], default: **blank**.

Reference to an object of one of the *Coordinate Systems* classes, defining the coordinate system in which the MoM mesh will be computed.

Length Unit (*length_unit*) [item from a list of character strings], default: **m**.

Defines the length unit in which the (x,y,z) -coordinates in the mesh file are given (mm, cm, m, km, in, ft).

Remarks

A MoM mesh is calculated and exported to a file when a *Write MoM Mesh* command is executed. The mesh has been specified in the *MoM* object which is the target of this command.

Numbering of bodies

The scattering bodies are numbered in the order in which they are specified. For example, if the scatterer specified in the *MoM* object is a *Scatterer Cluster* consisting of a *Polygonal Stiffeners* scatterer, which specifies 15 stiffeners, and a second *Polygonal Stiffeners* scatterer, which specifies 40 stiffeners, then the *body_index* may take values from 1 through 15 for reference to the stiffeners specified in the first *Polygonal Stiffeners* object and the values 16 through 55 for the stiffeners specified in the second *Polygonal Stiffeners* object.

ACTIVATE MOVEMENT (activate_movement)

Purpose

A command of the type *Activate Movement* activates and moves the coordinate systems and the herein defined structures to one of the positions specified in a *Movement Definition* object.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> activate_movement
(
    position           : <i>
)
where
<target> = name of an object of type Movement Definition
<i>      = integer
```

Target

<target> [name of an object].

Reference to an object of the class *Movement Definition*. This object defines how a coordinate system and all objects defined herein are rotated or translated.

Attributes

Position (*position*) [integer].

Index of a position specified as in the *Movement Definition* object.

Remarks

In the object *Movement Definition* one or more movements are defined. During these movements the specified coordinate systems and the herein defined objects will be moved through of set of positions. These positions are enumerated with an index and it is the index of the requested position which shall be specified in the argument *position*. The indexing system is detailed under Remarks in class *Movement Definition*.

CHECK BLOCKAGE (check_blockage)

Purpose

A command of the type *Check Blockage* initiates a forward ray tracing to identify blockage in a system of scatterers (typically a reflector antenna system).

Links

Command Types

Syntax

```
COMMAND OBJECT <target> check_blockage
(
    source : sequence(ref(<n>), ...)
)
where
<target> = name of an object of type Blockage Check
<n>      = name of an object
```

Target

<target> [name of an object].

Reference to an object of the class *Blockage Check*. This object defines the sequence in which the scatterers are intended to be hit by the rays. Also the range of the emitted rays are defined herein.

Attributes

Source (source) [sequence of names of other objects].

References to objects of the class *Source*. Rays are emitted from each of these sources and checked for blockage.

Remarks

When blockage occurs it is written in the standard output file of GRASP.

EXECUTE EXTERNAL COMMAND (execute_external_command)

Purpose

A command of the type *Execute External Command* is used to request execution of user-defined external command. The command name and command line arguments are specified in an object of class *External Command*, please consult the description of the *External Command* class for details.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> execute_external_command  
(  
)
```

where

<target> = name of an object of type *External Command*

Target

<target> [name of an object].

The external command specified by the target object will be executed.

GET XYZ PLOT (OBSOLETE) (get_xyz_plot)

Purpose

A command of the type *Get XYZ Plot (Obsolete)* activates the computation of a 3D-geometry and stores the data on an XYZ-data file. The command type is obsolete and maintained for compatibility with older GRASP-versions.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> get_xyz_plot
(
    plot_objects           : sequence(ref(<n>), ...)
)
where
<target> = name of an object of type Plot Settings (Obsolete)
<n>      = name of an object
```

Target

<target> [name of an object].

Reference to an object of the *Plot Settings (Obsolete)* class. This object holds the name of the file to which the XYZ-data are stored. Any plot data already contained in the object will be overwritten, and if the plot data file already exists it will be overwritten as well.

Attributes

Plot Objects (*plot_objects*) [sequence of names of other objects], default: **all**.

Sequence of references to *Plot Objects*. The name of the reference may be all which comprises references to all the *Plot Objects* of the project. The XYZ-data of the *Plot Objects* are calculated and stored in the data file specified in the target object.

ADD XYZ PLOT (OBSOLETE) (add_xyz_plot)

Purpose

A command of the type *Add XYZ Plot (Obsolete)* activates the computation of a 3D-geometry and adds the data to an existing XYZ-data file. The command type is obsolete and maintained for compatibility with older GRASP-versions.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> add_xyz_plot
(
    plot_objects           : sequence(ref(<n>), ...)
)
where
<target> = name of an object of type Plot Settings (Obsolete)
<n>      = name of an object
```

Target

<target> [name of an object].

Reference to an object of the *Plot Settings (Obsolete)* class. This object holds the name of the file to which the XYZ-data are added. The file must exist, otherwise *Get XYZ Plot (Obsolete)* should be applied.

Attributes

Plot Objects (*plot_objects*) [sequence of names of other objects], default: **all**.

Sequence of references to *Plot Objects*. The name of the reference may be 'all' which comprises references to all the *Plot Objects* of the project. The XYZ-data of the *Plot Objects* are calculated and added to the XYZ-data already stored in the data file specified in the target object.

GET ALL XYZ PLOT (OBSOLETE) (get_all_xyz_plot)

Purpose

A command of the type *Get All XYZ Plot (Obsolete)* activates the computation of a 3D-geometry and stores the data on an XYZ-data file. The command type is obsolete and maintained for compatibility with older GRASP-versions.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> get_all_xyz_plot  
(  
)
```

where

<target> = name of an object of type *Plot Settings (Obsolete)*

Target

<target> [name of an object].

Reference to an object of the *Plot Settings (Obsolete)* class. This object holds the name of the file to which the XYZ-data are stored. Any plot data already contained in the object will be overwritten, and if the plot data file already exists it will be overwritten as well.

GET OPENGL PLOT (OBSOLETE) (get_ogl_plot)

Purpose

A command of the type *Get OpenGL Plot (Obsolete)* activates the computation of a 3D-geometry and stores the data on an XYZ-data file. The command type is obsolete and maintained for compatibility with older GRASP-versions.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> get_ogl_plot
(
    plot_objects           : sequence(ref(<n>), ...)
)
where
<target> = name of an object of type Plot Settings (Obsolete)
<n>      = name of an object
```

Target

<target> [name of an object].

Reference to an object of the *Plot Settings (Obsolete)* class. This object holds the name of the file to which the XYZ-data are stored. Any plot data already contained in the object will be overwritten, and if the plot data file already exists it will be overwritten as well.

Attributes

Plot Objects (*plot_objects*) [sequence of names of other objects], default: **all**.

Sequence of references to *Plot Objects*.

The name of the reference may be 'all' which comprises references to all the *Plot Objects* of the project.

The XYZ-values of the *Plot Objects* are calculated for the geometries specified by this attribute.

ADD OPENGL PLOT (OBSOLETE) (add_ogl_plot)

Purpose

A command of the type *Add OpenGL Plot (Obsolete)* activates the computation of a 3D geometry and adds the data to an existing XYZ-data file. The command type is obsolete and maintained for compatibility with older GRASP-versions.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> add_ogl_plot
(
    plot_objects : sequence(ref(<n>), ...)
)
where
<target> = name of an object of type Plot Settings (Obsolete)
<n>      = name of an object
```

Target

<target> [name of an object].

Reference to an object of the *Plot Settings (Obsolete)* class. This object holds the name of the file to which the XYZ-data are added. The file must exist, otherwise *Get OpenGL Plot (Obsolete)* should be applied.

Attributes

Plot Objects (*plot_objects*) [sequence of names of other objects], default: **all**.

Sequence of references to *Plot Objects*.

The name of the reference may be 'all' which comprises references to all the *Plot Objects* of the project.

The XYZ-values of the *Plot Objects* are calculated for the geometries specified by this attribute and added to the XYZ-data already stored in the data file of the target.

GET ALL OPENGL PLOT (OBSOLETE) (get_all_ogl_plot)

Purpose

A command of the type *Get All OpenGL Plot (Obsolete)* activates the computation of a 3D-geometry and stores the data on an XYZ-data file. The command type is obsolete and maintained for compatibility with older GRASP-versions.

Links

Command Types

Syntax

```
COMMAND OBJECT <target> get_all_ogl_plot  
(  
)
```

where

<target> = name of an object of type *Plot Settings (Obsolete)*

Target

<target> [name of an object].

Reference to an object of the *Plot Settings (Obsolete)* class. This object holds the name of the file to which the XYZ-data are stored. Any plot data already contained in the object will be overwritten, and if the plot data file already exists it will be overwritten as well.

COMPUTE CHAIN (compute_chain)

Purpose

A command of the type **Compute Chain** activates field computation through a chain of components. It is mainly intended to be used from the Frame Design Tool where this command is generated automatically by the Command Wizard. This is described in details in the 'QUAST add-on' manual.

Components can be feeds, reflectors, plane mirrors, beam splitters, loads and output field cuts and grids. The computations starts with an initial component (typically a feed) and hereafter the field through a sequence of specified components is computed.

Links

[Command Types](#)

Syntax

```
COMMAND OBJECT <target> compute_chain
(
    start_component : struct(frame:ref(<n>),
                               component:ref(<n>),
                               output_field_method:<si>),
    list_of_components : sequence(
        struct(
            component:ref(<n>),
            output_field_method:<si>),
        ...),
    plane_wave_expansion : sequence(
        struct(
            component:ref(<n>),
            beam_cone_angle:<r>),
        ...),
    gaussian_beam_mode_definition : sequence(
        struct(
            component:ref(<n>),
            mmax:<i>,
            nmax:<i>),
        ...))
)
```

where

<target> = name of an object of type **Frame**
 <i> = integer
 <n> = name of an object
 <r> = real number
 <si> = item from a list of character strings

Target

<target> [name of an object].

Reference to an object of the class **Frame**. This object is usually created by the Frame Design Tool by which a system of connected components can be defined.

Attributes

Start Component (*start_component*) [struct].

Defines the initial component in the chain of components to be analysed. The initial component is usually a feed, but it can also be a scattering component (reflector, plane mirror or beam splitter) if the currents on such a component have already been calculated.

Frame (*frame*) [name of an object].

Reference to the Frame in which the initial component is located.

Component (*component*) [name of an object].

Name of the initial component.

Output Field Method (*output_field_method*) [item from a list of character strings], default: **NA**.

Analysis method to compute the field from the initial component.

NA

Not Applicable. This value must be supplied for feeds.

PO

Physical Optics. Can be specified for reflectors, plane mirrors and beam splitters, where the PO currents have already been calculated.

List of Components (*list_of_components*) [sequence of structs].

Defines the next components in the chain of components to be analysed. Each struct in the sequence has the same members as the struct above for the Start Component. The member Output Field Method may, however, have other values

Component (*component*) [name of an object].

Name of the component.

Output Field Method (*output_field_method*) [item from a list of character strings], default: **NA**.

Analysis method to compute the field from this component.

NA

Not Applicable. This value must be supplied for output field cuts and grids, and for loads

PO

Physical Optics. Allowed for reflectors, plane mirrors, beam splitters, lenses, apertures and interferometers. The density of the PO integration grid is determined automatically.

Gauss_Laguerre

Gauss-Laguerre beam mode expansion. Can be specified for reflectors, plane mirrors, beam splitters, lenses, apertures and interferometers. This activates the Gauss-laguerre analysis method which is very fast, but less accurate than PO.

Plane Wave Expansion (*plane_wave_expansion*) [sequence of structs].

The incident field on the specified component is expanded in a spectrum of plane waves. This is recommended for beam splitters and interferometers because the special surface materials on such components will be treated more accurately than by standard PO. It has no effect on reflectors, plane mirrors, lenses and apertures.

Component (*component*) [name of an object].

Component for which the incident field shall be expanded in a plane-wave spectrum.

Beam Cone Angle (*beam_cone_angle*) [real number], default: **90**.

The plane-wave expansion is truncated such that only plane waves with direction of propagation inside the cone defined by Beam Cone Angle are retained. The Beam Cone Angle should be set to a value such that the major part of the power (e.g. 95%) is contained in this cone. See the description of class **Plane Wave Expansion**. A value larger than necessary will increase the computation time.

Gaussian Beam Mode Definition (*gaussian_beam_mode_definition*) [sequence of structs].

Modification of the default truncation of the Gauss-Laguerre series.

Component (*component*) [name of an object].

Component for which the default Gauss Laguerre mode truncation shall be changed. This component shall also appear in the List of Components attribute with Output Field Method set to Gauss_Laguerre.

Mmax (*mmax*) [integer], default: **4**.

Maximum value of the m-index.

Nmax (*nmax*) [integer], default: **4**.

Maximum value of the n-index.

Remarks

See the 'QUAST add-on' manual for further details.

SET ATTRIBUTE (set)**Purpose**

By a command of the type *Set Attribute* the values for one or more attributes of a specified object may be defined or modified.

Links*Command Types***Syntax**

COMMAND OBJECT <object_name> set (<arg>)

where:

<object_name> = name of an object

<arg> = an argument defining one or more attributes of the object

Attributes

<object_name> [name of an object].

The user-specified name of the object to be created.

<arg> [argument].

One or more arguments, separated by comma, defining the value of a number of attributes of the specified object. The syntax of the arguments shall follow the syntax described for the attributes of the class to which the object belongs. See the remarks below.

Remarks

One or more attributes to a specified object may be defined or modified. The values of the attributes shall be specified in the same way as they are written in the *tor*-file. The syntax depends on the attribute in question and is found in the description of the class to which the object belongs.

Consider as example the generation of an object defining a coordinate system. The name of the object shall be *New_Coor_Sys* and the coordinate system shall have axes parallel to those of the global coordinate system but the origin shall be displaced to (0.1 m, 0.2 m, 0.3 m) in the global coordinate system.

The object is generated by a command of the type *Create Object*:

COMMAND OBJECT New_Coor_Sys create coor_sys

which creates a new object with name *New_Coor_Sys* of class *Coordinate System*. By default this coordinate system is equivalent to the global coordinate system. We will then change the position of the origin to (0.1 m, 0.2 m, 0.3 m) which is done by

COMMAND OBJECT New_Coor_Sys set (origin : struct(x: 0.1 m, y: 0.2 m, z: 0.3 m))

according to the syntax for the attribute *origin*, see section Syntax in class *Coordinate System*.

CREATE OBJECT (create)

Purpose

A command of the type *Create Object* creates an object with a name as specified. Attributes not having a default value are left undefined (the object will be registered as incomplete).

Links

Command Types

Syntax

COMMAND OBJECT <object_name> *create* <class_name>

where:

<object_name> = name of an object

<class_name> = name of a class

Attributes

<object_name> [name of an object].

The user-specified name of the object to be created.

<class_name> [name of a class].

The name of the class to which the object shall belong.

Remarks

An object with the specified name will be created. Values to the attributes of the object may be given by a command of the type *Set Attribute*.

DELETE OBJECT (delete)**Purpose**

A command of the type *Delete Object* deletes a specified object.

Links

Command Types

Syntax

COMMAND OBJECT <name> *delete*

where:

<name> = name of an object

Attributes

<name> [name of an object].

The name of the object to be deleted.

Remarks

The object with the specified name will be deleted.

RENAME (rename)

Purpose

A command of the type *Rename* renames an object.

The command shall only be used when GRASP is operated in batch mode, see Chapter 7.

Links

Command Types

Syntax

COMMAND OBJECT <name_1> *rename* <name_2>

where:

<name> = name of an object

Attributes

<name_1> [name of an object].

The name of the object to be renamed.

<name_2> [name of an object].

The new name of the object.

If an object with this name already exists an error is issued and the object will not be renamed.

Remarks

The command is used to rename an object without changing its content.

COPY (copy)

Purpose

A command of the type *Copy* generates a copy of an object.

The command shall only be used when GRASP is operated in batch mode, see Chapter 7.

Links

Command Types

Syntax

COPY <name_1> <name_2> /Y

where:

<name> = name of an object

/Y = command control parameter

Attributes

<name_1> [name of an object].

The name of the object to be copied.

<name_2> [name of an object], default: **Copy_of_<name_1>**.

The name of the copied object. If <name_2> is not given the copied object will be named 'Copy_of_' concatenated with the name of the original object.

If an object with this name already exists an error is issued and the copying will not be carried out.

/Y [command control].

When /Y is added to the end of the command the copied object will be named as specified regardless to the existence of an object already having this name. That object will be deleted.

Remarks

This command is used to copy the contents of an already existing object into a new object.

FILES READ ALL (files read all)

Purpose

A command of the type *Files Read All* reads all objects given in a *tor*-file into the active project.

The command shall only be used when GRASP is operated in batch mode, see Chapter 7.

Links

Command Types

Syntax

FILES READ ALL <name>

where:

<name> = file name

Attributes

<name> [file name].

The name of the *tor*-file to be read. The file name shall include the relative path to the folder with the file.

Remarks

All objects of the specified *tor*-file will be read into the active GRASP project.

FILES WRITE ALL (files write all)

Purpose

A command of the type *Files Write All* writes all objects of the actual project to a *tor*-file.

The command shall only be used when GRASP is operated in batch mode, see Chapter 7.

Links

Command Types

Syntax

FILES WRITE ALL <name>

where:

<name> = file name

Attributes

<name> [file name].

The name of the *tor*-file to which the objects are written. The file name shall include the relative path to the folder with the file.

Remarks

All objects of the current GRASP project will be written on the specified file. The extension of the file name should be *.tor* for compatibility reasons.

Warning: If a file with the specified name exists it will be overwritten without further warning.

BROWSE OBJECTS (browse objects)

Purpose

A command of the type *Browse Objects* will list all objects in the project.

The command shall only be used when GRASP is operated in batch mode, see Chapter 7.

Links

Command Types

Syntax

BROWSE OBJECTS

Attributes

The command has no attributes

Remarks

The *Browse Objects* command will list all objects that exists in the project. The list is given in the command window.

BROWSE GHOSTS (browse ghosts)

Purpose

A command of the type *Browse Ghosts* will list the ghost objects in the project.

The command shall only be used when GRASP is operated in batch mode, see Chapter 7.

Links

[Command Types](#)

Syntax

BROWSE OBJECTS

Attributes

The command has no attributes

Remarks

The command lists the ghost objects - if any - in the project. A ghost object is an object, which has not been defined but is referenced by another object. A project cannot be executed when ghost objects occur. Ghost objects may be generated by typing errors or by erroneous commands. Inclusion of the command *Browse Ghosts* is thus a useful check before time consuming commands are started in a batch run.

The list of the ghost objects is given in the command window.

BROWSE INCOMPLETE (browse incomplete)

Purpose

A command of the type *Browse Incomplete* will list incomplete objects in the project.

The command shall only be used when GRASP is operated in batch mode, see Chapter 7.

Links

Command Types

Syntax

BROWSE OBJECTS

Attributes

The command has no attributes

Remarks

The command lists the incomplete objects - if any - in the project. An incomplete object is an object, for which one or more attributes are undefined (nor have been given a default value).

The list of the incomplete objects is given in the command window.

QUIT (quit)

Purpose

A command of the type *Quit* will stop the execution of the program.

The command shall only be used when GRASP is operated in batch mode, see Chapter 7.

Links

Command Types

Syntax

QUIT

Attributes

The command has no attributes

Remarks

This command shall be the last in a command sequence. If the command is reached in the middle of a command sequence the execution of the program is stopped. This may be used to execute only a part of a command sequence. Thus, in a complex project with many separate calculations it may - during the testing period but also later - be useful to place at the top of the *tci*-file a copy of the commands for a single set of calculation followed by the QUIT command.

9.5 Applicable Units and the dB-Scale

Units

GRASP offers the following units for the attributes, where appropriate:

Valid units of length are

mm	for millimetre(s)
cm	for centimetre(s)
m	for metre(s)
km	for kilometre(s)
in	for inch(es)
ft	for foot (feet)

Valid units of frequency are

Hz	for hertz
kHz	for kilohertz
MHz	for megahertz
GHz	for gigahertz
THz	for terahertz

The unit for conductivity is

S/m for Siemens per metre

dB-scale and field units

Quantities which are expressed in power may also be expressed logarithmic in units of decibel (dB). The conversion is defined by

$$P^{dB} = 10 \log_{10}(P^{watt}) \quad (1)$$

where P^{watt} is the quantity in linear scale which shall be converted, and P^{dB} is the same quantity expressed in dB.

Conventionally, the conversions in electromagnetics are on linear power values (having the dimension of watt). Therefore the quantity to be converted to dB is denoted P^{watt} in the equation above.

In GRASP the fields (\vec{E} and \vec{H}) and the currents (\vec{J}) are determined in linear scale having the unit $watt^{1/2}$, see GRASP Technical Description Section 4.1 . The expression for e.g. the electromagnetic field in dB, E^{dB} , is thus based on the power of the field which is given by the amplitude squared, $|\vec{E}|^2$, thus

$$E^{dB} = 10 \log_{10}(|\vec{E}|^2) \quad (2)$$

This expression is often written as

$$E^{dB} = 20 \log_{10}(|\vec{E}|) \quad (3)$$

Note that if the fields, \vec{E} and \vec{H} , are expressed in the international standard units, the SI units, then the field dimensions are volt/metre (V/m) and ampere/metre (A/m), respectively. This occurs if the field is read from a file in the Electromagnetic Data Exchange (EDX) format . As the conversion to dB is carried out by the same equations as above the results seem different. Electric fields read from files in EDX format will be presented in a dB-scale which expresses $\text{dB}^{(V/m)^2}$ while fields read from files in TICRA format is expressed in dB^{watt} .

The electric field given in GRASP units, \vec{E}_{GRASP} , are related to the electric field given in SI units, \vec{E}_{SI} , by the relation

$$\vec{E}_{GRASP} = \frac{1}{k\sqrt{2}\zeta} \vec{E}_{SI} \quad (4)$$

and, with the same notation for the magnetic field,

$$\vec{H}_{GRASP} = \frac{1}{k} \sqrt{\frac{\zeta}{2}} \vec{H}_{SI} \quad (5)$$

where k is the wavenumber and ζ is the free space impedance. It is seen that the conversion factors are frequency dependent.

Links

[Alphabetical List of Classes and Command Types](#)

[Classes](#)

[Command Types](#)

[File Extensions](#)

[File Formats](#)

[Appendices](#)

[Reference Section](#)

[Contents](#)

9.6 File Extensions

Purpose

For the files used for input and output in GRASP it is recommended to use different file extensions depending on the content of the files. The recommended file extensions are listed below together with a short description.

Project Control:

- .tor Object repository (input of the objects describing the case to be calculated)
- .tci Command input (calculation commands)
- .g9p GRASP project file (input)
- .out General output file from a GRASP calculation
- .log General log file from a GRASP calculation

Reflector Descriptions:

- .rim Numerical rim, input to class *Tabulated Rim, xy-Input*, output from class *Rim Data Output*. The content is described in *Reflector Rim Data*.
- .rsf Numerical, rotationally symmetric surface, input to class *Rotationally Symmetric*. The content is described in *Rotationally Symmetric Surface*.
- .sfc Numerical general surface, input to class *Regular xy-Grid* and *Unfurlable Surface, Button Attached*, output from class *Surface Data Output*. The content of these files is described in *Surface Data in Rectangular Grid*. Input also to class *Irregular xy-Grid, Pseudo Splines*. The content of this file is described in *Surface Data in Irregular Points*.
- .zer Surface given by Zernike modes, input to class *Zernike Surface*. The content is described in *Surface Data as Zernike Modes*.
- .spl Surface given by cubic splines, input to class *Spline Surface*. The content is described in *Surface Defined by Cubic Splines*.

General Geometries

- .msh A tabulated meshed geometry, input to class *Tabulated Mesh* and output from command *Write MoM Mesh*.

Material Properties

- .tep Electrical properties, input to class *Tabulated Electrical Properties*, output from class *Electrical Properties Data Output*. The content is described in *Tabulated Electrical Properties for Scatterers*.

Source (Feed) Positions and Excitations:

- .isp Irregular source (feed) positions, input to class *Tabulated General Array*. The content is described in *Irregularly Spaced Feed Data*.
- .rsp Regular source (feed) positions, input to class *Tabulated Planar Grid Array*. The content is described in *Feed Data in Regular Grid*.
- .exi Feeds/beams excitations, input to class *Tabulated Planar Grid Array* and *Tabulated General Array*. The content is described in *Array Element Excitation Coefficients*.

- .sph Spherical modes for a feed, input to class *Tabulated SWE Coefficients*, output from class *Spherical Wave Expansion (SWE)*. The content of the file is described in *Spherical Wave ab-Coefficients* and *Spherical Wave Q-Coefficients*.
- .cur Currents on a scatterer, output from and input to the classes under *PO Analysis* and *MoM*. The file is shortly described in *Standard Currents File*.
- .bem Beam pointing directions (in u,v), input to class *Spherical Grid*. The content is described in *Beam Grid Directions*.
- .sta Far-field directions, input to class *Tabulated uv-Points*. The content is described in *Field Directions*.

Patterns and Fields:

- .cut Field pattern in cuts, output from class *Spherical Cut*, *Cylindrical Cut*, *Planar Cut* and *Surface Cut*. The content is described in *Field Data in Cuts*. Also input to class *Tabulated Pattern* with the content as described in *Tabulated Pattern Data*.
- .grd Field pattern in grid, output from class *Spherical Grid*, *Cylindrical Grid*, *Planar Grid* and *Surface Grid*. The content is described in *Field Data in Rectangular Grid*.
- .fmt Field at given points, output from class *Tabulated uv-Points*. The content is described in *Field Data in Tabulated Directions*.

Scattering and Impedance Matrices:

- .par The S, Y, and Z-parameters for a system with voltage generators, output from class *Voltage Generator*. The content is described in *S, Y, and Z-Parameters*

Ray Traces:

- .trc Determined GTD ray traces from a source to field points, output from and input to class *Multi-Reflector GTD*. The file is shortly described in *Ray Traces*.

Plots:

- .xyz 3D plot in XYZ-points, output from class *Plot Settings (Obsolete)*. The content is described in *Three-Dimensional Plotter Data as XYZ-Values*.
- .cpf Definitions of currents for *Currents Colour Plot*, output from class *MoM*, cf. *Currents Colour Plot File*.

Coordinate Systems

- .cor Definition of a coordinate system, input to class *Tabulated Coordinate System*. The content is described in *Definition of Coordinate Systems*.

Links

[Alphabetical List of Classes and Command Types](#)

[Classes](#)

[Command Types](#)

[Applicable Units](#)

[File Formats](#)

Appendices

Reference Section

Contents

9.7 File Formats

This section contains a description of the data files used by GRASP. The [List of File Formats](#) is given below.

GRASP reads and generates a number of data files which serve as links to other programs. The format of the files is compatible with previous versions of GRASP. Numbers and character strings are read record by record in free format (apart from two file types which shall be written in fixed format) meaning that the format is arbitrary except that there must be at least one space between each data value (numbers and character strings) in the record. Alternatively, a comma is allowed as separator, possibly surrounded by blanks. Because the blanks may be used as separators blanks cannot be accepted within a character string. Character strings are denoted `string` in the following (under the column heading `Format`). However, in heading and text records blanks are accepted. These are specified as `characters` in the following.

The format of the input records is described in the following, record by record. A record number is given (under the column heading `Record`) but this is only used for reference here. Thus, there may be several records in the input file with the same record number.

The contents of the input records are given (under the column heading `Contents`) in one of two ways:

1. The record must contain the specific stated content. This is then given in brackets (which must be included). Such records are used as headers. Example:

`[title]`

and the seven characters “[title]” must be inserted in the input record.

2. A list of variables is given. The types of the variables are given in the column `Format`. Commonly used types are `characters`, `string`, `integer` and `real number`. Example:

`NP, KSPACE, KTIP`

(3 integers)

and three integers must be specified in this input record.

The values to be given for the variables are explained in the text below each input record.

The input of an array of numbers (e.g. coordinates to points in a plane) is indicated in the following way:

`((X(I), Y(I), I=1, NP)`

(real numbers)

which means that the vectors X and Y must be given in the sequence:

`X(1), Y(1), X(2), Y(2), ..., X(NP), Y(NP).`

It is insignificant how many numbers are given in each record as long as they are stored in the correct order and separated by blanks or by commas.

The sequence in which a two-dimensional array is stored is indicated by constructions like

$(Z(I,J), J = 1, NY), I = 1, NX)$ (real numbers)

meaning that the second index J is running faster than the first index, i.e. the sequence becomes:

$Z(1,1), Z(1,2), \dots, Z(1,NY), Z(2,1), \dots, Z(2,NY), \dots, Z(NX,1), \dots, Z(NX,NY).$

The following types of data files are defined together with the applied *File Extensions*:

F1	<i>Reflector Data</i>	
F1.1	<i>Reflector Rim Data</i>	.rim
F1.2	<i>Rotationally Symmetric Surface</i>	.rsf
F1.3	<i>Individually Defined Gaps</i>	.dat
F1.4	<i>Panel Misalignments</i>	.pp
F1.5	<i>Surface Data in Rectangular Grid</i>	.sfc
F1.6	<i>Surface Data in Irregular Points</i>	.sfc
F1.7	<i>Surface Data as Zernike Modes</i>	.zer
F1.8	<i>Surface Defined by Cubic Splines</i>	.spl
F2	<i>Field Data</i>	
F2.1	<i>Field Data in Cuts</i>	.cut
F2.2	<i>Field Data in Rectangular Grid</i>	.grd
F2.3	<i>Field Data in Tabulated Directions</i>	.fmt
F2.4	<i>Beam Grid Directions</i>	.bem
F2.5	<i>Field Directions</i>	.sta
F3	<i>Currents Stored on File</i>	
F3.1	<i>Standard Currents File</i>	.cur
F3.2	<i>Currents Colour Plot File</i>	.cpf
F4	<i>Tabulated Feed Data</i>	
F4.1	<i>Tabulated Pattern Data</i>	.cut
F4.2	<i>Spherical Wave ab-Coefficients</i>	.sph
F4.3	<i>Spherical Wave Q-Coefficients</i>	.sph
F4.4	<i>Hexagonal Wave Guide Modal Aperture Field</i>	.dat
F5	<i>Array Element Data</i>	
F5.1	<i>Irregularly Spaced Feed Data</i>	.isp
F5.2	<i>Feed Data in Regular Grid</i>	.rsp
F5.3	<i>Array Element Excitation Coefficients</i>	.exi
F6	<i>Electrical Properties for Scatterers</i>	
F6.1	<i>Tabulated Electrical Properties for Scatterers</i>	.tep
F7	<i>Three-Dimensional Plotter Data</i>	
F7.1	<i>Three-Dimensional Plotter Data as XYZ-Values</i>	.xyz
F8	<i>Definition of Coordinate Systems</i>	
F8.1	<i>Definition of Tabulated Coordinate System</i>	.cor
F9	<i>Coupling Data</i>	
F9.1	<i>Coupling Data in Cuts</i>	.cut

F9.2	<i>Coupling Data in Grids</i>	.grd
F10	<i>General Geometries</i>	
F10.1	<i>Meshed Geometries</i>	.msh
F11	<i>Ray Traces</i>	.trc
F12	<i>S, Y, and Z-Parameters</i>	.par
F13	<i>Gauss-Laguerre Beam Coefficients</i>	.gbc

Links

[Alphabetical List of Classes and Command Types](#)

[Classes](#)

[Command Types](#)

[Applicable Units](#)

[File Extensions](#)

[Appendices](#)

[Reference Section](#)

[Contents](#)

F1 Reflector Data

The following reflector data file types are available:

- *Reflector Rim Data*
- *Rotationally Symmetric Surface*
- *Individually Defined Gaps*
- *Panel Misalignments*
- *Surface Data in Rectangular Grid*
- *Surface Data in Irregular Points*
- *Surface Data as Zernike Modes*
- *Surface Defined by Cubic Splines*

Links

[List of File Formats](#)

[Reference Section](#)

F1.1 Reflector Rim Data

Format

The rim of a reflector can be specified by numerical data in a file. Different formats of the rim data may be applied and are controlled by the attribute *file_form* in one of the *Tabulated Rim* objects.

If *file_form* is specified to 'free_format', a general tabulated format may be applied. The format of the file, e.g. the row and the column numbers, is further specified in the *Tabulated Rim* objects.

If *file_form* is specified to 'old_grasp' the format described below shall be applied. This format is also used by the obsolete class *Tabulated Rim* and the file extension is *.rim*. The rim is specified by a number of points in (x, y) -coordinates or polar coordinates (ρ, ϕ) . In the latter case the format distinguishes between equispaced and unequally spaced points in ϕ . In all cases the points must be ordered so that ϕ increases monotonously i.e. the points must be ordered counter-clockwise.

Record Contents	Format
1 TEXT	(characters)

TEXT – Record with identification text

2 NP, KSPACE, KXYRPH	(3 integers)
----------------------	--------------

NP	= Number of points in which the rim contour is specified.
KSPACE	= 0 – The edge points are equispaced in ϕ . = 1 – The edge points are unequally spaced in ϕ .
KXYRPH	= 1 – The points are defined by (x, y) -coordinates. = 2 – The points are defined by (ρ, ϕ) -coordinates.

If (KSPACE=0 and KXYRPH=2) go to record No. 3.1

If (KSPACE=1 and KXYRPH=1) go to record No. 3.2

If (KSPACE=1 and KXYRPH=2) go to record No. 3.3

The combination (KSPACE=0 and KXYRPH=1) is not allowed.

3.1 (RHO(I), I=1, NP)	(real numbers)
-----------------------	----------------

RHO(I) – ρ -value of point No. I at ϕ -angle, $\phi = 360^\circ \cdot (I-1) / NP$

---end of data file---

3.2 (XY(1,I), XY(2,I), I=1, NP)	(real numbers)
---------------------------------	----------------

XY(1,I) - x -coordinate of point No. I
XY(2,I) - y -coordinate of point No. I

---end of data file---

3.3 (RPH(1,I), RPH(2,I), I=1,NP) (real numbers)

RPH(1,I) - ϕ -value in degrees of point No. I
RPH(2,I) - ρ -value of point No. I

---end of data file---

Links

Reflector Data

List of File Formats

Reference Section

F1.2 Rotationally Symmetric Surface

Format

The format is used in the definition of surfaces of class *Rotationally Symmetric* and the file extension is .rsf. The surface is rotationally symmetric around the z -axis and specified by a number of points on a radial arc where the distance is defined by

$$\rho = \sqrt{x^2 + y^2} \quad (6)$$

The format is:

Record Contents	Format
-----------------	--------

1 TEXT	(characters)
--------	--------------

TEXT – Record with identification text

2 NP, KSPACE, KTIP	(3 integers)
--------------------	--------------

NP KSPACE = 0 KSPACE = 1 KTIP = 0 KTIP = 1	– Number of points, at least 3. – Equispaced points in ρ . – Unequally spaced points in ρ . – Surface will have tangent parallel to the xy-plane at $\rho = 0$. – Reflector will have tip at $\rho = 0$.
--	---

If KSPACE = 1, go to record No. 3.1 else go to record No. 3.2.

3.1 (RHO(I), Z(I), I=1,NP)	(real numbers)
----------------------------	----------------

RHO Z	– Radial distance from z -axis to I'th point. Must be given in increasing order. – z -coordinate at the I'th point.
----------	---

---end of file for KSPACE=1---

3.2 RS, RE	(2 real numbers)
------------	------------------

RS RE	– Start value and – End value of radial distance from z -axis to surface point.
----------	---

3.3 (Z(I), I=1,NP)	(real numbers)
--------------------	----------------

Z(I)	z -coordinate at the distance ρ from the z -axis where $\rho = RS + \Delta\rho \cdot (I-1)$ and $\Delta\rho = (RE - RS) / (NP-1)$
------	--

---end of file for KSPACE=0---

Links

Reflector Data

List of File Formats

Reference Section

F1.3 Individually Defined Gaps

Format

Several gaps can be specified by the user. The format is used by the class *Individually Defined Panels* and the standard file extension is *.dat*.

The following format shall be applied. Text shown in brackets must be written as shown, including the brackets.

Record Contents	Format
-----------------	--------

1	[VERSION]	(9 characters)
---	-----------	----------------

2	FILE_VERSION	(characters)
---	--------------	--------------

Identification of file type and version, presently only the following text is accepted:

TICRA Gap File, Version 1

3	[TITLE]	(7 characters)
---	---------	----------------

4	TITLE	(characters)
---	-------	--------------

One or more records with arbitrary identification text.

5	[LENGTH UNIT]	(13 characters)
---	---------------	-----------------

6	LENGTH_UNIT	(characters)
---	-------------	--------------

Unit of length (see valid units in *Applicable Units*) for the data defining the gap curve and the width of the gap.

The following records 7–9 must be present for each gap defined.

7	[START GAP]	(11 characters)
---	-------------	-----------------

8	(X(I), Y(I), W(I), I=1, NP)	(real numbers)
---	-----------------------------	----------------

NP is the number of points specified on the actual gap curve.

X(I) *x*-coordinate of point I on gap-curve.

Y(I) *y*-coordinate of point I on gap-curve.

W(I) Width of gap at point I on gap-curve. The width is the projected width in the *xy*-plane of the reflector coordinate system.

9 [END GAP] (9 characters)

---end of file---

A simple example of a gap file is shown below. The file defines a single, 10 m long gap with its centre line along the y -axis of the reflector coordinate system. The projected gap width decreases linearly from 0.2 m (at $y = -5$ m) to 0.1 m (at $y = 5$ m):

```
[version]
Ticra Gap File, Version 1
[title]
test gap file
[length unit]
m
[start gap]
0. -5. 0.20
0. -4. 0.19
0. -3. 0.18
0. -2. 0.17
0. -1. 0.16
0. 0. 0.15
0. 1. 0.14
0. 2. 0.13
0. 3. 0.12
0. 4. 0.11
0. 5. 0.10
[end gap]
```

Links

[Reflector Data](#)

[List of File Formats](#)

[Reference Section](#)

F1.4 Panel Misalignments

Format

Misalignments can be specified for one or several panels of a *Panels in Polar Grid* reflector. The standard file extension is *.pp*.

The following format shall be applied. Text shown in brackets must be written as shown, including the brackets.

Record Contents	Format
-----------------	--------

1	[VERSION]	(9 characters)
---	-----------	----------------

2	FILE_VERSION	(characters)
---	--------------	--------------

Identification of file type and version, presently only the following text is accepted:

TICRA Panel Misalignments File, Version 1

3	[TITLE]	(7 characters)
---	---------	----------------

4	TITLE	(characters)
---	-------	--------------

Records with arbitrary identification text.

5	[LENGTH UNIT]	(13 characters)
---	---------------	-----------------

6	LENGTH_UNIT	(characters)
---	-------------	--------------

Unit of length (see valid units in *Applicable Units*) for the data defining the control points and the misalignments.

The following records 7-10 must be present for each panel for which misalignments are specified.

7	[PANEL]	(7 characters)
---	---------	----------------

8	I_RING, I_PANEL	(2 integers)
---	-----------------	--------------

I_RING	The number of the ring to which the panel belongs.
I_PANEL	The number of the panel in this ring (cf. the remarks in the description of class <i>Panels in Polar Grid</i>)

9 (X(I), Y(I), DX(I), DY(I), DZ(I), I=1, NP) (real numbers)

NP is the number of control points specified on the actual panel.

X(I) *x*-coordinate of control point No. I.

$Y(I)$ y -coordinate of control point No. I.

DX(I) *x*-coordinate of misalignment vector for point No. I.

DY(I) *y*-coordinate of misalignment vector for point No. I.

DZ(I) *z*-coordinate of misalignment vector for point No. I.

10 [END GAP] (11 characters)

---end of file---

A simple example of an misalignments file is shown below. Misalignments are specified for two panels, namely panel 1 and 3 in the second ring of panels. At each panel four control points are defined. The displacement vectors are all parallel to the z -axis of the reflector coordinate system as $dx = 0$ and $dy = 0$.

```
[version]
Ticra Panel Misalignment File, Version 1
[title]
Reflector with 8 panels
[length unit]
m
[panel]
2 1
 3.300000 -3.300000 0.00000 0.00000 -0.04000
 7.000000 -7.000000 0.00000 0.00000 -0.06000
 7.000000 7.000000 0.00000 0.00000 0.06000
 3.300000 3.300000 0.00000 0.00000 0.04000
[end panel]
[panel]
2 3
-3.300000 3.300000 0.00000 0.00000 -0.04000
-7.000000 7.000000 0.00000 0.00000 0.06000
-7.000000 -7.000000 0.00000 0.00000 0.06000
-3.300000 -3.300000 0.00000 0.00000 -0.04000
[end panel]
```

Links

Reflector Data

List of File Formats

Reference Section

F1.5 Surface Data in Rectangular Grid

Format

A surface defined by a function of the form $z = f(x, y)$ can be specified by z -values in a rectangular grid. This format is used by the class *Regular xy-Grid* and the file extension is *.sfc*.

The format is:

Record Contents	Format
-----------------	--------

1	TEXT	(characters)
---	------	--------------

TEXT – Record with identification text

2	XS, YS, XE, YE	(4 real numbers)
---	----------------	------------------

XS	– Starting x -value
YS	– Starting y -value
XE	– End x -value
YE	– End y -value

3	NX, NY	(2 integers)
---	--------	--------------

NX	– Number of x -values
NY	– Number of y -values

4	((Z(I, J), J=1, NY), I=1, NX)	(real numbers)
---	-------------------------------	----------------

Z(I, J)	z -value of reflector surface at the point (x, y) where
	$x = XS + \Delta x (I-1)$
	$y = YS + \Delta y (J-1)$
	$\Delta x = (XE-XS) / (NX-1)$
	$\Delta y = (YE-YS) / (NY-1)$

This record may be divided into several data lines with at least one z -value on each data line.

---end of file---

Links

[Reflector Data](#)

[List of File Formats](#)

[Reference Section](#)

F1.6 Surface Data in Irregular Points

Format

The surface defined as a function of the form $z = f(x, y)$ is specified by the z -values in an irregular set of (x, y) specified points. This format is used by the class *Irregular xy-Grid, Pseudo Splines* and the file extension is *.sfc*.

Record Contents	Format
-----------------	--------

1 TEXT	(characters)
--------	--------------

TEXT – Record with identification text

2 N_POINTS	(1 integer)
------------	-------------

N_POINTS – Number of (x, y, z) -values

3 X(I), Y(I), Z(I), I=1,N_POINTS	(3 real numbers)
----------------------------------	------------------

X(I), Y(I), Z(I)

the (x, y, z) -coordinates of the I'th surface point.

---end of file---

Links

[Reflector Data](#)

[List of File Formats](#)

[Reference Section](#)

F1.7 Surface Data as Zernike Modes

Format

The surface is specified by Zernike polynomials using the following Zernike coefficients. This format is used by the class *Zernike Surface* and the standard file extension is *.zer*. The format of the file can be modified by the attributes in the *Zernike Surface* class, but the default format is:

Record Contents	Format
1 TITLE	(characters)
TITLE – Record with identification text	
2 NC	(1 integer)
NC – Number of Zernike coefficients	
3.. M(I), N(I), C1(I), C2(I)	(2 integers, 2 real numbers)
Continued for I=1, NC	
Values for the I'th Zernike coefficient	
M – m-index.	
N – n-index.	
C1 – First coefficient.	
C2 – Second coefficient.	

The Zernike coefficients C1 and C2 can be given as amplitude and rotation angle in degrees, ‘amp_deg’ (TICRA format), or as even and odd symmetric components, ‘even_odd’ (old POS & POD format). The definition used must be specified in the *Zernike Surface* object by the attribute *mode_definition*.

---end of file---

Links

[Reflector Data](#)

[List of File Formats](#)

[Reference Section](#)

F1.8 Surface Defined by Cubic Splines

Format

A surface can be specified by cubic splines using the following spline coefficients. This format is used by the class *Spline Surface* and the standard file extension is *.spl*.

The format is:

Record Contents	Format
1 TEXT	(characters)
TEXT – Record with identification text	
2 XS, YS, XE, YE	(4 real numbers)
XS – Starting x -value for spline definition area	
YS – Starting y -value for spline definition area	
XE – End x -value for spline definition area	
YE – End y -value for spline definition area	
3 NX, NY	(2 integers)
NX – Number of spline coefficients along x	
NY – Number of spline coefficients along y	
4 ((C(I,J), J=1,NY), I=1,NX)	(real numbers)
C(I,J) – Spline coefficient $C_{I,J}$. Index I corresponds to the one-dimensional spline in the x -direction and index J to the one-dimensional spline in the y -direction.	
---end of file---	

Links

[Reflector Data](#)

[List of File Formats](#)

[Reference Section](#)

F2 Field Data

The file formats for field values in cuts, uv -grids and irregularly arranged points are described in the following sections together with the file formats for field directions.

- *Field Data in Cuts*
- *Field Data in Rectangular Grid*
- *Field Data in Tabulated Directions*
- *Beam Grid Directions*
- *Field Directions*

Links

[List of File Formats](#)

[Reference Section](#)

F2.1 Field Data in Cuts

Format

The file is generated by objects of the classeses *Spherical Cut*, *Planar Cut*, *Surface Cut*, and *Cylindrical Cut*for storing field data in cuts. The file extension is *.cut*.

A cut consists of records of types 1, 2 and 3 as described below. If more than one cut is contained in a file all records must be repeated for each cut.

Record Contents	Format
-----------------	--------

1	TEXT	(characters)
---	------	--------------

TEXT – Record with identification text

2	V_INI, V_INC, V_NUM, C, ICOMP, ICUT, NCOMP numbers, one integer, 1 real number, 3 integers)	(2 real
---	--	---------

The V_ and C values are angles in degrees and their definition is controlled by the parameter ICUT:

- V_INI – Initial value.
- V_INC – Increment.
- V_NUM – Number of values in cut.
- C – Constant.
- ICOMP – Polarisation control parameter.
- ICUT – Control parameter of cut.
- NCOMP – Number of field components.

For spherical cut:

The V_ and C values are angles in degrees and their definition is controlled by the parameter ICUT:

- ICUT =1 A standard polar cut where ϕ is fixed (C) and θ is varying (V_)
- ICUT =2 A conical cut where θ is fixed (C) and ϕ is varying (V_).

The field components F1, F2 are specified by the parameter ICOMP. For near fields the third component F3 always contains the radial E_ρ -component. F1 and F2 are for

ICOMP

- =1 Linear E_θ and E_ϕ .
- =2 Right hand and left hand circular (E_{rhc} and E_{lhc}).
- =3 Linear E_{co} and E_{cx} (Ludwig's third definition).
- =4 Linear along major and minor axes of the polarisation ellipse, E_{maj} and E_{min} .
- =5 XPD fields: E_θ/E_ϕ and E_ϕ/E_θ .
- =6 XPD fields: E_{rhc}/E_{lhc} and E_{lhc}/E_{rhc} .
- =7 XPD fields: E_{co}/E_{cx} and E_{cx}/E_{co} .
- =8 XPD fields: E_{maj}/E_{min} and E_{min}/E_{maj} .
- =9 Total power $|\bar{E}|$ and $\sqrt{E_{rhc}/E_{lhc}}$.

If ICOMP is negative the above polarisation definitions apply according to ICOMP's absolute value, but the negative sign indicates that the polarisation is not defined in the cut coordinate system (attribute polarisation_modification has been set to 'on' in the cut-defining object, see class *Spherical Cut*).

- NCOMP - Number of field components.
- =2 The file contains two field components for each point as specified above.
- =3 When the field is a near field the file also contains the third radial component, E_ρ .

For planar cut and surface cut:

The definition of the V_ and C values is controlled by the parameter ICUT:

- ICUT =1 A radial cut where ϕ is the fixed cut angle in degrees (C) and ρ is the varying distance in user defined units (V_).
- ICUT =2 A circular cut where ρ is the fixed distance in user defined units (C) and ϕ is the varying angle in degrees (V_).

The field components F1, F2 are specified by the parameter ICOMP. The third component F3 always contains the z-component (E_z). For ICOMP:

- =1 Linear E_ρ and E_ϕ .
 - =2 Right hand and left hand circular (E_{rhc}, E_{lhc}).
 - =3 Linear components along x and y , E_{co} and E_{cx} .
 - =4 Linear along major and minor axes of the polarisation ellipse, E_{maj} and E_{min} , see below.
 - =5 XPD fields: E_ρ/E_ϕ and E_ϕ/E_ρ .
 - =6 XPD fields: E_{rhc}/E_{lhc} and E_{lhc}/E_{rhc} .
 - =7 XPD fields: E_{co}/E_{cx} and E_{cx}/E_{co} .
 - =8 XPD fields: E_{maj}/E_{min} and E_{min}/E_{maj} .
 - =9 Total power $|\bar{E}|$ and $\sqrt{E_{rhc}/E_{lhc}}$.
- NCOMP - Number of field components.
- =3 The file contains three field components for each point as specified above.

For cylindrical cut:

The definition of the V and C values are controlled by the parameter ICUT:

ICUT =1 An axial cut where ϕ is the fixed angle of the cut in degrees (C) and z is the varying distance in user defined units (V_).

ICUT =2 A circular cut where z is the fixed distance in user defined units (C) and ϕ is the varying angle in degrees (V_).

The field components F1, F2 are specified by the parameter ICOMP. The third component F3 always contains the radial component (E_ρ). ICOMP:

=2 Right and left hand components (E_{rhc}, E_{lhc}). Based on the linear components as given in the next record.

=3 Linear E_ϕ, E_z .

=4 Linear along major and minor axes of the polarisation ellipse, E_{maj} and E_{min} , see below.

=6 XPD fields: E_{rhc}/E_{lhc} and E_{lhc}/E_{rhc} .

=7 XPD fields: E_z/E_ϕ and E_ϕ/E_z .

=8 XPD fields: E_{maj}/E_{min} and E_{min}/E_{maj} .

=9 Total power $|\vec{E}|$ and $\sqrt{E_{rhc}/E_{lhc}}$.

NCOMP - Number of field components.

=3 The file contains three field components for each point as specified above.

Record No. 3 and the following records contain the field values

If NCOMP=2

3 (F1(I), F2(I), I = 1, V_NUM) (4 real numbers on each record)

F1,F2 - Complex arrays containing the two components of the field for the I'th data point.
 $V = V_{INI} + V_{INC} \cdot (I-1)$

If NCOMP=3

3 (F1(I), F2(I), F3(I), I = 1, V_NUM) (6 real numbers on each record)

F1,F2,F3 - Complex arrays containing the three components of the field for the I'th data point.
 $V = V_{INI} + V_{INC} \cdot (I-1)$

---end of data file---

For ICOMP=1, 2, 3, 5, 6 or 7 F1, F2 (and optionally F3) contain the real and imaginary parts of the field in linear scale.

For ICOMP=4:

Real part of F1 is the major axis of the polarisation ellipse (linear scale).

Real part of F2 is the minor axis of the polarisation ellipse (linear scale).

Imaginary parts of F1 and F2 are zero.

For ICOMP=8

Real part of F1 is the major axis divided by the minor axis of the polarisation ellipse (linear scale).

Real part of F2 is the minor axis divided by the major axis of the polarisation ellipse (linear scale).

Imaginary parts of F1 and F2 are zero.

For ICOMP=9

Real part of F1 is the total power $|\bar{E}|$ of the field (linear scale)

Imaginary part of F1 is zero.

F2 is the complex square root of the ratio rhc/lhc.

The phase of this value is the rotation angle of the polarisation ellipse.

Links

[Field Data](#)

[List of File Formats](#)

[Ludwig's 3rd Definition of Polarization](#)

[Reference Section](#)

F2.2 Field Data in Rectangular Grid

Format

This format is used for storing field values in a rectangular grid. Files of this type are generated by objects of the classes *Spherical Grid*, *Planar Grid*, *Surface Grid*, and *Cylindrical Grid*, and the file extension is *.grd*. Data points may be located either on a sphere, on a plane, on a scatterer surface or on a cylindrical surface.

If the field points are located on a sphere the direction to a field point \hat{r} is related to the polar angles θ and ϕ by

$$\hat{r} = \hat{x} \sin \theta \cos \phi + \hat{y} \sin \theta \sin \phi + \hat{z} \cos \theta \quad (7)$$

If the field points are located on a plane it is assumed that the plane is parallel to the xy -plane of the output coordinate system so that a point can be defined by its x - and y -coordinates.

If the field points are located on a cylinder the vector to a field point \vec{r} is defined by

$$\vec{r} = \hat{x} \rho \cos \phi + \hat{y} \rho \sin \phi + \hat{z} z \quad (8)$$

where ρ is the constant radius of the cylinder and ϕ and z are the two variables that specifies a point on the cylinder.

The field points are in all cases parameterised by two variables which are generally denoted X and Y and which run from XMIN to XMAX and from YMIN to YMAX, respectively. The file is organised so that X is varying faster than Y. The variables X and Y should be considered as general names for the actual variables which may e.g. be ϕ and θ for points on a sphere or any of the other options as specified by IGRID below.

The file format is:

Record Contents	Format
1 TEXT	(characters)
2 KTYPE	(integer)
3 NSET, ICOMP, NCOMP, IGRID	(4 integers)

TEXT – Record with identification text. This record is repeated until a record containing ++++ as the first 4 characters is reached.

KTYPE = 1 – standard format for 2D grid. For files used in GRASP this variable is always 1.

NSET – Number of field sets or beams.
 ICOMP – Control parameter of field components.
 NCOMP – Number of components.
 IGRID – Control parameter of field grid type.

For spherical grid:

The field components F1, F2 are controlled by the parameter ICOMP. For near fields the third component F3 always contains the radial E_r component.

ICOMP

- =1 linear E_θ and E_ϕ .
- =2 Right and left hand circular components (rhc,lhc).
- =3 Linear co and cx components (Ludwig's third definition).
- =4 Major and minor axes of polarisation ellipse.
- =5 XPD fields: E_θ/E_ϕ and E_ϕ/E_θ .
- =6 XPD fields: rhc/lhc and lhc/rhc.
- =7 XPD fields: co/cx and cx/co.
- =8 XPD fields: major/minor and minor/major.
- =9 total power $|\bar{E}|$ and $\sqrt{rhc/lhc}$.

If ICOMP is negative the above polarisation definitions apply according to ICOMP's absolute value, but the negative sign indicates that the polarisation is not defined the grid coordinate system (attribute *polarisation_modification* has been set to 'on' in the grid-defining object, see class *Spherical Grid*).

- NCOMP – Number of field components.
- =2 The file contains two field components for each point as specified above.
- =3 If the field is a near field the file also contains the third radial component.

IGRID - Type of field grid.

=1 uv-grid:
 $(X, Y) = (u, v)$ where u and v are the two first coordinates of the unit vector to the field point. Hence,
 $\hat{r} = (u, v, \sqrt{1 - u^2 - v^2})$
 u and v are related to the spherical angles by $u = \sin \theta \cos \phi$, $v = \sin \theta \sin \phi$.

=4 Elevation over azimuth:
 $(X, Y) = (\text{Az}, \text{El})$, where Az and El define the direction to the field point by
 $\hat{r} = -\sin \text{Az} \cos \text{El}, \sin \text{El}, \cos \text{Az} \cos \text{El}$.

=5 Elevation and azimuth:
 $(X, Y) = (\text{Az}, \text{El})$, where Az and El define the direction to the field point through the relations
 $\text{Az} = -\theta \cos \phi$, $\text{El} = \theta \sin \phi$
to the spherical angles θ and ϕ .

=6 Azimuth over elevation:
 $(X, Y) = (\text{Az}, \text{El})$, where Az and El define the direction to the field point by
 $\hat{r} = -\sin \text{Az}, \cos \text{Az} \sin \text{El}, \cos \text{Az} \cos \text{El}$.

=7 $\theta\phi$ -grid:
 $(X, Y) = (\phi, \theta)$, where θ and ϕ are the spherical angles of the direction to the field point.

=9 Azimuth over elevation, EDX definition:
 $(X, Y) = (\text{Az}, \text{El})$, where Az and El define the direction to the field point by
 $\hat{r} = \sin \text{Az} \cos \text{El}, \sin \text{El}, \cos \text{Az} \cos \text{El}$.
(This format is not implemented in the Post-Processor).

=10 Elevation over azimuth, EDX definition:
 $(X, Y) = (\text{Az}, \text{El})$, where Az and El define the direction to the field point by
 $\hat{r} = \sin \text{Az}, \cos \text{Az} \sin \text{El}, \cos \text{Az} \cos \text{El}$.
(This format is not implemented in the Post-Processor).

For planar grid:

The field components F1, F2 are controlled by the parameter ICOMP. The third component F3 always contains the z-component (E_z).

ICOMP

=1 Linear $E\rho$ and $E\phi$.
 =2 Right and left hand circular components.
 (rhc, lhc) .
 =3 Linear components along x- and y-direction.
 =4 Major and minor axes of polarisation ellipse.
 =5 XPD fields: $E\rho/E\phi$ and $E\phi/E\rho$.
 =6 XPD fields: rhc/lhc and lhc/rhc .
 =7 XPD fields: Ex/Ey and Ey/Ex .
 =8 XPD fields: major/minor and minor/major.
 =9 total power $|\vec{E}|$ and $\sqrt{rhc/lhc}$.

NCOMP - Number of field components.
 =3 The file contains three field components for each point as specified above.

IGRID - Type of field grid
 =2 $\rho\phi$ -grid:
 $(X, Y) = (\rho, \phi)$ where ρ and ϕ are related to the x- and y-coordinates by
 $x = \rho \cos \phi, y = \rho \sin \phi$
 =3 xy-grid:
 $(X, Y) = (x, y)$. X and Y are identical to the xy-coordinates on the plane.

For surface grid:

The field components F1, F2 are controlled by the parameter ICOMP. The third component F3 always contains the z-component (Ez).

ICOMP
 =1 Linear $E\rho$ and $E\phi$.
 =2 Right and left hand circular components.
 (rhc, lhc) .
 =3 Linear components along x- and y-direction.
 =4 Major and minor axes of polarisation ellipse.
 =5 XPD fields: $E\rho/E\phi$ and $E\phi/E\rho$.
 =6 XPD fields: rhc/lhc and lhc/rhc .
 =7 XPD fields: Ex/Ey and Ey/Ex .
 =8 XPD fields: major/minor and minor/major.
 =9 total power $|\vec{E}|$ and $\sqrt{rhc/lhc}$.

NCOMP - Number of field components.
 =3 The file contains three field components for each point as specified above.

IGRID - Type of field grid
 =3 xy-grid:
 $(X, Y) = (x, y)$. X and Y are identical to the xy-coordinates on the plane.

For cylindrical grid:

The field components F1, F2 are controlled by the parameter ICOMP. The third component F3 always contains the radial component (E_ρ) .

ICOMP

- =2 Right and left hand circular components (rhc,lhc) based on the linear components below.
- =3 Linear (E_ϕ, E_z) .
- =4 Major and minor axes of polarisation ellipse.
- =6 rhc/lhc and lhc/rhc.
- =7 E_z/E_ϕ and E_ϕ/E_z .
- =8 major/minor and minor/major.
- =9 total power $|\vec{E}|$ and $\sqrt{rhc/lhc}$.

NCOMP - Number of field components.

- =3 The file contains three field components foreach point as specified above.

IGRID - Type of field grid

- =8 ϕz -grid:
 $(X, Y) = (\phi, z)$ where ϕ and z are the cylindrical parameters of the vector to the field point.

4 (IX(I), IY(I), I=1, NSET) (2 integers on each record)

IX, IY - Centre of set or beam No. I. See the following record for explanation.

All the following records are repeated NSET times

5 XS, YS, XE, YE (4 real numbers)

Limits of 2D grid. The unit of X and Y follows the settings in the generating project, angles are in degrees. The grid points (X, Y) run through the values

$$\begin{aligned} X &= XCEN + XS + DX * (I-1) \\ Y &= YCEN + YS + DY * (J-1) \end{aligned}$$

where

$$DX = (XE-XS) / (NX-1), \quad DY = (YE-YS) / (NY-1)$$

and

$$XCEN = DX * IX, \quad YCEN = DY * IY$$

The number of grid values NX and NY and the range of the index I and J are defined in the following records.

6 NX, NY, KLIMIT (3 integers)

NX - Number of columns
NY - Number of rows
KLIMIT - Specification of limits in a 2D grid
 =0 Each row contains data for all NX columns.
 =1 The number of data points for each row is
 defined in the following records.

The following records 7 and 8 are repeated NY times ($J=1, NY$) for each beam

If KLIMIT = 0 skip to records No. 8 with IS = 1 and IE = NX

If KLIMIT = 1 record 7 is read

7 IS, IN (2 integers)

IS - Column number of first data point in row J
 IN - Number of data points in row J

If KLIMIT = 0, IS and IN are always assumed to be 1 and NX, respectively.

If $IN = 0$ skip records No. 8. and repeat from record No. 7.

IF IN > 0 continue at record No. 8.2 or 8.3 with IE = IS+IN-1.

If NCOMP = 2

8.2 (F1(I), F2(I)), I = IS, IE (4 real numbers on each record)

F1, F2 - Complex field with two components.

If NCOMP = 3

8.3 (F1(I), F2(I), F3(I)), I = IS, IE(6 real numbers on each record)

F_1, F_2, F_3 - Complex field with three components.

---end of data file---

For ICOMP=1, 2, 3, 5, 6 or 7 F1, F2 (and optionally F3) contain the real and imaginary parts of the field in linear scale.

For ICOMP=4 F1 and F2 contain

Real part of F1 is major axes of polarisation ellipse
(linear scale)

Real part of F2 is minor axes of polarisation ellipse
(linear scale)

Imaginary part of F1 and F2 is zero

For ICOMP=8

Real part of F1 is the major axis divided by the minor axis of the polarisation ellipse (linear scale).
Real part of F2 is the minor axis divided by the major axis of the polarisation ellipse (linear scale).
Imaginary parts of F1 and F2 are zero.

For ICOMP=9 F1 and F2 contain

Real part of F1 is total power $|\bar{E}|$ of field (linear scale)
Imaginary part of F1 is zero
F2 is the complex square root of the ratio rhc/lhc.
The phase of this value is the rotation angle of the polarisation ellipse.

Links

[Field Data](#)

[List of File Formats](#)

[Ludwig's 3rd Definition of Polarization](#)

[Reference Section](#)

F2.3 Field Data in Tabulated Directions

Format

This file is used for storing field data in arbitrarily arranged far-field directions. The far-field directions must be provided in another data file, see *Field Directions*. Files of this type are generated by the object class *Tabulated uv-Points* and the file extension is *.fmt*.

The file format is:

Record Contents	Format
-----------------	--------

1 TITLE	(characters)
--------------	--------------

TITLE – Record with identification text

2 N_SETS, N_PARTS	(2 integers)
------------------------	--------------

N_SETS – Number of field sets (beams) in the file.
 N_PARTS – Number of partitions in a set.

3 N_POINTS	(1 integer)
-----------------	-------------

N_POINTS – Number of directions in the following partition.

4 I_BEAM, BEAM_NAME	(1 integer, characters)
--------------------------	-------------------------

I_BEAM – Index of actual beam.
 BEAM_NAME – Name of actual beam.

5 (E1(I), E2(I), I=1, N_POINTS)	(4 real numbers on each record)
--------------------------------------	---------------------------------

E1, E2 – Complex field, real and imaginary part for each.

Record 5 is repeated the number of times given by N_POINTS in record 3, the partitions of records 3, 4 and 5 are repeated N_PARTS times, and the sets with all the partitions are repeated N_SETS times.

Links

[Field Data](#)

[List of File Formats](#)

[Reference Section](#)

F2.4 Beam Grid Directions

Format

This format is used for storing beam directions around which the grid in the class *Spherical Grid* is centred. The file extension is *.bem*

The file format is:

Record Contents	Format
-----------------	--------

1	TITLE ((characters))
---	----------------------

TITLE – Record with identification text.

This record is repeated until a record containing ++++ as the first 4 characters is reached.

2	GRID_TYPE (string)
---	--------------------

GRID_TYPE – Identification label of the grid type. Labels allowed are

- uv
- elevation_over_azimuth
- elevation_and_azimuth
- azimuth_over_elevation
- theta_phi
- elevation_over_azimuth_EDX
- azimuth_over_elevation_EDX

3	ID(I), X_BEAM(I), Y_BEAM(I) (string, 2 real numbers)
---	--

ID – Identification string for beam No. I.

X_BEAM, Y_BEAM – Direction of the element beam No. I.

The coordinates X_BEAM, Y_BEAM shall, according to the grid type, be given as follows (note the order):

- u, v
- azimuth, elevation
- phi, theta

Angles shall be given in degrees.

This record is repeated for each beam.

Links

[Field Data](#)

[List of File Formats](#)

[Reference Section](#)

F2.5 Field Directions

Format

This format is used for storing field directions. The field directions may be grouped in sets denoted partitions. Data for a partition consists of the records 1 and 2 as described below. If more than one partition is contained in a file the records 1 and 2 must be repeated for each partition. The file extension is `.sta`.

The file format is:

Record Contents	Format
1 N_POINTS	(integer)
N_POINTS	Number of field directions in the partition
2 U,V,GOAL,WEIGHT,IPOL,ROT,ATT,ID	(4 real numbers, 1 integer, 2 real numbers, string)
U	– <i>u</i> -coordinate of field direction.
V	– <i>v</i> -coordinate of field direction.
GOAL	– Optimisation goal in dB. (not used in GRASP, a dummy value must be given)
WEIGHT	– Optimisation weight. (not used in GRASP, a dummy value must be given)
IPOL	– Components to be calculated: =1 Linear co-polar component (co). =2 Linear cross-polar component (cx). =3 Circular right hand component. =4 Circular left hand component. =5 Major axis of polarisation ellipse. =6 Minor axis of polarisation ellipse. =7 The ratio co/cx in dB. =8 The ratio cx/co in dB. =9 The ratio RHC/LHC in dB. =10 The ratio LHC/RHC in dB. =11 The ratio Major/Minor in dB. =12 The ratio Minor/Major in dB.
ROT	– Rotation in degrees of linear polarisation reference.
ATT	– Distance attenuation (positive) in dB relative to the sub-satellite point. (not used in GRASP, a dummy value must be given).
ID	– Optional identification string for the direction.

Dummy values must be given for those parameters not used in GRASP to ensure the file is read correctly. The parameters are included for compatibility with POS.

GRASP will calculate the field components for the given value of IPOL. For instance, if IPOL is 1 or 2, linear field components will be calculated.

Record 2 is repeated the number of times indicated by N_POINTS in record 1.

The data file is terminated by record 1 with N_POINTS = 0.

Links

[Field Data](#)

[List of File Formats](#)

[Reference Section](#)

F3 Currents Stored on File

The following file types for currents stored on file are available:

- *Standard Currents File*
- *Currents Colour Plot File*

Links

[List of File Formats](#)

[Reference Section](#)

F3.1 Standard Currents File

Format

In files of this type, electric and magnetic currents and PTD elements are stored in binary format. The files are generated by objects of the classes under *PO Analysis* and *MoM*. The file extension is *.cur*.

Links

Currents Stored on File

List of File Formats

Reference Section

F3.2 Currents Colour Plot File

Format

In this file, information on electric and magnetic currents can be stored. The information is stored as coefficients to polynomials by which the currents may be determined over its region of definition, typically a scatterer. This is used for plotting currents using objects of class *Currents Colour Plot*. The currents colour plot file is generated by objects of class *MoM* and the file extension is *.cpf*.

Links

Currents Stored on File

List of File Formats

Reference Section

F4 Tabulated Feed Data

The following tabulated feed data file types are available:

- *Tabulated Pattern Data*
- *Spherical Wave ab-Coefficients*
- *Spherical Wave Q-Coefficients*
- *Hexagonal Wave Guide Modal Aperture Field*

Links

[*List of File Formats*](#)

[*Reference Section*](#)

F4.1 Tabulated Pattern Data

Format

This file contains a tabulated feed pattern in spherical cuts typically generated by a measurement or another program. The file is used by the class *Tabulated Pattern* and the file extension is *.cut*.

The spherical cuts are polar cuts for which ϕ is fixed and θ is varying. The data of each cut consists of the records 1, 2 and 3 as described below. The records 1, 2 and 3 must be repeated for each θ -cut in the total pattern with the ϕ -angle increasing from cut to cut. At least four cuts must be given.

If the feed model is used for several frequencies the total patterns with all cuts must be appended to the file for each frequency in the same sequence as the connected frequency object.

A file of this type can be generated by an object of class *Spherical Cut*.

The format is:

Record Contents	Format
1 TITLE	(characters)
TITLE – Record with identification text	
2 THS,DTH,NTH,PHI,ICOMP,ICON,NCOMP	(2 real numbers, 1 integer, 1 real number, 3 integers)
THS	Initial θ -angle in degrees.
DTH	Angular θ -steps in degrees.
NTH	Number of θ -points in cut.
PHI	ϕ -value in degrees of cut.
ICOMP	The polarisation components are: =1 Linear θ and ϕ components. =2 Right hand and left hand circular components. =3 Linear co- and cross-components (Ludwig's 3rd definition, see <i>Ludwig's 3rd Definition of Polarization</i> for the definition, especially at the poles)
ICON	Must be set to 1 for compatibility reasons
NCOMP	Number of components in file. Always = 2
3 YA(I),YB(I)	(4 real numbers on each record)

This record is repeated NTH times.

YA(I)	Complex value of first polarisation component.
YB(I)	Complex value of second polarisation component.

---end of file---

Links

[Tabulated Feed Data](#)

[List of File Formats](#)

[Ludwig's 3rd Definition of Polarization](#)

[Reference Section](#)

F4.2 Spherical Wave ab-Coefficients

Format

This file contains the spherical wave *ab*-coefficients of a feed radiation. Such a file was previously typically generated by an object of class *Spherical Wave Expansion (SWE)*, but the format is now obsolete. The file extension is *.sph*. See *Spherical Wave Q-Coefficients* for the format of the file for the more general *Q*-coefficients.

The spherical-wave expansion is of the form

$$\bar{E}(r, \theta, \phi) = \sum_{m=-M}^{M} \sum_{n=|m|}^{N(m)} (a_{mn} \bar{m}_{mn} + b_{mn} \bar{n}_{mn}) \quad (9)$$

If the feed model is used for several frequencies the total patterns consisting of the records 1, 2, 3 and 4 as described below must be appended to the file with the same sequence as the connected frequency object.

The format is:

Record Contents	Format
1 TITLE	(characters)
TITLE – Record with identification text	
2 MMAX	(integer)
MMAX value of M in Eq. (9)	
3.1 (NNMAX(I), I=1, MMAX+1)	(integers)
NNMAX(I) value of N(m) in Eq. (9) where m = I - 1	
4.1 (A(MN); B(MN), MN=1, 2, ...)	(4 real numbers on each record)
A(MN) a _{mn} in Eq. (9)	
B(MN) b _{mn} in Eq. (9)	

The wave coefficients must be stored in the following sequence for $m = 0, \pm 1, \dots, \pm M$

$$a_{01}, b_{01}, a_{02}, \dots, a_{0N(0)}, b_{0N(0)}$$

(10)

$$a_{11}, b_{11}, a_{-11}, b_{-11}, a_{12}, \dots, a_{1N(1)}, b_{1N(1)}, a_{-1N(1)}, b_{-1N(1)}$$

(11)

⋮

(12)

⋮

(13)

⋮

(14)

$$a_{M,M}, b_{M,M}, a_{-M,M}, b_{-M,M}, a_{M,M+1}, \dots, a_{M,N(M)}, b_{M,N(M)}, a_{-M,N(M)}, b_{-M,N(M)}$$

(15)

Note that the waves with azimuthal index m have the azimuthal variation $\exp(-jm\phi)$.

If the mode coefficients have been determined by the spherical wave expansion program SWEP, the input data are automatically generated in the proper format in a highly efficient way.

Links

[Tabulated Feed Data](#)

[List of File Formats](#)

[Reference Section](#)

F4.3 Spherical Wave Q-Coefficients

Format

This file with extension *.sph* contains the spherical wave *Q*-coefficients as calculated by an object of class *Spherical Wave Expansion (SWE)*. The file is also used as input to an object of class *Tabulated SWE Coefficients*. See *Spherical Wave ab-Coefficients* (obsolete) for the format of the *ab*-coefficients file. For greatest accuracy it is recommended to use the *Q*-coefficients.

The first 8 records contain various header information. Although some of these are not used here, they must nevertheless be present in the file.

The Spherical Wave Expansion coefficients may be specified at several frequencies, sequentially defined in the *Frequency* object referred in the object writing or reading the coefficients file. For each frequency in the sequence, the data structure described below must be concatenated into one file, one set of data for each frequency, and the data must be concatenated according to the frequency sequence. The data structure contains for each frequency a header section followed by the Spherical Wave Expansion coefficients.

The file is read in the following free formatted fashion:

Record Contents	Format
-----------------	--------

1	PRGTAG	(characters)
---	--------	--------------

PRGTAG – Program tag and time stamp, text planted by the program that created the file.

2	IDSTRG	(characters)
---	--------	--------------

IDSTRG – Record with identification text.

3	NTHE, NPHI, NMAX, MMAX	(4 integers)
---	------------------------	--------------

Control data for the Spherical Wave Expansion

NTHE Number of θ -samples over 360° , NTHE must be even and $NTHE \geq 4$ *) .

NPHI Number of ϕ -samples over 360° , $NPHI \geq 3$ *) .

NMAX Maximum value for polar index n in the expansion, $1 \leq NMAX \leq NTHE/2$.

MMAX Maximum value for azimuthal index $|m|$ in the expansion, $0 \leq MMAX \leq \min([(NPHI-1)/2], NMAX)$ with $\min(I1, I2)$ being the smaller of I1 and I2 and [X] being the integer part of X.

*) See 'Note on definition of sample spacing' below.

The following 5 records contain dummy data items of the type specified.

4 TEXT STRING (characters)

5 FIVE REAL DATA ITEMS (5 real numbers)

6 FIVE REAL DATA ITEMS (5 real numbers)

7 TEXT STRING (characters)

8 TEXT STRING (characters)

-- end of header records --

The following records contain the Spherical Wave Expansion coefficients in the order of the azimuthal modes. For each azimuthal mode of indices m and $-m$, the sequence of the Q-coefficients, Q'_{smn} , are preceded by a record containing the modal index $|m|$ for that mode, as well as the power contained in the mode. The scheme is as follows :

m.1 M, POWERM (1 integer, 1 real number)

M	Azimuthal mode index, $ m $
POWERM	Power in all coefficients of index $\pm m$

m.2 Q1MN, Q2MN (4 real numbers on each record)

Q1MN, Q2MN - Wave coefficients Q'_{smn} for $s = 1, 2$ and (($m=-m$ and $m=+m$), $n = \max(1, |m|, \dots, N_{\max})$)

The above scheme is repeated for all azimuthal modes, $|m| = 0, \dots, M_{\max}$.

Explicitly, for $m = 0$ the sequence is

$$\{Q'_{101}, Q'_{201}, Q'_{102}, Q'_{202}, \dots, Q'_{1,0,N_{\max}}, Q'_{2,0,N_{\max}}\} \quad (16)$$

while for $|m| = 1$ we have the sequence

$$\begin{aligned} & \{Q'_{1,-1,1}, Q'_{2,-1,1}, Q'_{111}, Q'_{211}, Q'_{1,-1,2}, Q'_{2,-1,2}, Q'_{112}, Q'_{212}, \dots, \\ & Q'_{1,-1,N_{\max}}, Q'_{2,-1,N_{\max}}, Q'_{1,1,N_{\max}}, Q'_{2,1,N_{\max}}\} \end{aligned} \quad (17)$$

and - in general - for $|m| = \mu$

$$\begin{aligned} & \{Q'_{1,-\mu,\mu}, Q'_{2,-\mu,\mu}, Q'_{1\mu\mu}, Q'_{2\mu\mu}, Q'_{1,-\mu,\mu+1}, Q'_{2,-\mu,\mu+1}, Q'_{1,\mu,\mu+1}, Q'_{2\mu,\mu+1}, \dots, \\ & Q'_{1,-\mu,N_{\max}}, Q'_{2,-\mu,N_{\max}}, Q'_{1,\mu,N_{\max}}, Q'_{2,\mu,N_{\max}}\}. \end{aligned} \quad (18)$$

The number of coefficients for each $|m|$ is $4(N_{\max} - (|m|-1))$, except for $m = 0$, where only $2N_{\max}$ coefficients are present.

The electric field, \vec{E}_{SI} in SI-units, is given by the following expansion in spherical waves

$$\vec{E}_{SI}(r, \theta, \phi) = k\sqrt{\zeta} \sum_{smn} Q_{smn} \vec{F}_{smn}(r, \theta, \phi). \quad (19)$$

For this field, the total radiated power, P , is given by

$$P = \frac{1}{2} \sum_{smn} |Q_{smn}|^2 \text{ (Watts)}. \quad (20)$$

When the total radiated power is 4π Watts then the far-field pattern is normalised to dBi.

The coefficients in the file, Q'_{smn} , are related to the coefficients, Q_{smn} , in the spherical expansion, Eq. (19), by

$$Q'_{smn} = \frac{1}{\sqrt{8\pi}} Q_{smn}^* \quad (21)$$

where the asterisk means complex conjugation (see the note below). This normalisation of the coefficients in the file implies that a field normalised to dBi, i.e. a field radiating a power of 4π watts, will have (cf. Eq. (20))

$$P = \frac{1}{2} \sum_{smn} |Q_{smn}|^2 = 4\pi \text{ (Watts)} \quad (22)$$

and hence (cf. Eq. (3))

$$\sum_{smn} |Q'_{smn}|^2 = 1. \quad (23)$$

The Q'_{smn} -coefficients in the file thus reflect the power of the field and the parameter POWERM is given by

$$\text{POWERM} = \frac{1}{2} \sum_{sn} |Q'_{smn}|^2 \text{ (}|m|\text{ is fixed, } |m| = M\text{)} \quad (24)$$

and the total power P is expressed as

$$P = \sum_{|m|} \text{POWERM} = \frac{1}{2} \sum_{smn} |Q'_{smn}|^2. \quad (25)$$

This applies whether the pattern is power normalised or not. For power normalised patterns radiating 4π Watts we find

$$P = \frac{1}{2}. \quad (26)$$

If POWERM and P of Eqs. (24) and (25) are multiplied by 8π the power is expressed in Watts.

Note on Normalisation:

For historical reasons - and in order to stay compatible with other software - the definition of the coefficients in the file has been retained as the one originally introduced by Larsen in 1980. He used the time dependence $e^{-i\omega t}$, whereas GRASP employs $e^{j\omega t}$ in Eq. (19). Hence the complex conjugation in Eq. (21).

He introduced the excitation, v , of the antenna and the transmission coefficients, T_{smn} , by

$$Q'_{smn} = v T_{smn}. \quad (27)$$

The transmission coefficients are characteristic for the antenna and independent of the antenna excitation. If the antenna is matched and lossless then we have

$$\sum_{smn} |T_{smn}|^2 = 1. \quad (28)$$

Accordingly, for the radiated power, P ,

$$P = \frac{1}{2} \sum_{smn} |Q'_{smn}|^2 = \frac{1}{2} |v|^2 \sum_{smn} |T_{smn}|^2 = \frac{1}{2} |v|^2 \quad (29)$$

and a normalisation corresponding to $v = 1$ was a natural choice leading to Eq. (26).

Note on definition of sample spacing:

It shall be noted that NTHE (record No. 3) is different from the attribute *n_theta* in the object of class *Spherical Wave Expansion (SWE)* generating the file. The two quantities are related through $NTHE = (n_theta - 1) \times 2$. Further, *NPHI* may be odd which is not allowed for the attribute *n_phi*.

Links

[Tabulated Feed Data](#)

[List of File Formats](#)

[Reference Section](#)

F4.4 Hexagonal Wave Guide Modal Aperture Field

Format

This file contains the values of the generating function in the first quadrant of a hexagonal aperture for a number of waveguide modes and is needed if other modes than those built into the program (see *Hexagonal Horn*) are needed. The file may be used by class *Hexagonal Horn* and the file extension is *.dat*. The file is in fixed format.

The aperture field grid data is of the form

Record Contents	Format
1 AX, IDIM	(F8.4, I4)

2.1 ((IJM(I, K, L), I=1, IDIM), K=1, 2), L=1, 4	(16I4)
---	--------

3.1 (NYX(I, J), I=1, IDIM), J=1, 2	(16I4)
------------------------------------	--------

4 INDL(I), I=1, 2	(2I4)
-------------------	-------

The following data records contain data about the modal generating field and are repeated for each mode

5 CHAR, ISF	(A4, A2)
-------------	----------

6 ECM, NSYMX, NSYMY, MD	(F8.5, 3I1)
-------------------------	-------------

7.1 (A(I, J), I=1, IDIM), J=1, IDIM	(9F8.5)
-------------------------------------	---------

Links

[Tabulated Feed Data](#)

[List of File Formats](#)

[Reference Section](#)

F5 Array Element Data

The following array element data file types are available:

- *Irregularly Spaced Feed Data*
- *Feed Data in Regular Grid*
- *Array Element Excitation Coefficients*

Links

[List of File Formats](#)

[Reference Section](#)

F5.1 Irregularly Spaced Feed Data

Format

This file contains the irregular positions and orientations of the elements in an array. The file is used by the class *Tabulated General Array* and the file extension is *.isp*. The element positions are given by (x, y, z) -coordinates relative to the array coordinate system.

Record Contents	Format
1 TITLE	(characters)
TITLE – Record with identification text. This record is repeated until a record containing ++++ as the first 4 characters is reached.	
2 LENGTH_UNIT	(characters)
Length unit of all xyz-positions (see valid units in <i>Applicable Units</i>).	
3 ID, X ,Y, Z, THETA, PHI, PSI, SOURCE, BEAM_ID (string, 6 real numbers, 2 strings)	
ID	Identification string for array element.
X, Y, Z	Position of the element.
THETA, PHI, PSI	Orientation in degrees of the element.
SOURCE	Name of an object of class source used as the array element. The source is moved together with its own coordinate system so that it coincides with the element position and orientation.
BEAM_ID	Identification string selecting a specific beam from the source which is used as the array element: If the source has several beams (it may, for example, be a sub-array with the <i>element_composite</i> attribute set to 'element') then the desired beam must be selected using the appropriate identification string. If the source has only a single beam (it may be a simple feed or a sub-array with the <i>element_composite</i> attribute set to 'composite') then the BEAM_ID should not be specified.

This record is repeated for all elements to end of file.

Links

[Array Element Data](#)

[List of File Formats](#)

[Reference Section](#)

F5.2 Feed Data in Regular Grid

Format

This file contains the positions defined in a hexagonal or rectangular grid and the orientations of the elements in an array. The file is used by the class *Tabulated Planar Grid Array* and the file extension is *.rsp*. The element positions are given by indices in the grid which is defined relative to the array coordinate system.

For the records, blank is separator, comma is optional.

Record Contents	Format
-----------------	--------

1 TITLE	(characters)
--------------	--------------

TITLE – Record with identification string.

This record is repeated until a record containing ++++ as the first 4 characters is reached.

2 ID, MX, MY, THETA, PHI, PSI, SOURCE, BEAM_ID	(string, 2 integers, 3 real numbers, 2 strings)
---	---

ID	Identification string for array element. Each element in the array must be given an identification which is unique among the elements of this array.
MX, MY	Node indices of the element in the grid.
THETA, PHI, PSI	Orientation in degrees of the element.
SOURCE	Name of an object of class source used as the array element. The source is moved together with its own coordinate system so that it coincides with the element position and orientation.
BEAM_ID	Identification string selecting a specific beam from the source which is used as the array element: If the source has several beams (it may, for example, be a sub-array with the <i>element_composite</i> attribute set to 'element') then the desired beam must be selected using the appropriate identification string. If the source has only a single beam (it may be a simple feed or a sub-array with the <i>element_composite</i> attribute set to 'composite') then the BEAM_ID shall not be specified.

This record is repeated for all elements to end of file.

Links

[Array Element Data](#)

[List of File Formats](#)

[Reference Section](#)

F5.3 Array Element Excitation Coefficients

Format

This file contains the excitation coefficients of the elements in an array. The file is used by the classes *Tabulated General Array* and *Tabulated Planar Grid Array* and the file extension is *.exi*. The file does not need to contain excitation coefficients for all the elements, but only for those for which the coefficients are different from zero.

Record Contents	Format
-----------------	--------

1 TITLE	(characters)
---------	--------------

TITLE – Record with identification text.

This record is repeated until a record containing ++++ as the first 4 characters is reached.

2 ID, AMP, PHASE	(string, 2 real numbers)
------------------	--------------------------

ID	Identification string for array element.
----	--

AMP, PHASE	Amplitude in dB and phase in degrees of the excitation coefficient for the array element named 'ID'.
------------	--

This record is repeated for all elements to end of file.

Links

[Array Element Data](#)

[List of File Formats](#)

[Reference Section](#)

F6 Electrical Properties for Scatterers

One file type for electrical properties for scatterers is available:

- *Tabulated Electrical Properties for Scatterers*

Links

List of File Formats

Reference Section

F6.1 Tabulated Electrical Properties for Scatterers

Format

The electrical properties of a scatterer surface can be specified by tabulated reflection and transmission coefficients. Tabulated data are assumed to be generated in a coordinate system defined by x_c and y_c vectors and a possible rotation α , see a detailed description of the parameters in class *Tabulated Electrical Properties*. The surface material parameters are projected onto the reflector from the direction $z_c = x_c \times y_c$. The file extension is *.tep*.

If tabulated data are used for several frequencies the total data set consisting of the records 2, 3 and 4 as described below must be appended in the file with the same sequence as the connected frequency object.

The format is described in the following. A sample file may be generated by defining a reflector with suitable *Electrical Properties* and next store these properties in the present format by means of class *Electrical Properties Data Output*.

Record Contents	Format
1 FILEV	(18 characters)
FILEV - Identification of file type and version. 'TICRA-EL_PROP-V1.0'	
2 TITLE	(characters)
TITLE - Record with identification text.	
3 NTH, MP, THMAX	(2 integers, 1 real number)
NTH	no. of samples from an incidence angle of 0° to THMAX degrees, $NTH \geq 4$.
MP	no. of equidistant cuts in ϕ_i (≥ 1). ϕ_i is the azimuthal angle from the x_c -axis.
THMAX	maximum angle of θ_i incidence for which data are available. THMAX must be less than or equal to 90° .
4 (((CE(I,J,K), K=1,16), I=1, NTH), J=1, MP)	(4 real numbers on each record)

CE(I,J,1-4)	:	Complex reflection coefficients for front incidence
CE(I,J,5-8)	:	Complex transmission coefficients for front incidence
CE(I,J,9-12)	:	Complex reflection coefficients for rear incidence
CE(I,J,13-16)	:	Complex transmission coefficients for rear incidence

---end of file---

All above is for angles

$$\theta_i = (I - 1)\text{THMAX}/(\text{NTH} - 1), \quad I = 1, \dots, \text{NTH} \quad (30)$$

$$\phi_i = (J - 1)360^\circ/\text{MP}, \quad J = 1, \dots, \text{MP} \quad (31)$$

The reflection and transmission coefficients are ordered as follows

$$R_{\theta\theta}, R_{\theta\phi}, R_{\phi\theta}, R_{\phi\phi} \quad (32)$$

$$T_{\theta\theta}, T_{\theta\phi}, T_{\phi\theta}, T_{\phi\phi} \quad (33)$$

in the array CE described above with two coefficients per record, eight records for each angular set (θ_i, ϕ_i) .

All parameters must refer to the positive side of the reflector, the side with the positive normal. For the definition of coordinates we refer to the two drawings in Figure 1 and Figure 2, where the first is for incidence from the front side of the reflector (positive normal) and the second is for rear incidence. It is important to have coefficients for both front and rear incidence, since both cases will generally occur. If, for example, we consider a normal polarisation-sensitive reflector system composed of a gridded front reflector and a solid rear reflector, the field of the feed will be transmitted through and reflected in the front reflector, which is then illuminated from the front side. The transmitted field will next reflect in the solid rear reflector and then illuminate the gridded front reflector from the rear side causing a second transmission through this.

First, an incident plane wave that illuminates the front side of the material is considered, Figure 1. The direction of the incident plane wave is given by the polar angles (θ_i, ϕ_i) , where θ_i is measured from the positive z -axis and ϕ_i from the positive x -axis. The xyz -coordinate system is a local coordinate system on the surface where the z -axis is the positive normal of the surface. The angle θ_i has the range $0 \leq \theta_i < 90^\circ$ because it is assumed that the incident field hits the front side. The incident plane wave can be decomposed as

$$\bar{E}^i = E_\theta^i \hat{\theta}_i + E_\phi^i \hat{\phi}_i \quad (34)$$

and is partly reflected and partly transmitted through the surface where the unit vectors of incidence, $\hat{\theta}_i$ and $\hat{\phi}_i$ are the usual polar vectors shown on Figure F6.1-1 and defined explicitly by

$$\begin{aligned} \hat{\theta}_i &= \hat{x} \cos \theta_i \cos \phi_i + \hat{y} \cos \theta_i \sin \phi_i - \hat{z} \sin \theta_i \\ \hat{\phi}_i &= -\hat{x} \sin \phi_i + \hat{y} \cos \phi_i \end{aligned} \quad (35)$$

The reflected field is given by

$$\bar{E}^r = E_\theta^r \hat{\theta}_r + E_\phi^r \hat{\phi}_r \quad (36)$$

where

$$\begin{pmatrix} E_\theta^r \\ E_\phi^r \end{pmatrix} = \begin{pmatrix} R_{\theta\theta} & R_{\theta\phi} \\ R_{\phi\theta} & R_{\phi\phi} \end{pmatrix} \begin{pmatrix} E_\theta^i \\ E_\phi^i \end{pmatrix} \quad (37)$$

and the unit vectors of reflection, $\hat{\theta}_r$ and $\hat{\phi}_r$ are the negative mirror images of $\hat{\theta}_i$ and $\hat{\phi}_i$, respectively, so that

$$\begin{aligned} \hat{\theta}_r &= 2(\hat{\theta}_i \cdot \hat{z})\hat{z} - \hat{\theta}_i \\ \hat{\phi}_r &= -\hat{\phi}_i \end{aligned} \quad (38)$$

Similarly, the transmitted field is given by

$$\bar{E}^t = E_\theta^t \hat{\theta}_t + E_\phi^t \hat{\phi}_t \quad (39)$$

where

$$\begin{pmatrix} E_\theta^t \\ E_\phi^t \end{pmatrix} = \begin{pmatrix} T_{\theta\theta} & T_{\theta\phi} \\ T_{\phi\theta} & T_{\phi\phi} \end{pmatrix} \begin{pmatrix} E_\theta^i \\ E_\phi^i \end{pmatrix} \quad (40)$$

and the unit vectors of transmission $\hat{\theta}_t$ and $\hat{\phi}_t$ are defined by

$$\hat{\theta}_t = \hat{\theta}_i, \quad \hat{\phi}_t = \hat{\phi}_i \quad (41)$$

When the coefficients associated with rear incidence are considered, the angle θ_i of incidence shall be measured from the negative z -axis to the incident ray so that we still have $0 \leq \theta_i < 90^\circ$. The ϕ_i angle is as before measured from the positive x -axis towards the positive y -axis, see Figure 2. The unit vectors $\hat{\theta}_i$ and $\hat{\phi}_i$ of incidence are now defined by

$$\begin{aligned} \hat{\theta}_i &= -\hat{x} \cos \theta_i \cos \phi_i - \hat{y} \cos \theta_i \sin \phi_i - \hat{z} \sin \theta_i \\ \hat{\phi}_i &= -\hat{x} \sin \phi_i + \hat{y} \cos \phi_i \end{aligned} \quad (42)$$

The relations (36) through (41) are also valid when the incident field illuminates the backside.

It should be noted that all coefficients must refer to a reference plane, which is chosen to be the positive side of the surface. Some software packages for prediction of reflection and transmission coefficients may choose other reference planes by default, for instance the rear side for the transmission coefficients. In that case, the transmission coefficients must be multiplied by a correction factor which is

$$e^{jk\delta \cos \theta_i} \quad (43)$$

where δ is the thickness of the material and θ_i is the angle of incidence.

Links

[Electrical Properties for Scatterers](#)

[List of File Formats](#)

[Reference Section](#)

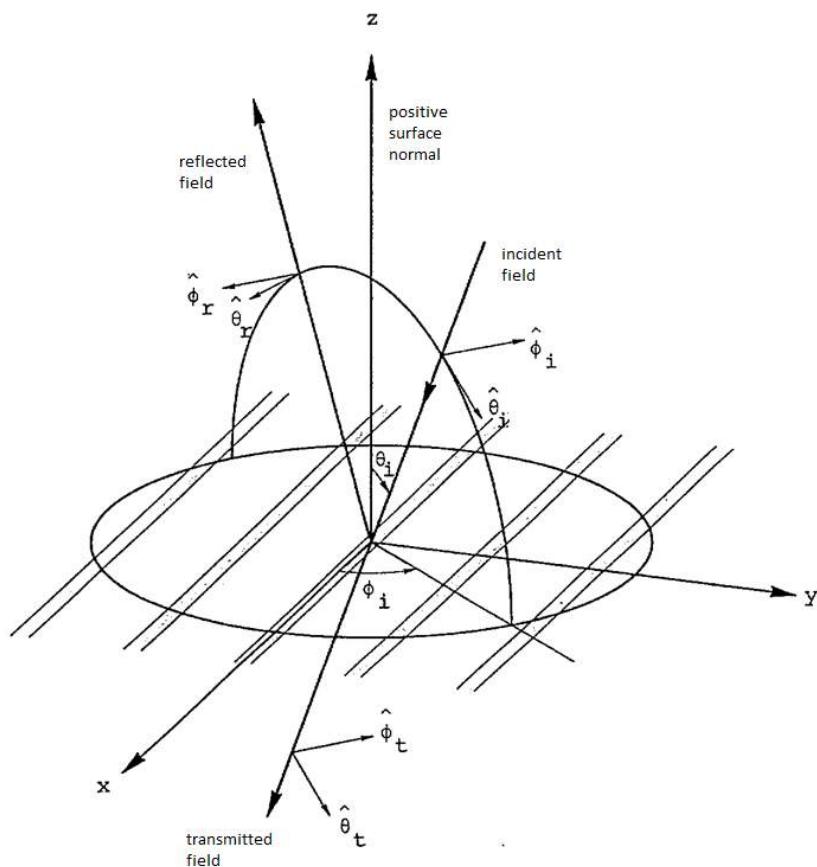


Figure 1 Incidence from front side.

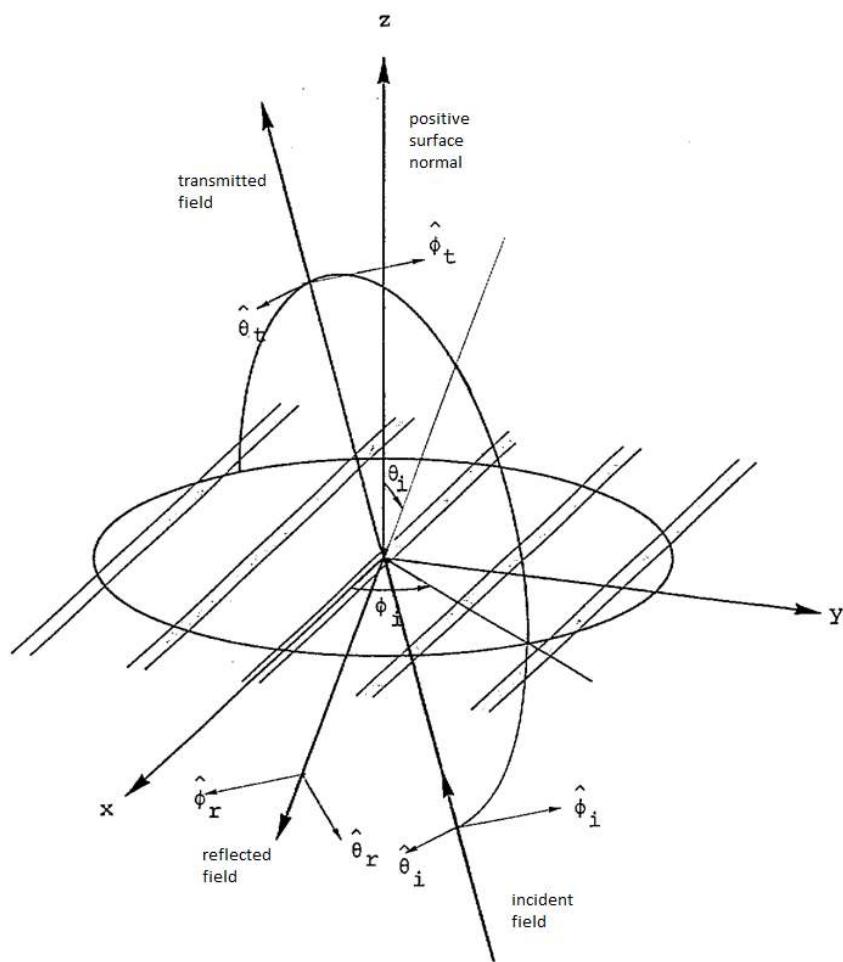


Figure 2 Incidence from backside.

F7 Three-Dimensional Plotter Data

One file type for storing 3D plotter data of an antenna system is available:

- *Three-Dimensional Plotter Data as XYZ-Values*

Links

[List of File Formats](#)

[Reference Section](#)

F7.1 Three-Dimensional Plotter Data as XYZ-Values

Format

This file contains the x , y and z values of 3D line points. The start of a line is specified by the integer value IPEN=3 and the line is connected to points with IPEN=2. The format of this file is fixed and the file extension is .xyz. The file format is obsolete and maintained for compatibility with older GRASP-versions.

The format is:

Record Contents	Format
1 FILEV	(19 characters)
FILEV - Identification of file type and version. 'TICRA-XYZ_PLOT-V2.0'	
2 X, Y, Z, R, G, B, IPEN, CONTROL	(6E18.10,2I3)
X, Y, Z - x , y , z values of 3D point	
R, G, B - Weights of red, green and blue in the RGB colour model	
IPEN - Control of line points (Pen) =3 - start point (Pen up or move to) =2 - connected point (Pen down or draw to)	
CONTROL - Integer value =-1 - record 2.1 is read = 0 - record 2.1 is omitted	
If CONTROL = -1 record 2.1 is read	
2.1 OBJ_NAME	(characters)
OBJ_NAME - Name of the 3D object for which plot data starts in the line above	
Record 2 (and record 2.1 when CONTROL=-1 in record 2) are continued to end of file	
	---end of file---

Links

[Three-Dimensional Plotter Data](#)

[List of File Formats](#)

[Reference Section](#)

F8 Definition of Coordinate Systems

One file type for storing definition of tabulated coordinate systems is available:

- *Definition of Tabulated Coordinate System*

Links

List of File Formats

Reference Section

F8.1 Definition of Tabulated Coordinate System

Format

Files of this type are used for defining a coordinate system by the class *Tabulated Coordinate System* and the file extension is .cor.

The coordinate system shall be specified in a file of the following format:

Record Contents	Format
1 TITLE	(characters)
TITLE – Record with identification string.	
2 LENGTH_UNIT	(characters)
LENGTH_UNIT Unit of the numbers defining the origin of the coordinate system (see valid units in <i>Applicable Units</i>).	
3 (ORIGIN(I), I=1,3)	(3 real numbers)
ORIGIN(1-3) : x-, y- and z-coordinates in the reference coordinate system of the origin of the new coordinate system.	
4 (X_AXIS(I), I=1,3)	(3 real numbers)
X_AXIS(1-3) : Vector in the reference coordinate system in the direction of the x-axis of the new coordinate system.	
5 (Y_AXIS(I), I=1,3)	(3 real numbers)
Y_AXIS(1-3) : Vector in the reference coordinate system in the direction of the y-axis of the new coordinate system.	
---end of file---	

Links

[Definition of Coordinate Systems](#)

[List of File Formats](#)

[Reference Section](#)

F9 Coupling Data

The following file formats for coupling values are available:

- *Coupling Data in Cuts*
- *Coupling Data in Grids*

Links

List of File Formats

Reference Section

F9.1 Coupling Data in Cuts

Format

The file is generated by objects of the classes *Coupling System* and *Coupling (Obsolete)* (obsolete) for storing coupling quotients in cuts. The file format is the same as generated by objects of the classes *Spherical Cut*, *Planar Cut*, *Surface Cut* and *Cylindrical Cut* for storing field data in cuts. The file extension is *.cut*.

A cut consists of records of types 1, 2 and 3 as described below. If more than one cut is contained in a file all records must be repeated for each cut.

Record Contents	Format
-----------------	--------

1	TEXT	(characters)
---	------	--------------

TEXT – Line with identification text

2	V_INI, V_INC, V_NUM, C, ICOMP, ICUT, NCOMP numbers, 1 integer, 1 real number, 3 integers)	(2 real
---	--	---------

V_INI	– Initial value of the movement
V_INC	– Increment of the movement
V_NUM	– Number of movement values in the cut
C	– Dummy value
ICOMP	– Polarisation, not relevant. Fixed value = 3
ICUT	– Control parameter of cut. Fixed value = 1
NCOMP	– Number of field components. Fixed value = 2

Record No. 3 and the following records contain the coupling quotients. For backward compatibility, the coupling quotient is printed twice.

3	(CQ(I), CQ(I), I = 1, V_NUM)	(4 real numbers in each record)
---	------------------------------	---------------------------------

CQ	– Complex array containing the coupling quotients for the I'th data point at movement position V: $V = V_{INI} + V_{INC} * (I-1)$
----	--

---end of data file---

Links

[Coupling Data](#)

[List of File Formats](#)

[Reference Section](#)

F9.2 Coupling Data in Grids

Format

The file is generated by objects of the classes *Coupling System* and *Coupling (Obsolete)* (obsolete) for storing coupling quotients in grids. The file format is the same as generated by objects of the classes *Spherical Grid*, *Planar Grid*, *Surface Grid* and *Cylindrical Grid* for storing field data in grids. The file extension is *.grd*.

If two or more movements are specified in the coupling determination then the two fastest movements may constitute a grid. Additionally specified (slower) movements will cause output in several grids.

The file format is:

Record Contents	Format
1 TEXT	(characters)
TEXT – Record with identification text. This record is repeated until a record containing ++++ as the first 4 characters is reached.	
2 KTYPE	(integer)
Specifies type of file format KTYPE = 1 – standard format for 2D grid. For files used in GRASP this variable is always 1.	
3 NSET, ICOMP, NCOMP, IGRID	(4 integers)
NSET – Number of grids in the file. ICOMP – Polarisation, not relevant. Fixed value = 3 NCOMP – Number of field components. Fixed value = 2 IGRID – Grid type, not relevant. Fixed value = 7	
4 (IX(I), IY(I), I=1, NSET)	(2 integers on each record)
IX, IY – Centre of set, not relevant. Fixed values are 0, 0	
All the following records are repeated NSET times	
5 XS, YS, XE, YE	(4 real numbers)
Limits of 2D grid. The grid points (X, Y) run through the movement values	

$$\begin{aligned} X &= XS + DX * (I-1) \\ Y &= YS + DY * (J-1) \end{aligned}$$

where

X is in the last (fastest) movement, and

Y is in the second to the last movement, further

$$\begin{aligned} DX &= (XE-XS) / (NX-1) \\ DY &= (YE-YS) / (NY-1) \end{aligned}$$

The number of grid values NX and NY and the range of the index I and J are defined in the following records.

6 NX, NY, KLIMIT (3 integers)

NX	-	Number of columns
NY	-	Number of rows
KLIMIT	-	not relevant, Fixed value = 0

The coupling quotients are printed according to record 7 which is repeated NY times (J=1, NY).

For backward compatibility, each coupling quotient is printed twice.

7 (CQ(I,J), CQ(I,J), I = 1, NX) (4 real numbers in each record)

CQ(I,J) - Complex coupling value for movement to position/orientation X(I), Y(J)

---end of data file---

Links

[Coupling Data](#)

[List of File Formats](#)

[Reference Section](#)

F10 General Geometries

The only format for a general geometry of a scatterer is

- *Meshed Geometries*

Links

List of File Formats

Reference Section

F10.1 Meshed Geometries

Format

This format is applied for tabulated meshed geometries. The format is used for input files defining *Tabulated Mesh* scatterers, and for output files of generated meshed MoM geometries (by the *Write MoM Mesh* command). The recommended file extension is *.msh*.

The file specifies a number of homogeneous dielectric regions, a number of nodes, and a number of surface patches and wire segments defined by these nodes. The order of the various entries in the mesh file is not important, except that the dielectric regions and the nodes must be defined before they are referred to in the definition of a patch or a wire segment. It is therefore recommended to define the dielectric regions first, then the nodes, and finally the patches and the wires. To see an example of a valid mesh file, use the command *Write MoM Mesh* in GRASP.

For scatterers described by a *Tabulated Mesh* it must be noted that the patches are modelling the surfaces of the scatterer and need not to depict a detailed mesh needed for the *MoM* calculations (the meshing is carried out by the actual MoM object). The number of nodes and patches needed for describing the surface is thus determined by the curvature of the surface. The same applies for wires. A long straight wire may be described by two nodes (the end nodes) while a curved wire may need to be described by several wire segments of up to 4 nodes. Thus, a helix may be described by four wire segments per full turn where each segment is described by 4 nodes.

The content of the file must be as follows. Text shown in brackets must be written as shown, including the brackets:

Record Contents	Format
1 [VERSION]	(9 characters)
2 FILE_VERSION	(characters)
	Identification of file type and version, presently only the following text is accepted: TICRA Mesh File, Version 1.0
3 [TITLE]	(7 characters)
4 TITLE	(characters)
	One record with arbitrary identification text.
5a [REGION], REGION_NUMBER	(8 characters, 1 integer)

5b PERMITTIVITY, PERMEABILITY, LOSS_TANGENT (3 reals)

One or more sets of two records defining the regions.

REGION_NUMBER	A positive number uniquely identifying the region.
PERMITTIVITY	The relative permittivity of the region.
PERMEABILITY	The relative permeability of the region.
LOSS_TANGENT	The loss_tangent of the region.

6a [NODE], NODE_NUMBER (6 characters, 1 integer)

6b X, Y, Z (3 reals)

One or more sets of two records defining the nodes.

NODE_NUMBER	A positive number uniquely identifying the node.
X	<i>x</i> -coordinate of the node (in meter).
Y	<i>y</i> -coordinate of the node (in meter).
Z	<i>z</i> -coordinate of the node (in meter).

7a [PATCH], PATCH_NUMBER (7 characters, 1 integer)

7b N_NODES, REGION1, REGION2, ZS (3 integers, 2 reals)

7c (NODE(I), I=1, N_NODES) (integers)

One or more sets of three records (7a - 7c) defining the patches.

PATCH_NUMBER	A positive number uniquely identifying the patch.
N_NODES	The number of nodes required for the patch definition, may take one of the values 4, 9, 16, 25, 3 or 7; see the remarks below.
REGION1, REGION2	The numbers of the two regions on the two sides of the patch (optionally). If REGION1 and REGION2 are not given, the patch is assumed to be located in region 0 (the free space) and to be perfectly conducting. If REGION1 = REGION2 then the patch is perfectly conducting if ZS is not specified.
ZS	The complex surface impedance of the patch, in ohm/meter (optionally, requires that REGION1 = REGION2). ZS=0 specifies a perfect conductor.
NODE(I)	Number of the I'th node for specifying the patch. The nodes shall be given in the order as stated in the remarks below.
8a [WIRE], SEGMENT_NUMBER	(6 characters, 1 integer)
8b N_NODES, REGION, ZS	(2 integers, 2 reals)
8c (NODE(I), I=1, N_NODES), RS, RE	(N_NODES integers, 2 reals)

One or more sets of three records (8a - 8c) defining the wire segments.

SEGMENT_NUMBER	A positive number uniquely identifying the wire segment.
N_NODES	The number of nodes required for the definition of the wire segment, may take one of the values 2, 3 or 4; see the remarks below.
REGION	The number of the region in which the wire segment is located. A wire segment can not lay in, or pass, the boundary between two regions. If REGION is not given, the wire segment is assumed to be located in region 0 (the free space) and to be perfectly conducting.
ZS	The complex surface impedance of the wire segment, in ohm/meter (optional). ZS=0 specifies a perfect conductor.
NODE(I)	Number of the I'th node for specifying the wire segment.
RS	The wire radius at the first node in the wire segment (in meter).
RE	The wire radius at the last node in the wire segment (in meter).

Remarks and examples

Example on definition of a dielectric property The following two lines define region 2 as a lossless dielectric region with $\varepsilon_r = 4.0$ and $\mu_r = 1.0$:

```
[region] 2
4.0 1.0 0.0
```

A lossy medium is obtained by specifying a non-zero value for the loss tangent.

Example on definition of a node The following example defines the two nodes 340 and 341 at $(x, y, z) = (0, 0, 0)$ and $(x, y, z) = (0, 0, 1)$:

```
[node] 340
0.0 0.0 0.0
[node] 341
0.0 0.0 1.0
```

On definition of a patch The possible choices for the number of nodes for defining a patch depend on the type of the patch, see also the following figure:

Type of surface patch	Number of nodes in the definition
Bilinear quadrilateral	4
Biquadratic quadrilateral	9
Bicubic quadrilateral	16
4th order quadrilateral	25
Flat triangle	3
Curved triangle	7

The list of nodes has to be given in the sequence as indicated in Figure 1.

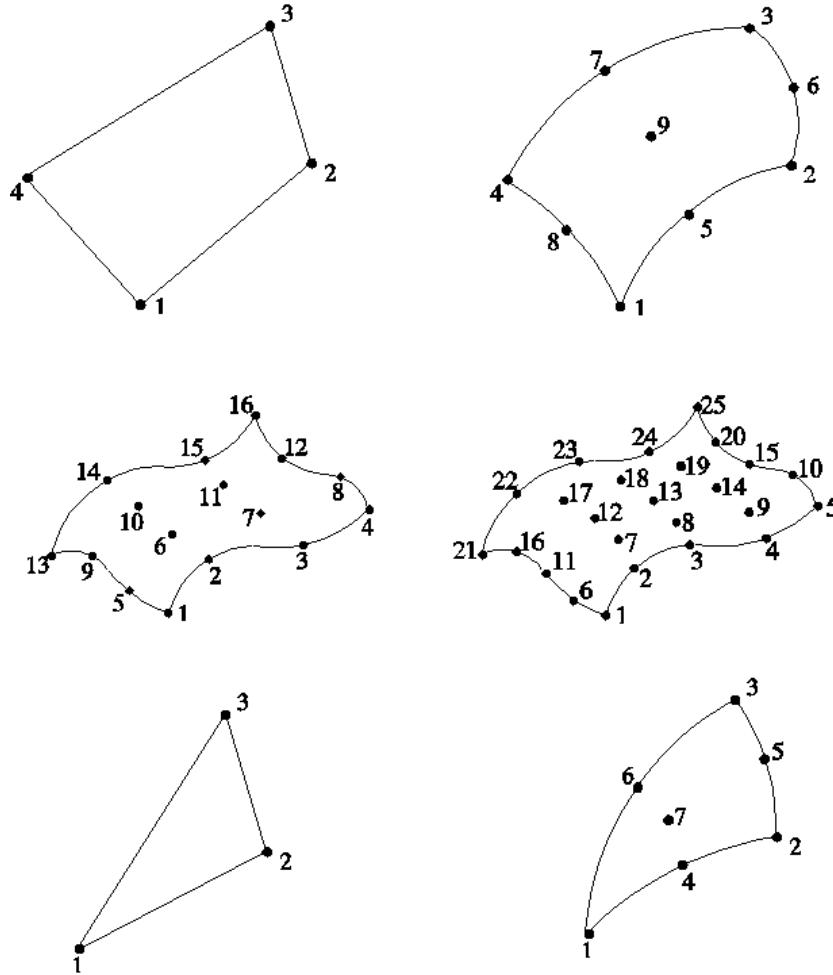


Figure 1

Numbering schemes for patches defined by 4, 9, 16, 25, 3, and 7 nodes. It is recommended to avoid triangles whenever possible, i.e., patches with three or seven nodes should only be used if a pure quadrilateral mesh is not possible. The direction of the normal is out of the paper for all the shown patches. The direction has no influence on the MoM computations.

If the impedance, Z_s , is also specified, the patch is conducting and has the specified surface impedance defined as

$$Z_s = \frac{1+j}{\sigma\delta_s} \quad (44)$$

in which σ is the conductivity and δ_s is the skin depth. Currently, ZS can only be non-zero if REGION1=REGION2.

Examples on definition of a patch

As an example, a nine-node patch is defined by

```
[patch] 1
9
340 341 623 90 909 9 87 23 222
```

This patch is assumed to be perfectly conducting and located in region 0. All the nodes referred to in the third line must be defined before the definition of the patch.

To define a four-node patch that separates previously defined dielectric regions 2 and 3, use

```
[patch] 1
4 2 3
340 341 623 90
```

Such a patch will support both electric and magnetic currents.

To define the same quadrilateral patch as above, but this time as perfectly conducting, specify ZS to zero:

```
[patch] 1
4 2 3 0.0 0.0
340 341 623 90
```

This quadrilateral patch will have two independent electric current sheets defined on it, one sheet of currents radiating into region 2 and another set of currents radiating into region 3.

If two identical region numbers are specified, i.e.

```
[patch] 1
9 1 1
340 341 623 90 909 9 87 23 222
```

the patch is assumed to be perfectly conducting and will only support a single sheet of electric currents. Thus, an equivalent definition is

```
[patch] 1
```

9 1 1 0.0 0.0
340 341 623 90 909 9 87 23 222

Finally, the following figure shows a scatterer consisting of two square *Rectangular Plates* with side lengths of 1 wavelength combined to one scatterer by *Scatterer Cluster*. The plates are applied as scatterer in a *MoM* object and the resulting meshing is plotted by *MoM Plot*. The mesh is printed to a file by *Write MoM Mesh* and a part of the file is also shown.

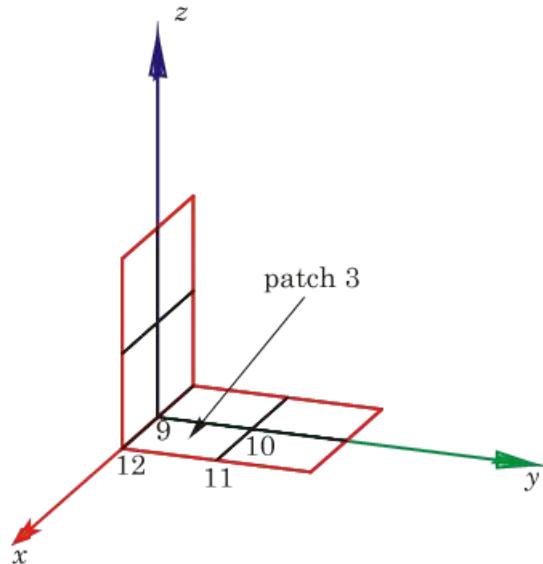


Figure 2

A scatterer defined by two adjoined square plates with side lengths of 1 wavelength. The scatterer is meshed (black lines) by *MoM*. Named are nodes 9 through 12 and a patch, patch 3, defined by these nodes.

[node] 9
0.00000000E+00 0.10000000E+01 0.00000000E+00
[node] 10
0.00000000E+00 0.15000000E+01 0.00000000E+00
[node] 11
0.50000000E+00 0.15000000E+01 0.00000000E+00
[node] 12
0.50000000E+00 0.10000000E+01 0.00000000E+00
[patch] 3
4 0 0 0.00000000E+00 0.00000000E+00
9 10 11 12

Figure 3

Part of the listing of the mesh in the figure above. The listing defines nodes 9 through 12 as well as patch number 3 in terms of these nodes.

On definition of a wire segment

The number of nodes for defining a wire segment may be 2, 3 or 4, see also the following figure. When the number of nodes is 2 the wire will be described by a linear function (it will be a straight wire segment), when the number of nodes is 3 or 4 the wire segment will be described by a quadratic or cubic function, respectively.

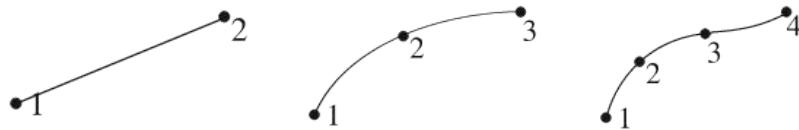


Figure 4

Numbering schemes for wire segments defined by 2, 3 and 4 nodes.

If the impedance, ZS, is also specified, the patch is conducting and has the specified surface impedance defined as

$$Z_s = \frac{1+j}{\sigma\delta_s} \quad (45)$$

in which σ is the conductivity and δ_s is the skin depth.

The radius of the wire segment varies linearly along the wire from RS (the radius at the start point) to RE (the radius at the end point).

Example on definition of a wire segment

As an example, a three-node wire segment is defined by

```
[wire] 1
3 1
9 11 10 0.001 0.002
```

This wire segment is assumed to be perfectly conducting and located in region 1. The wire segment passes the 3 nodes (nodes 9, 11 and 10) in the specified order and has the radius 0.001 meter at the first node (node 9) linearly varying along the wire segment to 0.002 meter at the last node (node 10).

The nodes must be defined before the definition of the wire segment.

Links

[Meshed Geometries](#)

[List of File Formats](#)

[Reference Section](#)

F11 Ray Traces

GTD ray traces from a source to field points may be stored in a file with the extension *.trc*. The content of the file is described in the manual for the Multi GTD add-on.

Links

[List of File Formats](#)

[Reference Section](#)

F12 S, Y, and Z-Parameters

Format

The file is generated by objects of class *Voltage Generator* for storing S-, Y-, or Z-parameters as specified in the object. The file extension is *.par*.

The first record in the file is a header which describes the content of the following N_FREQ records, where N_FREQ is the number of frequencies for which the parameters are determined. Each of these records starts with the value of the frequency followed by all parameters for that frequency. The parameters are listed in the order $S_{11}, S_{12}, \dots, S_{21}, S_{22}, \dots$ (with the S-parameters as example). The order of the parameters is detailed in the header record.

The number of parameters for each frequency is N^2 , where N is the number of generators defined in the *Voltage Generator* object. The parameters are complex numbers, thus each record contains $1 + 2N^2$ real numbers: the frequency (in GHz) followed by $2N^2$ numbers.

For the Y- and Z-matrix, the $2N^2$ numbers correspond to the real and imaginary parts of each matrix element; and for the S-matrix, the $2N^2$ numbers correspond to the amplitude (in dB) and phase (in degrees) of each matrix element. The values are given element by element.

The content of the file can thus be specified as follows:

Record Contents	Format
1 HEADER	(characters)
A header record that describes the contents of the following records. For the S-matrix the header starts with <code>#Freq. [GHz] 20*log10 S_11 Phase(S_11) , ...</code>	
for the Y-matrix the header starts with <code>#Freq. [GHz] REAL(Y_11) IMAG(Y_11) , ...</code>	
and for the Z-matrix the header starts with <code>#Freq. [GHz] REAL(Z_11) IMAG(Z_11) , ...</code>	
2 FREQ, S11, S12, ...	(1+2*N*N real numbers)
FREQ	The actual frequency.
S11, S12, ...	The S- (or Y- or Z-) matrix elements in sequence.

Links

[List of File Formats](#)

Reference Section

F13 Gauss-Laguerre Beam Coefficients

Format

Files of this type are used for storage of Gauss-Laguerre beam coefficients for a lens in binary format. The files are generated by objects of class *PO*, *Lens* and the file extension is *.gbc*.

Links

[List of File Formats](#)

[Reference Section](#)

Appendices

- *Ludwig's 3rd Definition of Polarization*
- *Near-Field Calculations*
- *Phase of a Radiating Horn*

Links

Alphabetical List of Classes and Command Types

Classes

Command Types

Applicable Units

File Extensions

File Formats

Appendices

Reference Section

Contents

Ludwig's 3rd Definition of Polarization

A.C. Ludwig has introduced a definition of co- and cross-polarization⁴ which is uniform on the forward hemisphere of a radiation pattern. The definition is therefore widely used for directive antennas as it gives a well defined description of the polarization of the antenna pattern from the main-beam direction to far out in the side lobes. The polarization definition in the backward direction has, however, an unexpected behavior as will be illustrated here. This has importance when, for example, the backward radiation of the feed in a reflector antenna is considered, a radiation which adds to the forward directed main beam of the antenna.

We will in the following consider a directive antenna radiating in the direction of the positive z-axis, Figure 1.

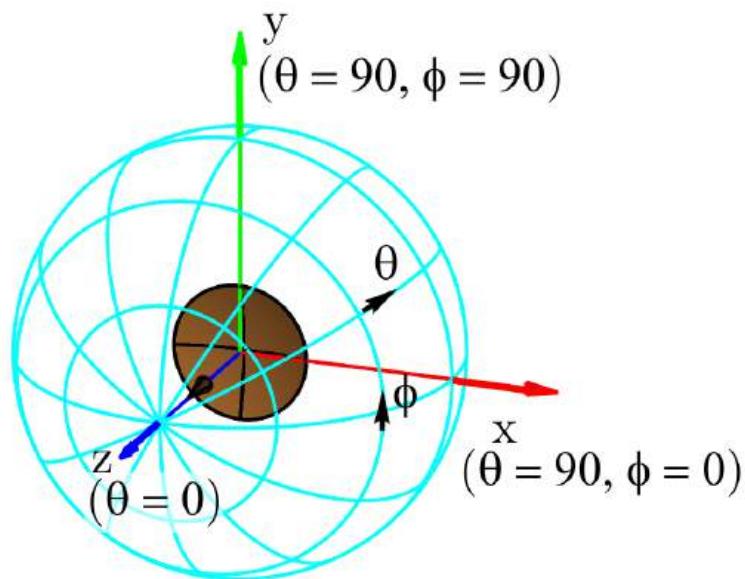


Figure 1 A directive antenna - illustrated by a reflector antenna - radiating in the direction $\theta = 0^\circ$. A spherical $\theta\phi$ -grid surrounds the antenna.

The TICRA software applies Ludwig's 3rd polarization definition for the co-and cross-polar unit vectors \hat{e}_{co} and \hat{e}_{cx} in the following form⁵:

$$\hat{e}_{co} = \hat{\theta} \cos \phi - \hat{\phi} \sin \phi$$

$$\hat{e}_{cx} = \hat{\theta} \sin \phi + \hat{\phi} \cos \phi$$

where $\hat{\theta}$ and $\hat{\phi}$ are the unit vectors in a spherical coordinate system, cf. Figure 1.

The direction of the co-polar unit vector \hat{e}_{co} is illustrated in Figure 2 on a sphere surrounding the antenna. Figure 3 shows in a similar way the directions of the

⁴A.C.Ludwig, "The definition of cross polarisation", IEEE Trans. Antennas Propagat., Vol. AP-21, pp. 116-119, 1973.

⁵By this definition $\hat{e}_{co} \times \hat{e}_{cx}$ becomes parallel to the outgoing radial vector \hat{r} . The original definition by Ludwig has the co- and cross-polar vectors parallel to \hat{e}_{cx} and \hat{e}_{co} , respectively, of the present definition.

cross-polar \hat{e}_{cx} -vector. It is seen how \hat{e}_{co} , Figure 2(a), in a wide region around the direction of the positive z-axis is more or less parallel to \hat{x} and, in Figure 3(a), \hat{e}_{cx} is parallel to \hat{y} . It is this feature which makes the polarization definition so useful.

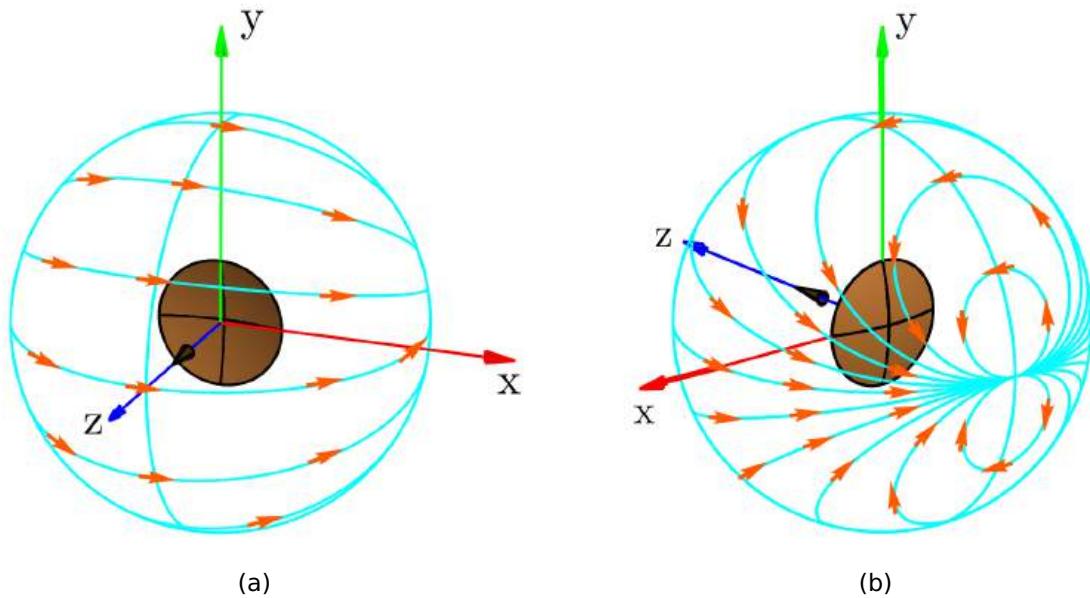


Figure 2 The orientation of \hat{e}_{co} over the front (a) and back (b) hemisphere.

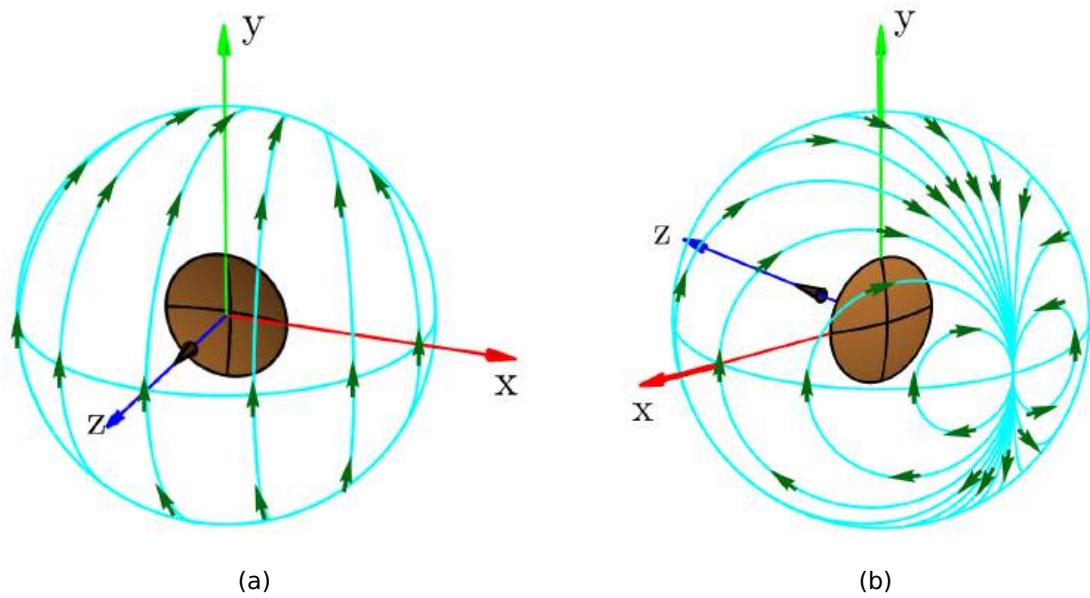


Figure 3 The orientation of \hat{e}_{cx} over the front (a) and back (b) hemisphere.

Any polarization definition has a pole. The well-known $\hat{\theta}\hat{\phi}$ polarization has a pole for $\theta = 0^\circ$ as well as for $\theta = 180^\circ$, cf. Figure 1. Ludwig's 3rd definition has only one pole, which is in the back hemisphere at $\theta = 180^\circ$, see Figure 2(b) for \hat{e}_{co} and

Figure 3(b) for \hat{e}_{cx} . At the pole, the polarization is undefined which can be seen by following the polarization vector \hat{e}_{co} in the zx-plane from the z-axis over the x-axis up to $\theta = 180^\circ$, Figure 2. In GRASP this is a spherical cut for $\phi = 0^\circ$. The vector is always pointing in direction of increasing θ , and approaching $\theta = 180^\circ$ the vector will approach the direction of $-\hat{x}$. But if we instead follow \hat{e}_{co} in the zy-plane, a spherical cut for $\phi = 90^\circ$, it is $+\hat{x}$ directed all the way from $\theta = 0^\circ$ up to θ close to 180° , the pole. The polarization is thus undefined at the pole.

This is reflected in GRASP when the field at the pole is requested. The method can only be applied for a spherical cut, for which there will always be a ϕ -value associated at the pole as well. GRASP select the direction of the reference vector that corresponds to approaching the pole in that particular spherical cut. This results in a continuous field in each cut, but different field values in different cuts.

The polarization definition near the pole implies - which often is a surprise - that the polarization vectors rotates twice (4π) when we move once around the pole. Consider Figure 4 which shows the orientation of the co-polar vector \hat{e}_{co} on the back hemisphere as in Figure 2(b). Following a small circle around the pole (as the purple coloured circle for $\theta = 165^\circ$ in Figure 4) we see how the direction of the co-polar vector \hat{e}_{co} in the left side ($\phi = 0^\circ$) points to the right and, for increasing ϕ rotates clockwise such that it in the zy-plane ($\phi = 90^\circ$) points to the left, parallel to the positive x-axis. For further increasing ϕ -values \hat{e}_{co} rotates such that the \hat{e}_{co} vector has rotated twice, 4π , after a full turn around the pole.

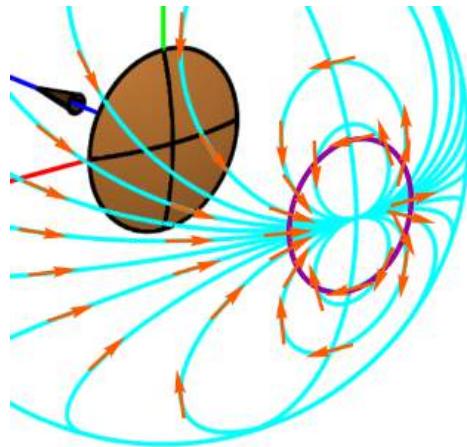


Figure 4

The co-polar polarization vector \hat{e}_{co} rotates two full turns around the pole at $\theta = 180^\circ$ when ϕ increases from 0° to 180° .

The same applies for the cross-polar vector \hat{e}_{cx} .

This polarization coordinate system is well-suited for presentation of the pattern from a reflector as shown in Figure 2, having its main beam in the direction of the z-axis. But for the feed, which is pointed in the opposite direction, the system is inappropriate since the main beam is coinciding with the pole. For feed pattern computations a new, local coordinate system should be introduced with its z-axis in the opposite direction of what is shown in the above figures.

Circular polarization is a simple combination of \hat{e}_{co} and \hat{e}_{cx} :

$$\hat{e}_{rhc} = \frac{1}{\sqrt{2}}(\hat{e}_{co} - j\hat{e}_{cx})$$

$$\hat{e}_{lhc} = \frac{1}{\sqrt{2}}(\hat{e}_{co} + j\hat{e}_{cx})$$

and also these vectors will rotate 4π on a small circle around the pole at $\theta = 180^\circ$. The amplitude of a RHC polarized field will not change in amplitude when the polarization vectors are rotated, but the phase will change with the rotation. Thus, a circular polarized field will show a phase change of 4π when moving around on a small circle near the pole.

Links

Alphabetical List of Classes and Command Types

Classes

Command Types

Applicable Units

File Extensions

File Formats

Appendices

Reference Section

Contents

Near-Field Calculations

In general, the field calculations in GRASP are based on a spherical wave expansion of the field of the specified source. Consequently, the source field is expanded in spherical modes and the field is calculated from these modes. Hereby is obtained a field that at any point satisfies Maxwell's equations, the field is continuous and has a correct graduate transition from the near field to the far field.

The feed models of *Gaussian Beam*, *Pattern Def.*, *Gaussian Beam, Near-Field Def.*, and *Hertzian Dipole* are exact. These models are themselves solutions to Maxwell's equations and a spherical wave expansion can not give a better modelling.

For some cases, for example to see the consequences of near-field effects, the far-field pattern may be forced to be applied at near-field distances. This simple model for the near field is denoted a 'distance-scaled' far field as the far-field pattern is multiplied by the complex factor

$$\frac{e^{-jkr}}{kr}$$

where r is the near-field distance and k is the wavenumber, $2\pi/\lambda$. This near-field model is obtained by specifying the attribute *far_forced* to 'on'.

A spherical wave expansion is not applied when *far_forced* is 'on', not even in the far field.

However, in general the near field will be determined from a spherical wave expansion of the far field, and the attribute *far_forced* has therefore as default the value 'off'. In this case the far field of the feed is automatically calculated at a sufficient number of points and a spherical wave expansion is determined. This expansion is stored and used each time the near field - or the field at any other point including the far field - is to be found.

The spherical wave expansion results in an accurate near-field determination and this is essential e.g. in PO calculations in order to determine the field incident on the reflector (or on any other scatterer) accurately. This requires, of course, a correspondingly accurate source model.

The near field may be determined accurately for all distances

$$r > N/k \tag{46}$$

where N is the highest value of the polar index n in the mode set. Depending on the source model from which the mode set derive, the field may be determined for smaller values of r in some part of the space, but a warning will be issued. When $r \leq N/k$ the modes with n in the interval $kr < n \leq N$ will be excluded and discontinuities may occur in the field at the values of r for which the number of excluded modes changes, $r = n/k$.

The main advantage of the spherical wave expansion is that a consistent field representation is obtained. But another advantage is that the number of spherical modes representing the field is limited by the physical size of the radiating source. Hereby simple *Feeds*, especially the *Simple Tapered Pattern (m=1)* and the *Cardioid Pattern*, become more realistic.

If the user wants to control the number of coefficients in the expansion of the source (e.g. for noise filtering by truncation) then the expansion may be performed by an object of class *Spherical Wave Expansion (SWE)*. The source field may be determined on basis of these coefficients by applying *Tabulated SWE Coefficients*.

Applications as ray optics, GO (Geometrical Optics) and GTD (Geometrical Theory of Diffraction), are based on far-field values at all distances. In such cases, the 'distance-scaled' far field may be used directly by setting the attribute *far_forced* to 'on'.

If *far_forced* is 'off', the GO/GTD calculations are still based on the far field but the applied far field is calculated from the spherical wave expansion of the given far field model.

The upper limit N of the polar index n in the mode set is given by

$$N = kr_0 + n_1 \quad (47)$$

where r_0 is the radius of the smallest sphere which has centre at the origin of the reference frame (the reference coordinate system of the source) and surrounds the complete source. All significant modes are included when $N = kr_0$ but some additional modes are needed as given by n_1 . A reasonable value for n_1 is $n_1 = 10$ but a more precise specification of n_1 is obtained by

$$n_1 = 0.045 \sqrt[3]{kr_0} (-P_{tr}) \quad (48)$$

where P_{tr} is the acceptable truncation error in dB relative to the total radiated power (P_{tr} is thus negative). In general a truncation error of $P_{tr} = -80$ dB is acceptable which results in

$$n_1 = 3.6 \sqrt[3]{kr_0} \quad (49)$$

and thus

$$N = kr_0 + 3.6 \sqrt[3]{kr_0} \quad (50)$$

n_1 shall, however, not be taken less than 10 and thus

$$N = kr_0 + \max(10, 3.6 \sqrt[3]{kr_0}) \quad (51)$$

where $\max()$ is a function taking the value of the largest of its arguments.

Links

[Alphabetical List of Classes and Command Types](#)

[Classes](#)

[Command Types](#)

[Applicable Units](#)

[File Extensions](#)

[File Formats](#)

[Appendices](#)

[Reference Section](#)

[Contents](#)

Phase of a Radiating Horn

A flare may be specified for a radiating horn. Usually the flare is specified by the slant *flare_length*, f_l , but an alternative is an indirect specification through a *semi_flare_angle*. In conjunction with the dimensions of the aperture, the position of the horn apex is then defined (see Figure 5).

This in turn defines the centre of curvature for the phase fronts of the aperture field. The phase of the aperture then varies as

$$e^{-jk\frac{x_f^2+y_f^2}{2f_l}} \quad (52)$$

where (x_f, y_f) are coordinates in the aperture. For the *Rectangular Horn* the phase of the aperture varies as

$$e^{-jk\left(\frac{x_f^2}{2f_x}+\frac{y_f^2}{2f_y}\right)} \quad (53)$$

where f_x and f_y are the slant flare lengths in the x_f, z_f - and the y_f, z_f -plane, respectively.

If the *flare_length* is specified to be less than the radius of the aperture (or a *semi_flare_angle* is specified to zero), then the feed will be considered as an open-ended waveguide and the phase of the aperture field will be constant.

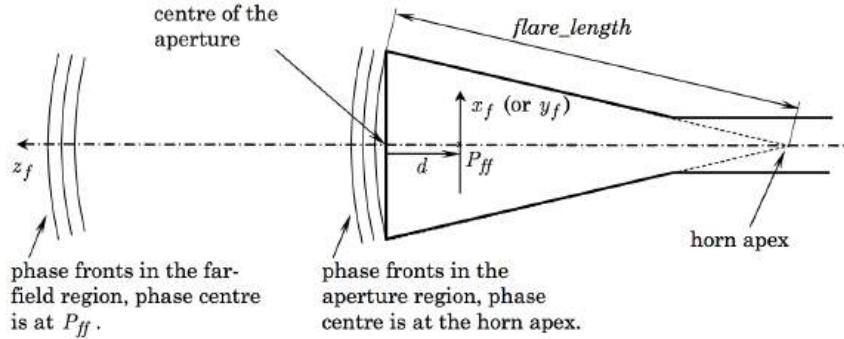


Figure 5 Use of phase reference displacement d . The reference point for the phase of the field is at P_{ff} ($d < 0$ in this drawing).

The far field is determined from the waveguide modes specified in the aperture together with the above-defined phase variation. By default, the phase of the far field refer to the centre of the horn aperture. However, the 'true' phase centre, P_{ff} , of the far field is usually positioned somewhere inside the horn between the aperture and the apex.

The phase reference for the far field may be changed to P_{ff} by specifying a *phase_displacement*. A positive *phase_displacement* moves the phase reference point in the positive direction of the horn axis, z_f . When the phase centre is positioned inside the horn, which is the most usual, the *phase_displacement*, d , shall be specified with a negative value.

The phase reference is displaced from the centre of the aperture to P_{ff} by multiplying the far field by the factor

$$e^{jkd(1-\cos\theta)} \quad (54)$$

where θ is the direction to the far-field measured from the z_f -axis.

The horn (or waveguide) aperture is defined in a feed coordinate system (x_f, y_f, z_f) with the horn axis being the z_f -axis. The origin is at P_{ff} , as defined by the *phase_displacement d*. This is in agreement with positioning the origin of the feed coordinate system at the focus of a focused reflector coordinate system.

Links

[Alphabetical List of Classes and Command Types](#)

[Classes](#)

[Command Types](#)

[Applicable Units](#)

[File Extensions](#)

[File Formats](#)

[Appendices](#)

[Reference Section](#)

[Contents](#)