ansys-aedt-toolkits-radar

ANSYS, Inc.
Southpointe
2600 Ansys Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
http://www.ansys.com
(T) 724-746-3304
(F) 724-514-9494

Feb 08, 2026

CONTENTS

Useful links: *Installation* | Source repository | Issues

The Radar Explorer Toolkit is a Python interface for creating and analyzing radar simulations. It helps you create and modify simulations, visualize results, and generate both standard and customized reports. This toolkit provides preprocessing and postprocessing radar tools. It supports various simulation modes, including RCS (Radar Cross-Section), range profiles, waterfall, and 2D/3D ISAR (Inverse Synthetic Aperture Radar).

Getting started Learn how to install and launch the toolkit.

Getting started User guide Learn how to use the toolkit.

User guide API reference Explore the APIs available for the toolkit.

API reference Examples Explore examples that show how to use the toolkit.

Examples Contribute Learn how to contribute to the toolkit codebase or documentation.

Contribute

CHAPTER

ONE

GETTING STARTED

This section explains how to install and launch the Radar Explorer Toolkit.

Installation Learn how to install and launch the toolkit.

Installation

1.1 Installation

You can install the Radar Explorer Toolkit using the latest installer from the repositorys Releases page or install it from PyPI.

1.1.1 Install the toolkit using the installer

Install the Radar Explorer Toolkit for your operating system using the latest installer:

Windows

To install the toolkit on Windows, follow these steps:

1. Download the latest installer for Windows from the repositorys Releases page.

   The file is named `Radar-Explorer-Toolkit-Installer-windows.exe`.

2. Run the installer.

3. Search for `Radar Explorer Toolkit` and launch it.

The **Radar Explorer Toolkit** window opens.

Linux - Ubuntu

Ubuntu

Supported Linux operating systems are Ubuntu 24.04 and 22.04. To install the toolkit on a supported Ubuntu version, follow these steps:

1. Update the `apt-get` repository and install required packages with **sudo** privileges:

   ```
   sudo apt-get update -y
   sudo apt-get install wget gnome libffi-dev libssl-dev libsqlite3-dev libxcb-
   ↪xinerama0 build-essential -y
   ```

2. Download and install the `zlib` library:

```
wget https://zlib.net/current/zlib.tar.gz
tar xvzf zlib.tar.gz
cd zlib-*
make clean
./configure
make
sudo make install
```

3. Download the latest Radar Explorer Toolkit installer for Ubuntu from the repositorys Releases page.

   The file is named `Radar-Explorer-Toolkit-Installer-ubuntu_*.zip`.

4. Run this command in the terminal:

```
unzip Radar-Explorer-Toolkit-ubuntu_*.zip
./installer.sh
```

5. Search for `Radar Explorer Toolkit` and launch it.

The **Radar Explorer Toolkit** window opens.

1.1.2 Uninstall the toolkit

To uninstall the Radar Explorer Toolkit, follow these steps:

1. From the toolkits menu, select **File > Uninstall**.
2. Click **Uninstall**.

1.1.3 Install the toolkit with Python

Install the Radar Explorer Toolkit from PyPI like any other open source Python package. You can install both the backend (AI) and user interface (UI) methods or only the backend methods.

> **Note**
>
> - If you have an existing virtual environment, skip step 1.
> - If you have already installed the toolkit in your virtual environment, skip step 2.

1. Create and activate a new Python virtual environment:

```
# Create a virtual environment
python -m venv .venv

# Activate it in a POSIX system
source .venv/bin/activate

# Activate it in a Windows CMD environment
.venv\Scripts\activate.bat

# Activate it in Windows PowerShell
.venv\Scripts\ActivateT.op si1
```

2. To install both the backend and UI methods from the GitHub repository:

```
python -m pip install ansys-aedt-toolkits-radar-explorer[all]
```

3. To install only the backend methods:

```
pip install ansys-aedt-toolkits-radar-explorer
```

4. If you installed both the backend and UI methods, launch the toolkit:

```
python .venv\Lib\site-packages\ansys\aedt\toolkits\radar_explorer\run_toolkit.py
```

Note

If you are a developer wanting to contribute to the Radar Explorer Toolkit, see *Contribute*.

CHAPTER

TWO

USER GUIDE

This section explains how to use the Radar Explorer Toolkit. It assumes that you have already installed and launched the toolkit as indicated in *Installation*.

Settings Learn about the toolkits general settings.

Settings Workflows Learn about the available workflows.

Workflows Radar configuration Learn about the radar configuration panels.

Radar configuration Post 3D Learn about 3D postprocessing.

3D Postprocessing Post 2D Learn about 2D postprocessing.

2D Postprocessing

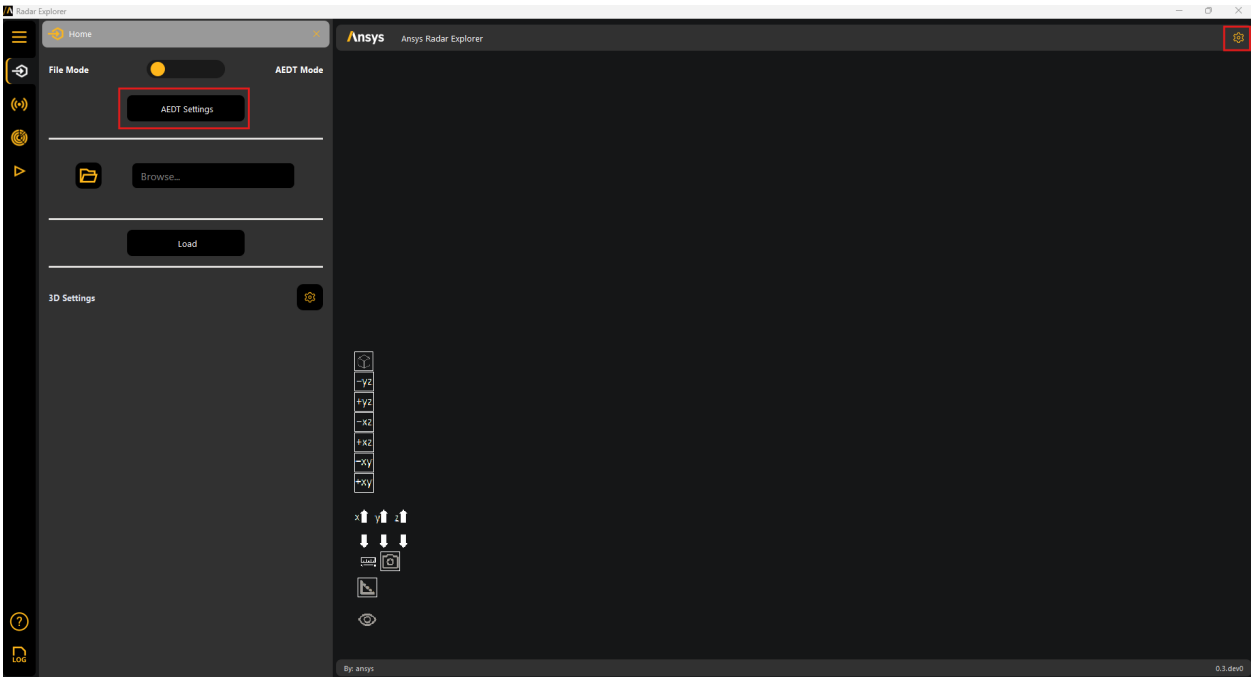
2.1 Settings

The **Settings** panel lets you configure options for launching or connecting to an AEDT session in graphical or non-graphical mode.

1. Open an existing project (if not already open) and specify the desired AEDT version.

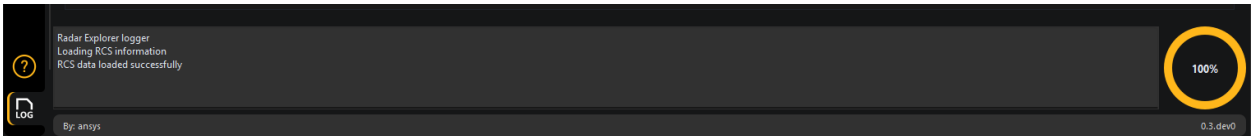
   After the toolkit connects, the **Settings** panel becomes unavailable.

2. Access the **Settings** panel from the tool icon or **Home** panel:



2.1.1 Toolkit progress

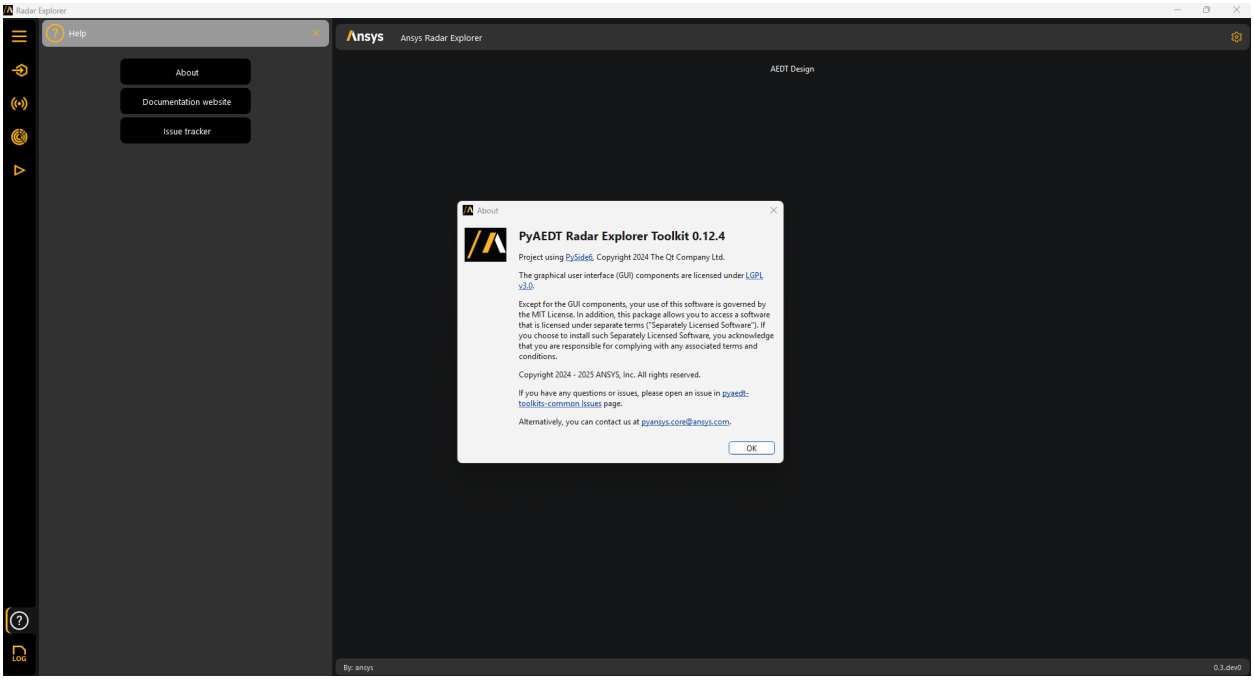
The toolkit interface includes a progress indicator and a logger box that displays the status of ongoing operations:



2.1.2 Help

Use the **Help** panel to access the following options:

- **About**: Displays information about the toolkit.
- **Documentation website**: Opens the official documentation website.
- **Issue tracker**: Opens the repositorys **Issues** page, where you can submit bugs or request enhancements.



2.2 Workflows

The **Home** panel provides workflows to perform these tasks:

- *Import a geometry*
- *Load a solved HFSS design*
- *Duplicate an HFSS design*
- *Load an analysis setup file*

> **Note**
>
> The workflows for importing a geometry, loading a solved HFSS design, and duplicating an HFSS design are available only when the Radar Explorer Toolkit is connected to AEDT. For more information, see *Settings*.

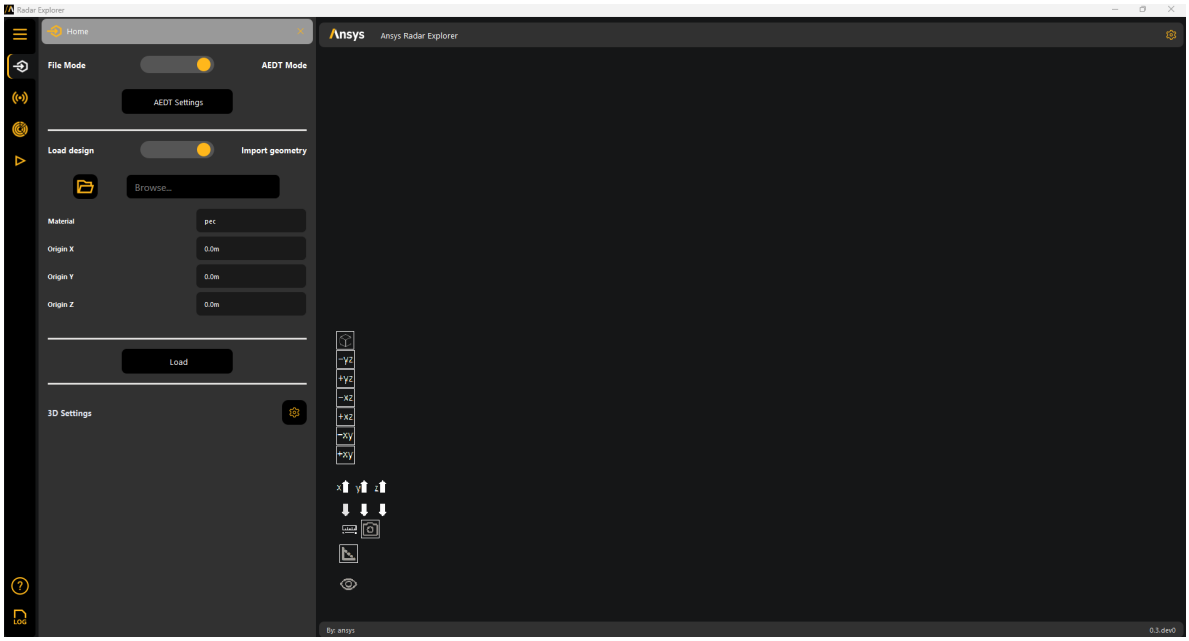
The **Home** panel also lets you access 3D settings. For more information, see *Access 3D settings*.

2.3 Import a geometry

Use this workflow to import a geometry file into the toolkits plotter window and HFSS:

1. Go to the **Home** panel on the left sidebar.
2. Set the toggles to **AEDT Mode** and **Import geometry**.
3. Click the file icon to browse to and select a geometry file.

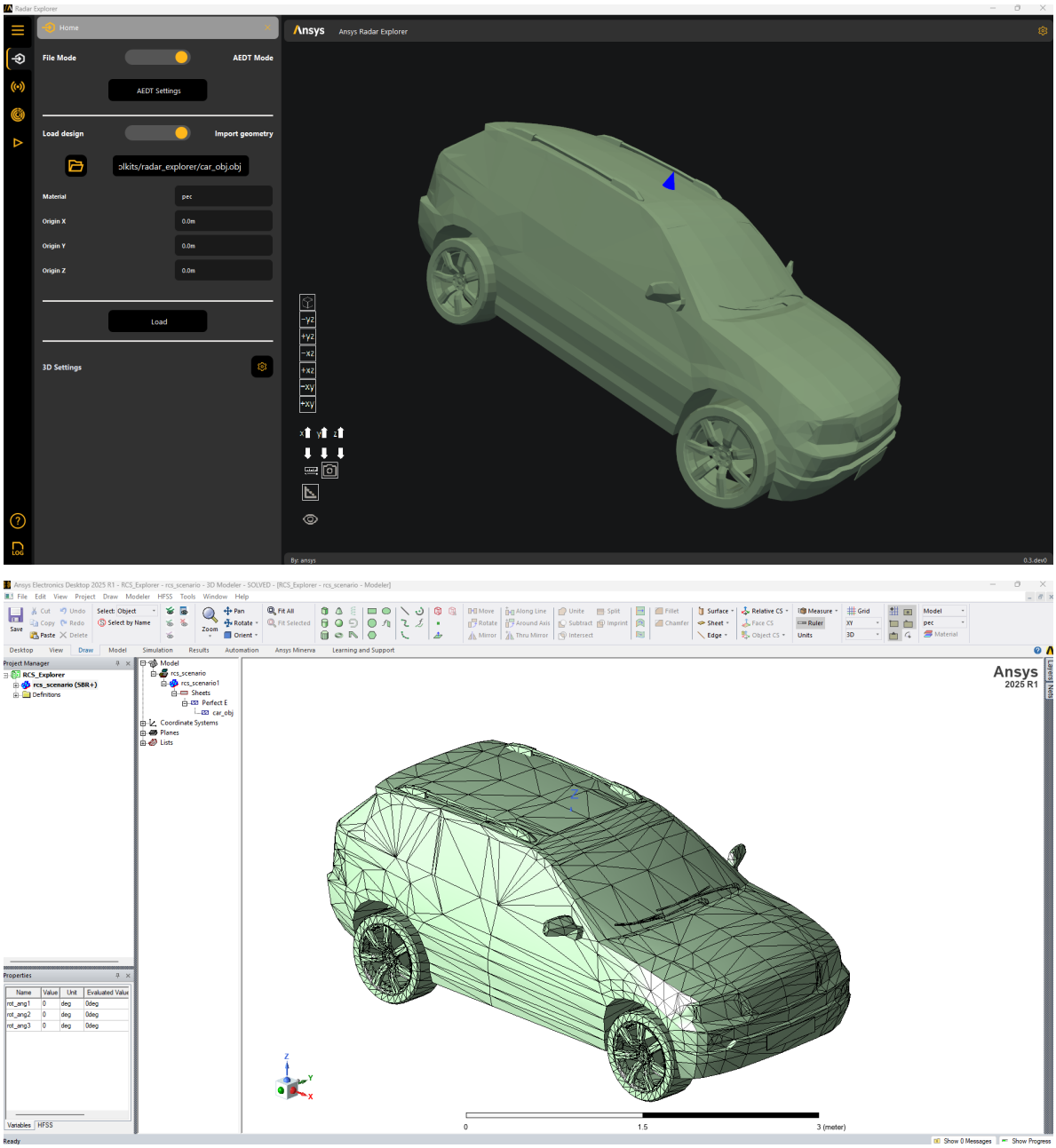
   Supported formats include STL, OBJ, GLTF, and GLB.
4. Edit the material, which must be available in HFSS materials, and position the geometry relative to the global coordinate system.



5. Click **Load** to import the selected geometry.

   The **Load** button becomes unavailable to prevent redundant uploads.

   Once loaded, the geometry appears in the plotter window and is also available in HFSS.



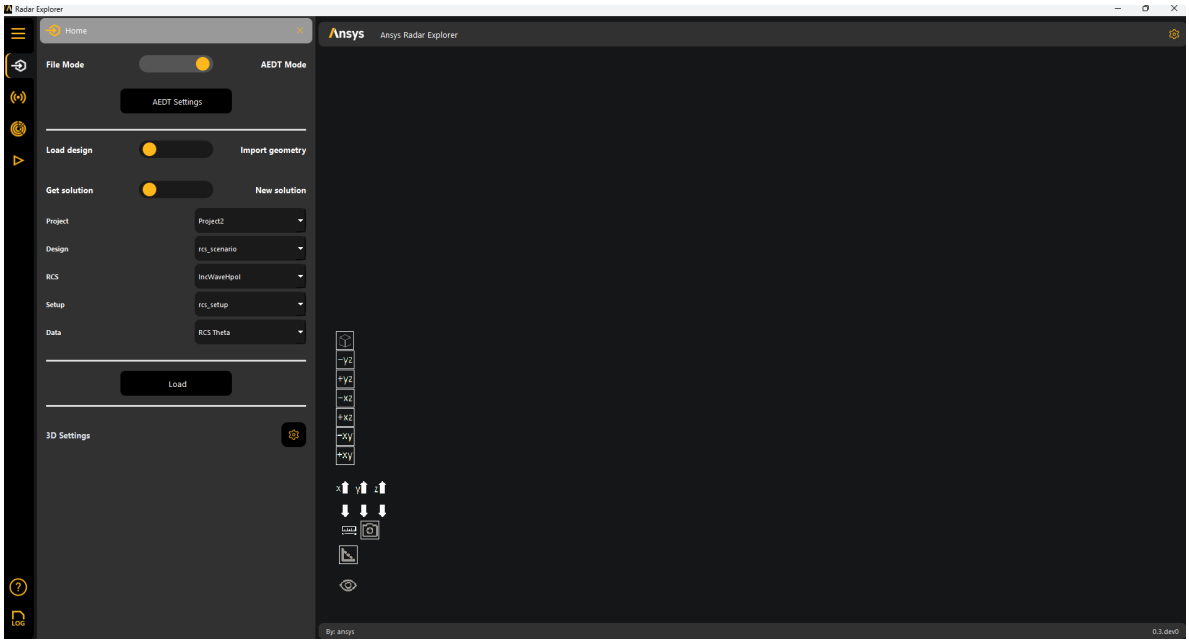
6. Change 3D settings if needed. For more information, see *Access 3D settings*.
7. Start radar configuration. For more information, see *Radar configuration*.

2.4 Load a solved HFSS design

Use this workflow to load a solved HFSS SBR+ design into the toolkit:

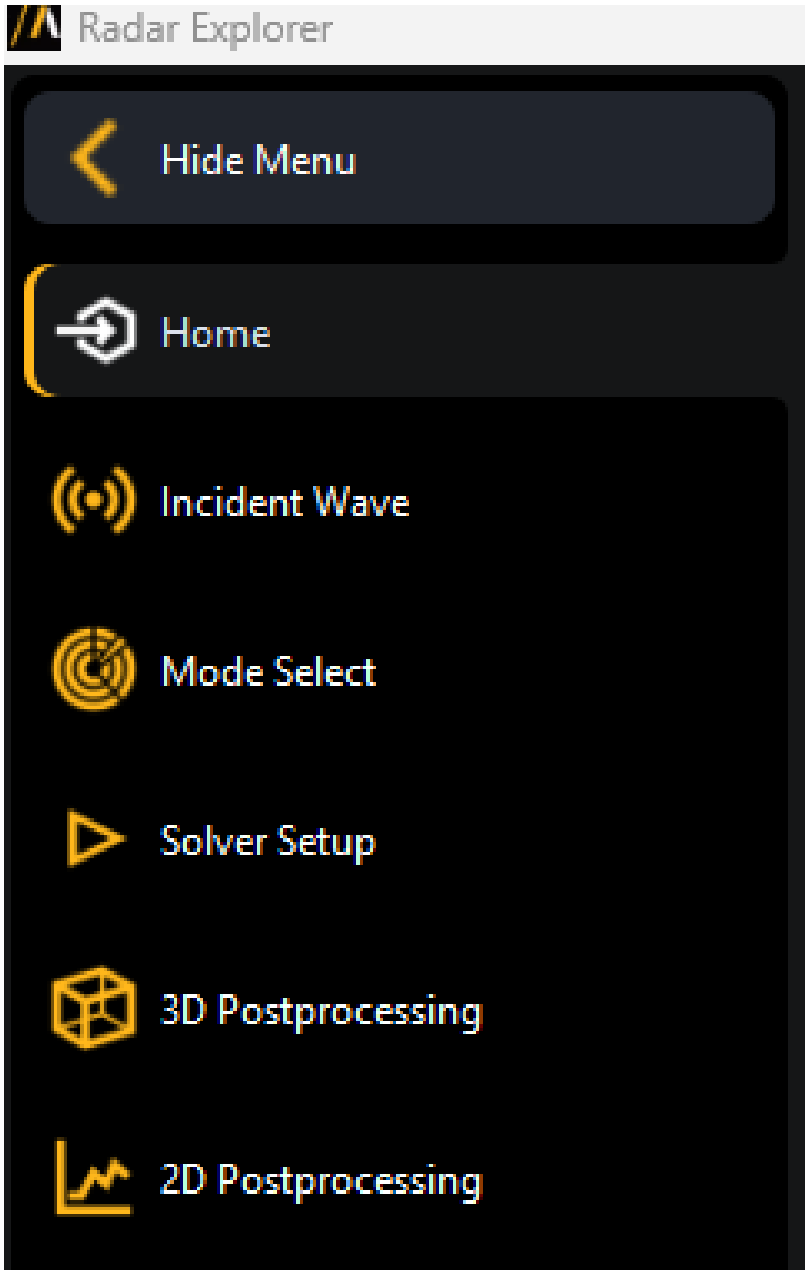
1. Go to the **Home** panel on the left sidebar.
2. Set the toggles to **AEDT mode** and **Load design**.
3. Ensure that you are connected to an active AEDT session with a solved design open.

   The design detail text boxes automatically populate with the relevant data from the connected session.



4. Set the toggle to **Get solution**.
5. Click **Load** to load the solved HFSS SBR+ design.

   If the results load correctly, In the menu, two additional panels become available for 3D and 2D postprocessing:



For more information, see *3D postprocessing* and *2D postprocessing*.

6. Change 3D settings if needed. For more information, see *Access 3D settings*.

Note

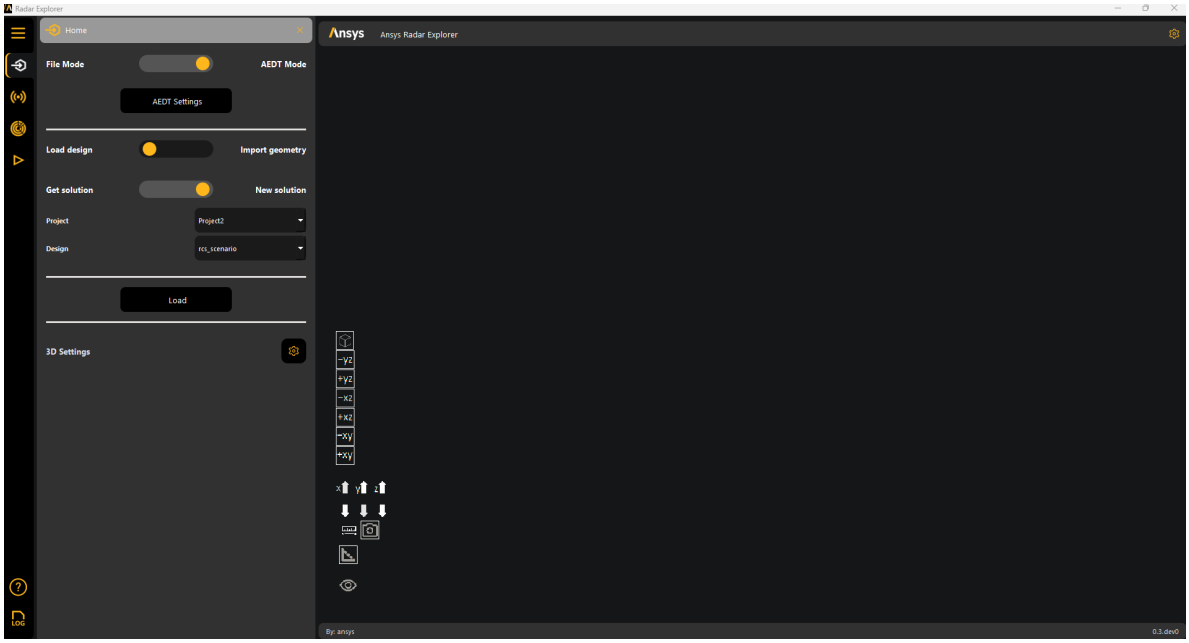
Once the design loads, you cannot modify the radar configuration.

2.5 Duplicate an HFSS design

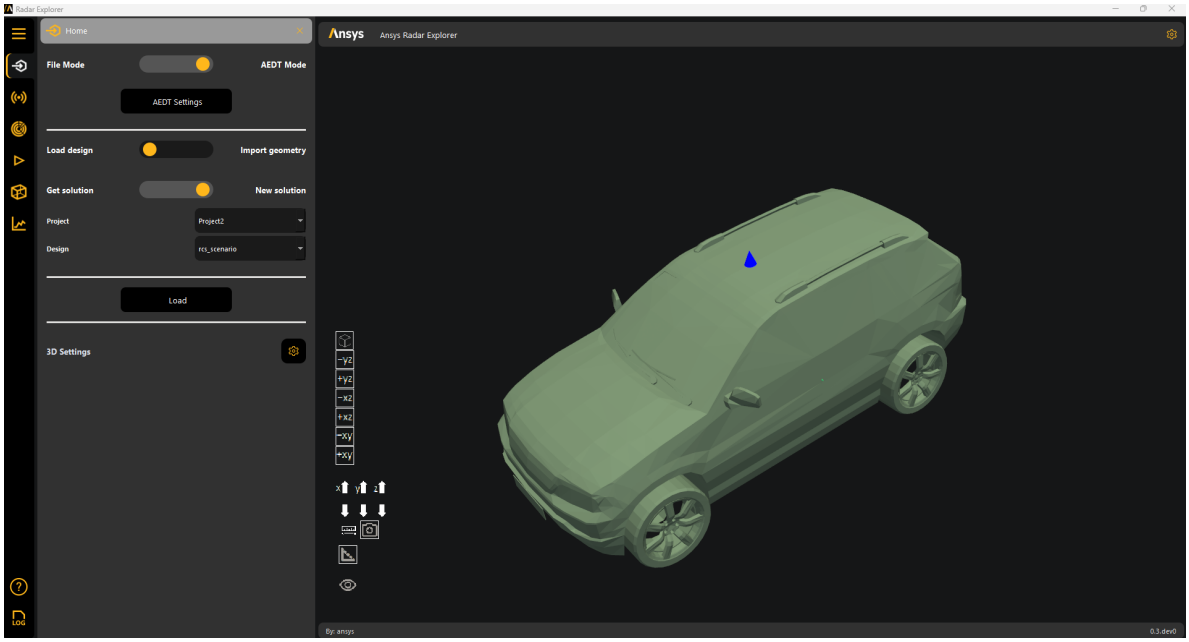
Use this workflow to duplicate an HFSS SBR+ design:

1. Go to the **Home** menu on the left sidebar.
2. Set the toggles to **AEDT Mode** and **Load design**.
3. Ensure that you are connected to an active AEDT session with a solved HFSS SBR+ open.

   The design detail text boxes automatically populate with data from the connected session.



4. Set the toggle to **New solution**.
5. Click **Load** to load the solved HFSS SBR+ design.



In the menu, two additional panels become available for 3D and 2D postprocessing. For more information, see *3D postprocessing* and *2D postprocessing*.

6. Change 3D settings if needed. For more information, see *Access 3D settings*.
7. Start radar configuration. For more information, see *Radar configuration*.

2.6 Load an analysis setup file

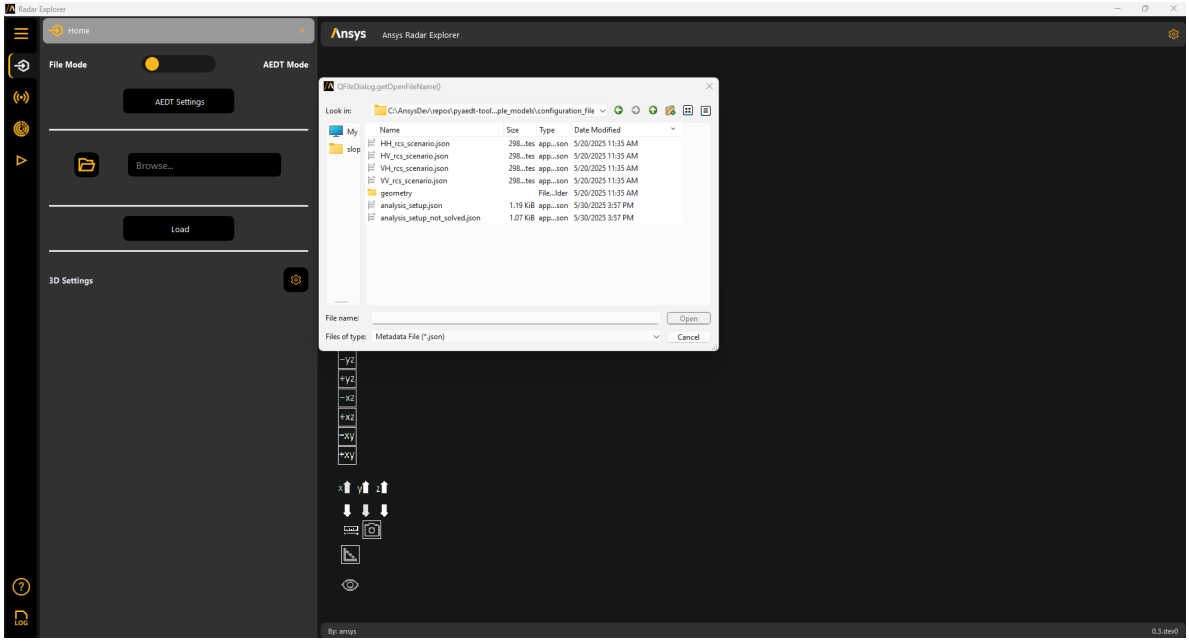
Use this workflow to reload previous work by importing an analysis setup file or radar metadata file.

Note

These files, generated by the toolkit when a simulation launches, are located in the `*YourProjectName*.pyaedt` directory.

- If the file contains results, you do not need to connect to HFSS.
- If the file contains results, you cannot modify the radar configuration.

1. Go to the **Home** panel on the left sidebar.
2. Set the toggle to **File Mode**.
3. Click the file icon to browse to and select an exported data file.

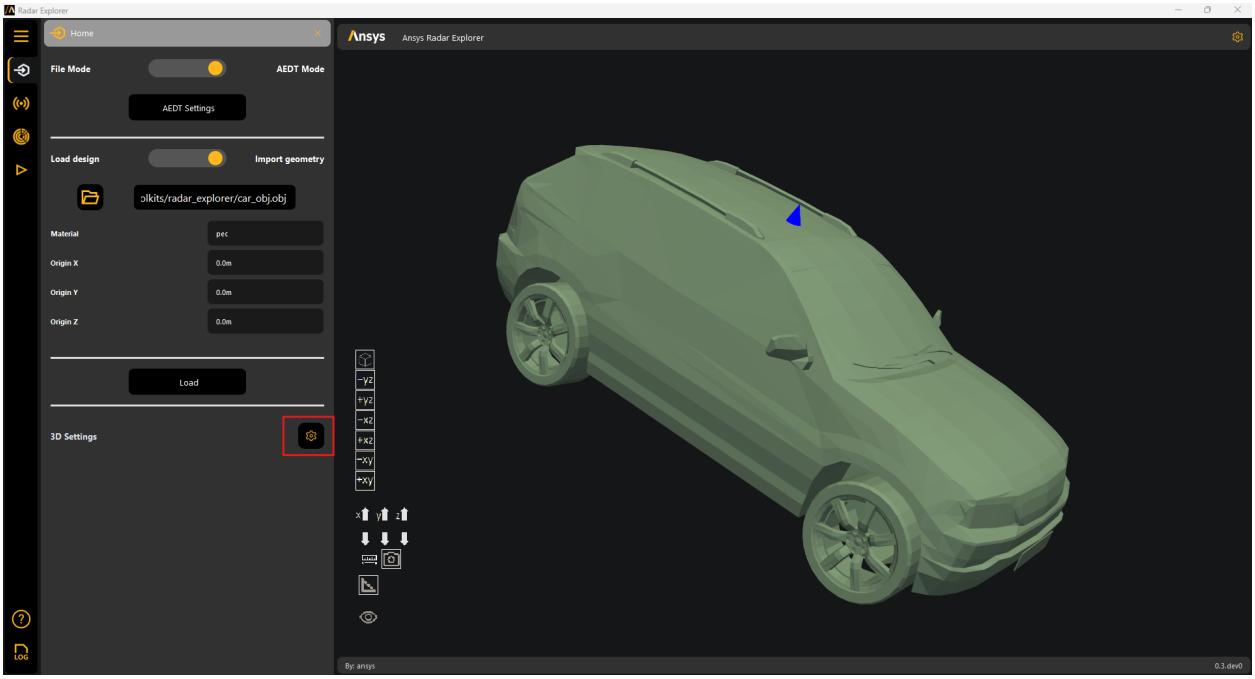


In the menu, two additional panels become available for 3D and 2D postprocessing. For more information, see *3D postprocessing* and *2D postprocessing*.

4. Change 3D settings if needed. For more information, see *Access 3D settings*.

2.7 Access 3D settings

Access the **3D settings** panel by clicking the gear icon in the **Home** panel:



The 3D settings shown under **Model settings** and **Plot settings** depend on whether an *object* or *solved data* is loaded. For either an object or solved data, the **Projection** setting can be **Perspective** or **Orthographic**.

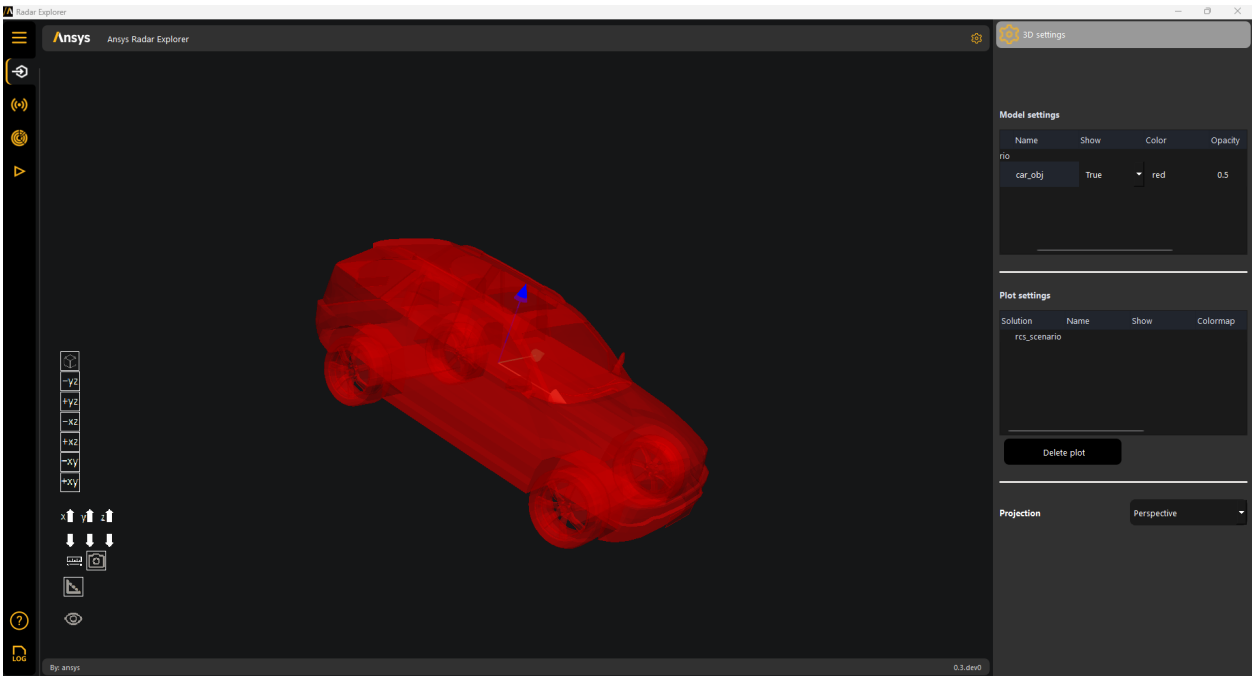
2.7.1 3D model and plot settings for an object

Under **Model settings**, these settings are available for an object:

- **Name**: Name of the object.
- **Show**: Whether to show the object.
- **Color**: Color to show the object in. You can choose from PyVista color options.
- **Opacity**: Value between 0 (transparent) and 1 (opaque).

Under **Plot settings**, these settings are available for an object:

- **Solution**: Name of the solution.
- **Name**: Name of the object, which is read-only.
- **Show**: Whether to show the object.
- **Colormap**: Color mapping for the results.



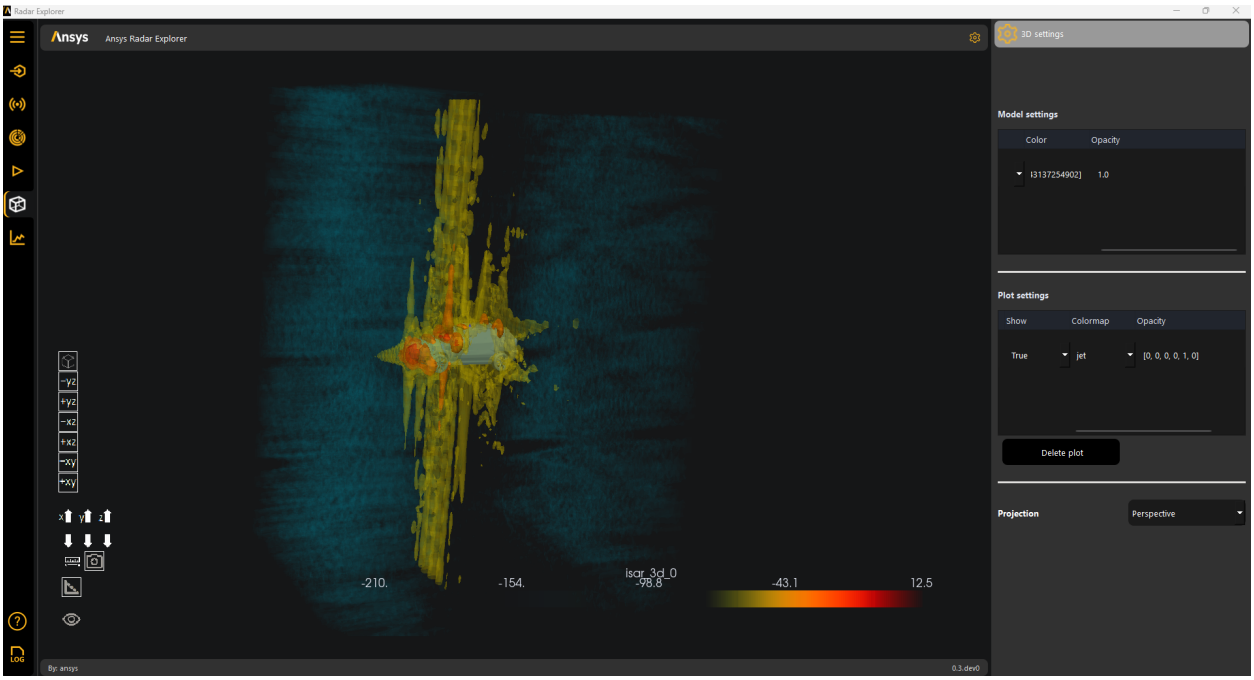
2.7.2 3D model and plot settings for solved data

Under **Model settings**, these settings are available for solved data:

- **Color**: Color to show the data in.
- **Opacity**: Value between 0 (transparent) and 1 (opaque).

Under **Plot settings**, these settings are available for solved data:

- **Show**: Whether to show the data.
- **Colormap**: Color mapping for the results.
- **Opacity**: Choose from available PyVista opacity options



2.8 Radar configuration

You can access the following radar configuration panels:

- *Incident Wave panel*
- *Mode Select panel*
- *Solver Setup*

> **Note**
>
> These panels are enabled only for the workflows for *importing a geometry*, *duplicating an HFSS design*, and *loading an analysis setup file* if the file does not contain results.

These panels also provide access to 3D settings. For more information, see *Access 3D settings*.

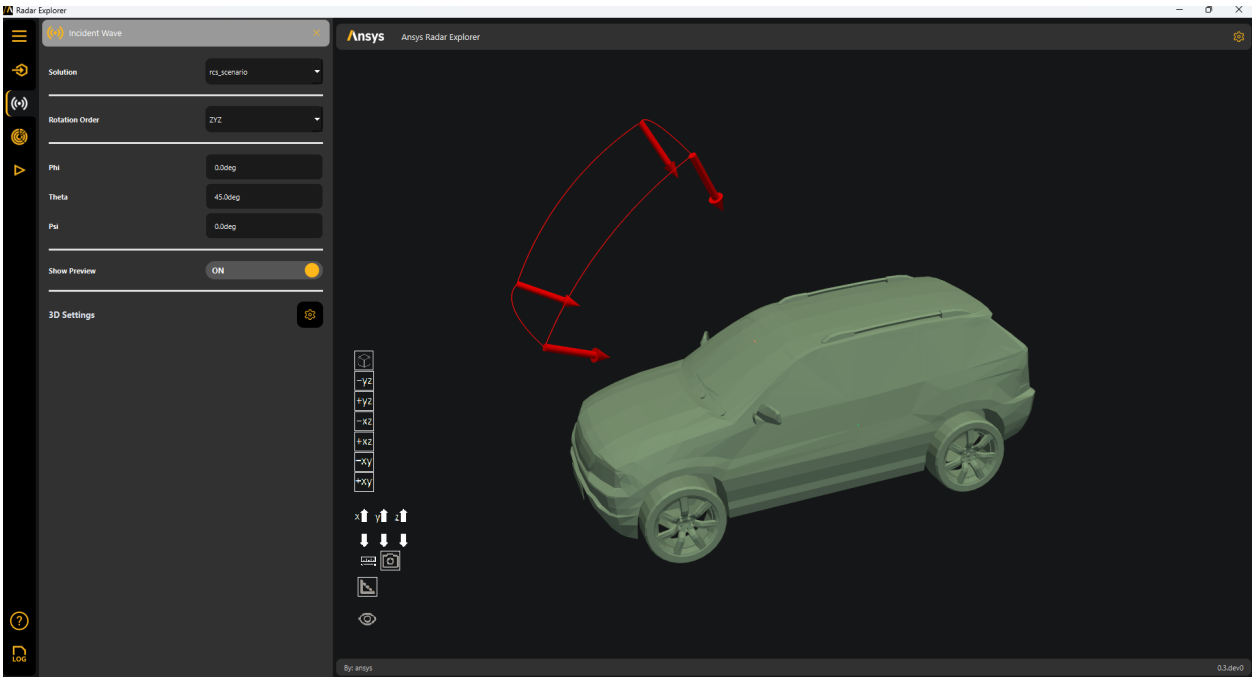
2.8.1 Incident wave panel

1. Go to the **Incident Wave** panel from the left sidebar.
2. Use this panel to rotate the model so it aligns with the radars main direction of incidence and to visualize the radars incidence and observation domains around the chosen center.

   This tool uses **Euler (ZYZ)** angles. Control orientation with these three parameters:

   - **Phi ():** Rotate about the **global Z** axis.
   - **Theta ():** Rotate about the **intermediate Y** axis (after).
   - **Psi ():** Rotate about the **final Z** axis (after and).

   All rotations follow the right-hand rule.
3. Set **Show Preview** to **ON** to display incident wave arrows in the plotter window:

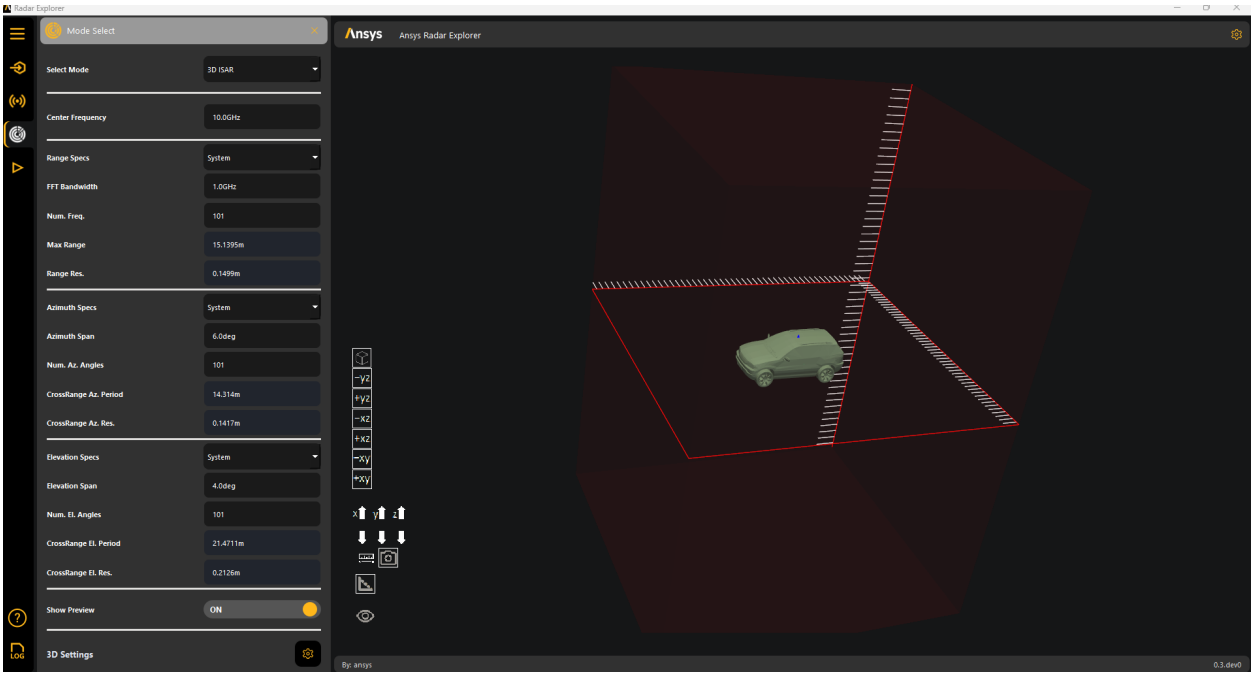


Note

The arrows change based on the mode selected in the *Mode Select* panel.

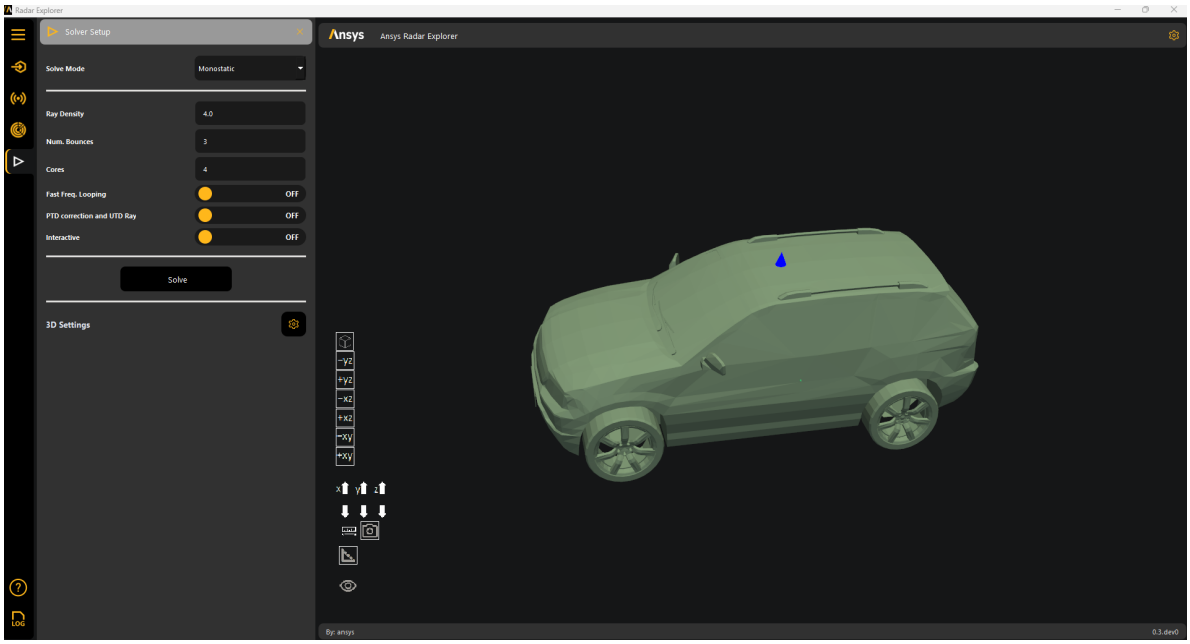
2.8.2 Mode select panel

1. Go to the **Mode Select** panel from the left sidebar.
2. For **Select Mode**, choose one of the following solve modes:
   - **Range Profile**
   - **2D ISAR**
   - **3D ISAR**
3. Set **Show Preview** to **ON** to display incident wave arrows in the plotter window:



2.8.3 Solver setup panel

1. Go to the **Solver Setup** panel from the left sidebar.
2. Configure the simulation parameters:
   - **Ray Density:** Number of rays to simulate for accuracy.
   - **Number of Bounces:** Maximum reflections a ray undergoes.
   - **Cores:** Number of cores to use. Using more cores speeds up simulations but might reduce accuracy.
   - **Fast Freq. Looping:** Whether to enable faster simulations at the cost of reduced accuracy.
   - **PTD correction and UTD Ray:** Whether to enable PTD and UTD in the SBR+ simulation.
   - **Interactive:** Whether to enable solving in graphical mode. When set to **OFF**, the simulation runs in non-graphical mode, thereby avoiding user interaction during simulation time, which might invalidate the simulation.



3. Click **Solve** to run the analysis.

   After the simulation finishes, the system automatically extracts and loads the computed RCS data.

4. Check the output logs for any warnings or errors generated during the simulation.

After solving, the menu makes two additional panels available for 3D and 2D postprocessing. For more information, see *3D Postprocessing* and *2D Postprocessing*.

2.9 3D Postprocessing

The menu only displays the **3D Postprocessing** panel if the design is solved and results are extracted by the toolkit.

You have five 3D postprocessing categories:

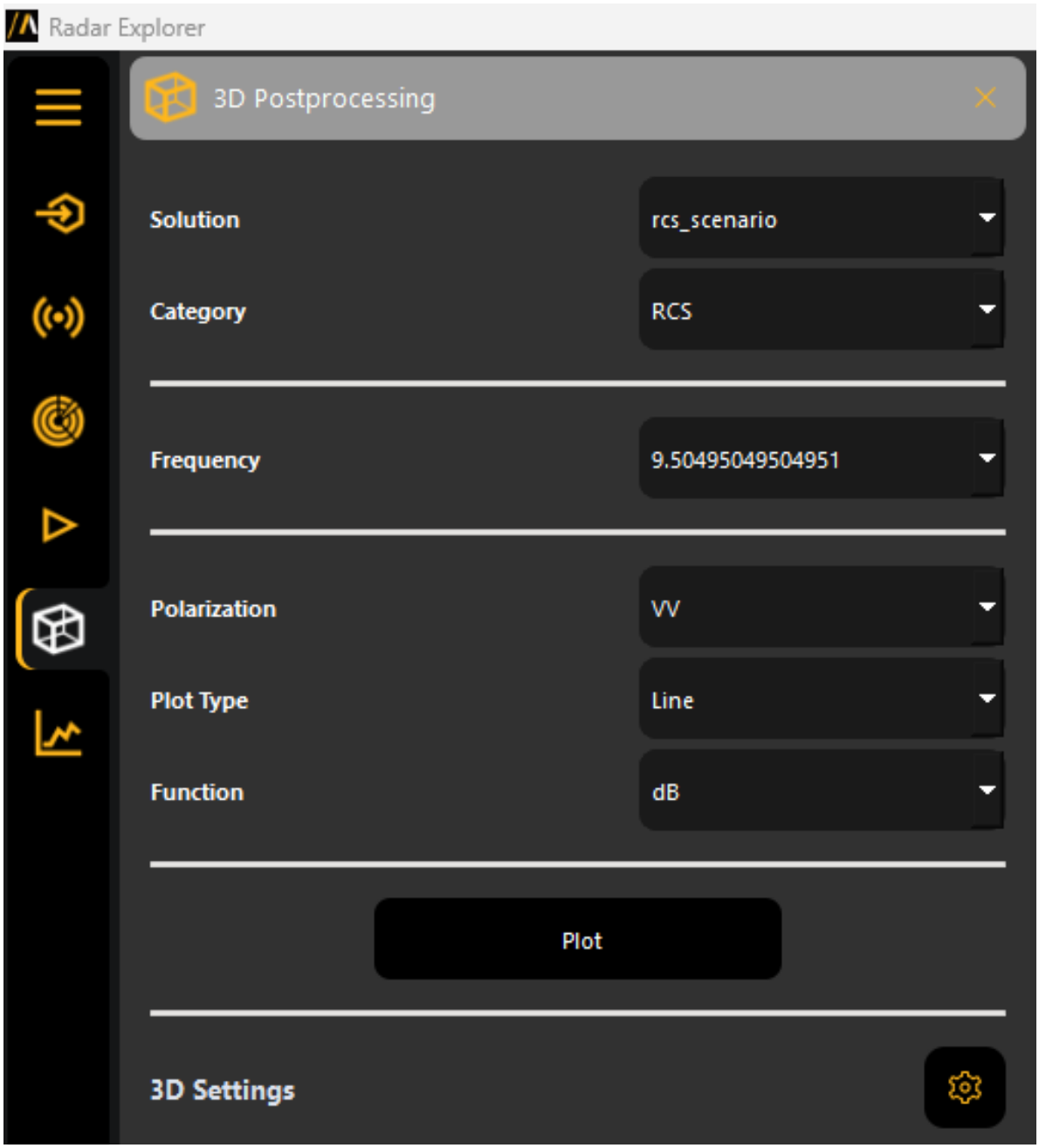
- *RCS*
- *Range profile*
- *Waterfall*
- *ISAR 2D*
- *ISAR 3D*

Note

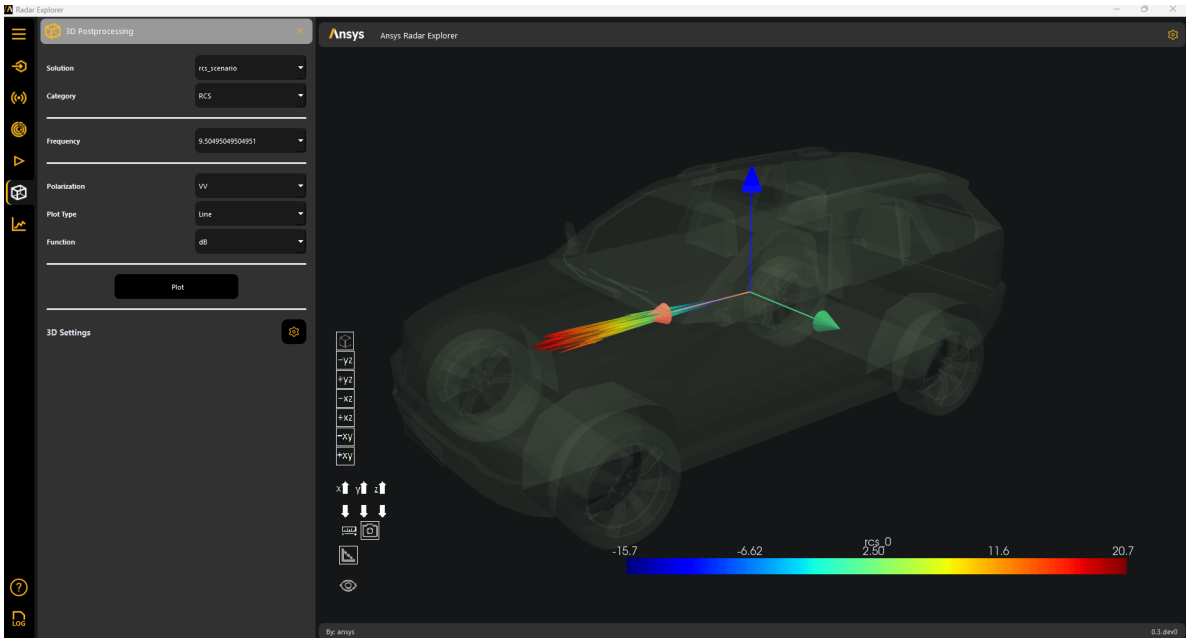
- You can access *3D settings* to change the plot settings.
- In the following images, **Solution** is set to **rcs_scenario**, which is for a car.

2.10 RCS

1. Go to the **3D Postprocessing** panel from the left sidebar. In the following image, **RCS** is the selected postprocessing category.

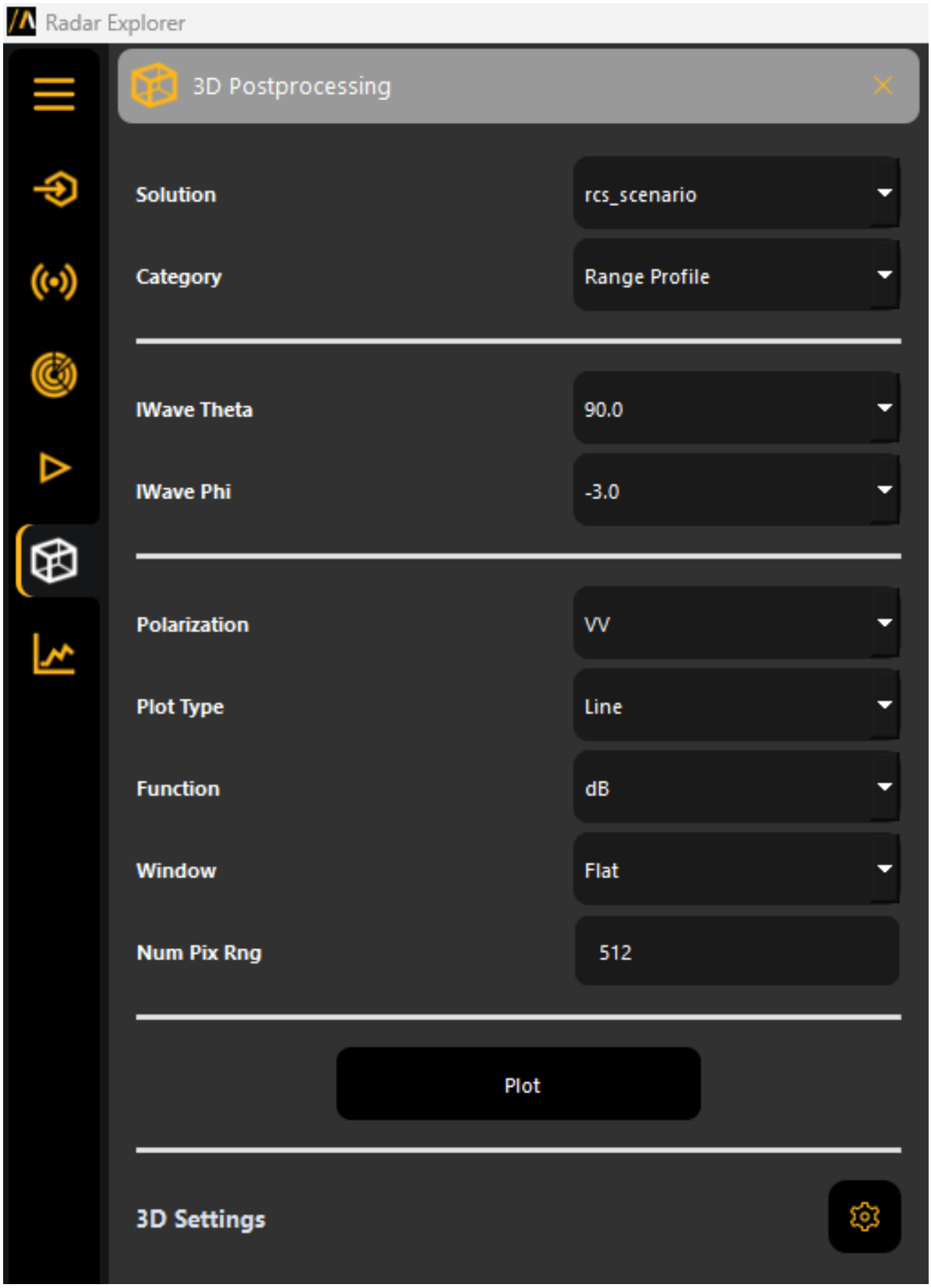


2. Click **Plot** to generate a 3D plot based on the specified settings.

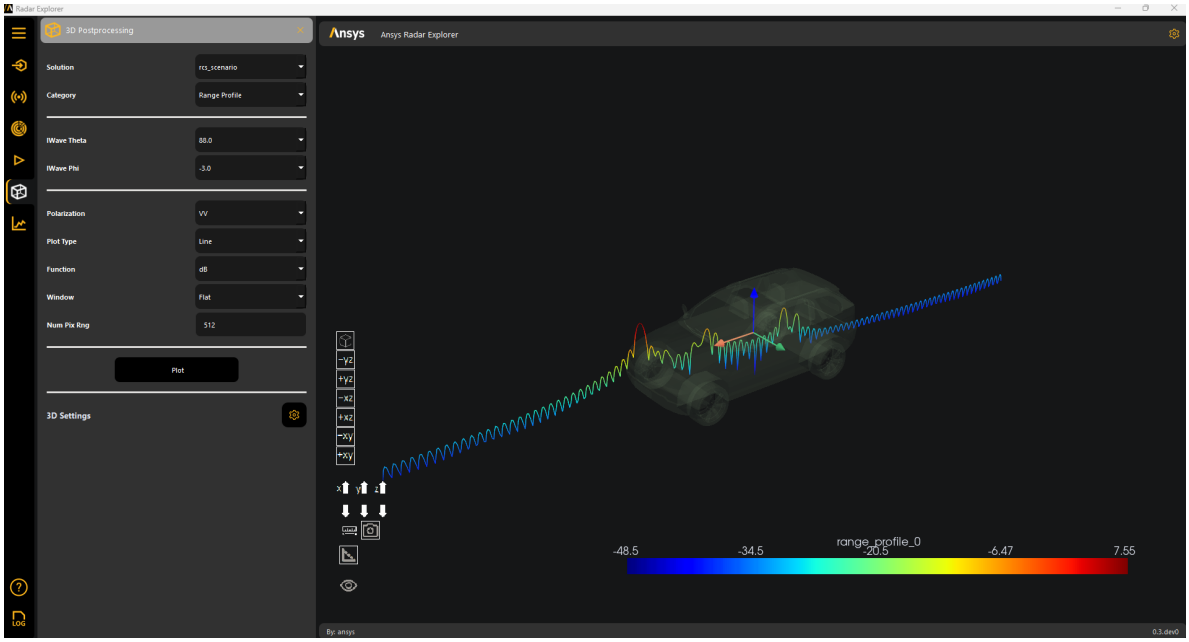


2.11 Range profile

1. Go to the **3D Postprocessing** panel from the left sidebar. In the following image, **Range Profile** is the selected postprocessing category.

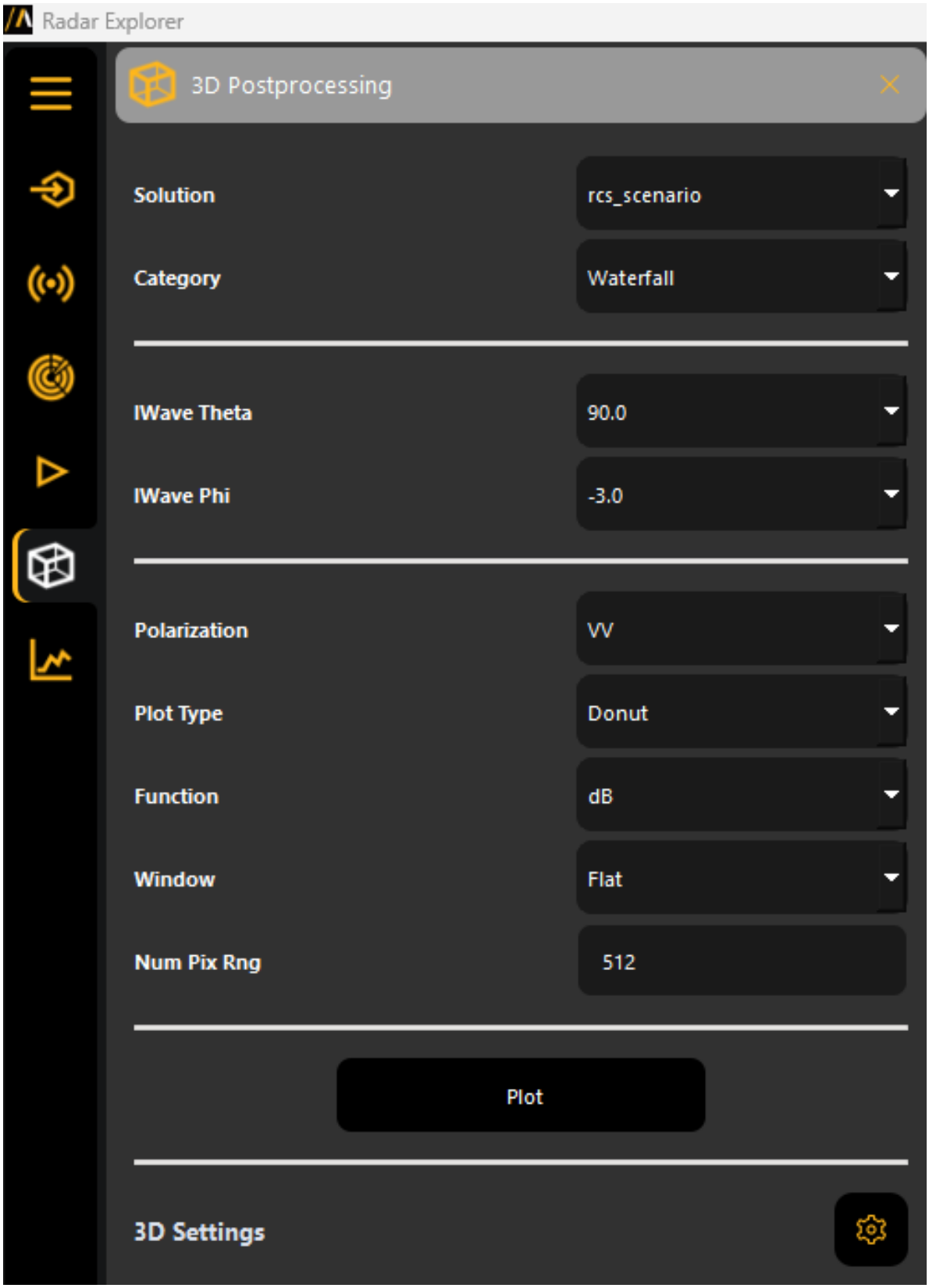


2. Click **Plot** to generate a 3D plot based on the specified settings.

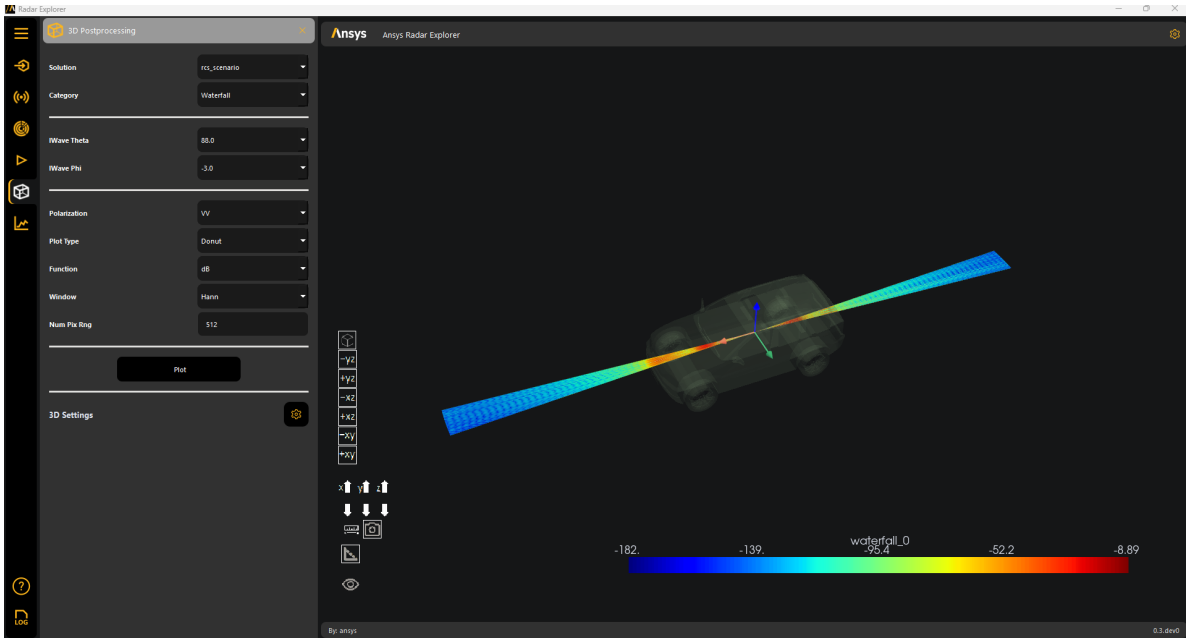


2.12 Waterfall

1. Go to the **3D Postprocessing** panel from the left sidebar. In the following image, **Waterfall** is the selected postprocessing category.

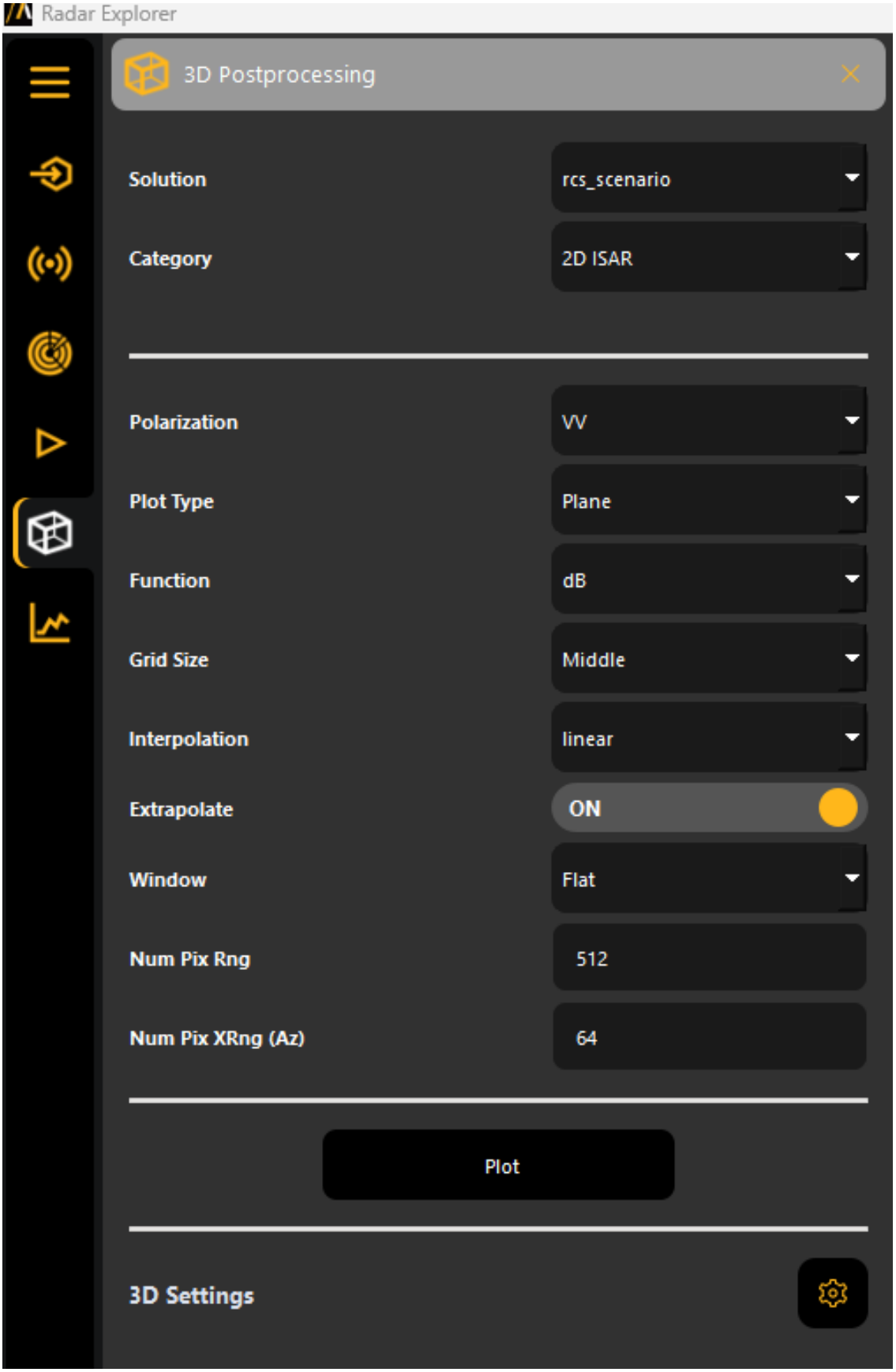


2. Click **Plot** to generate a 3D plot based on the specified settings.

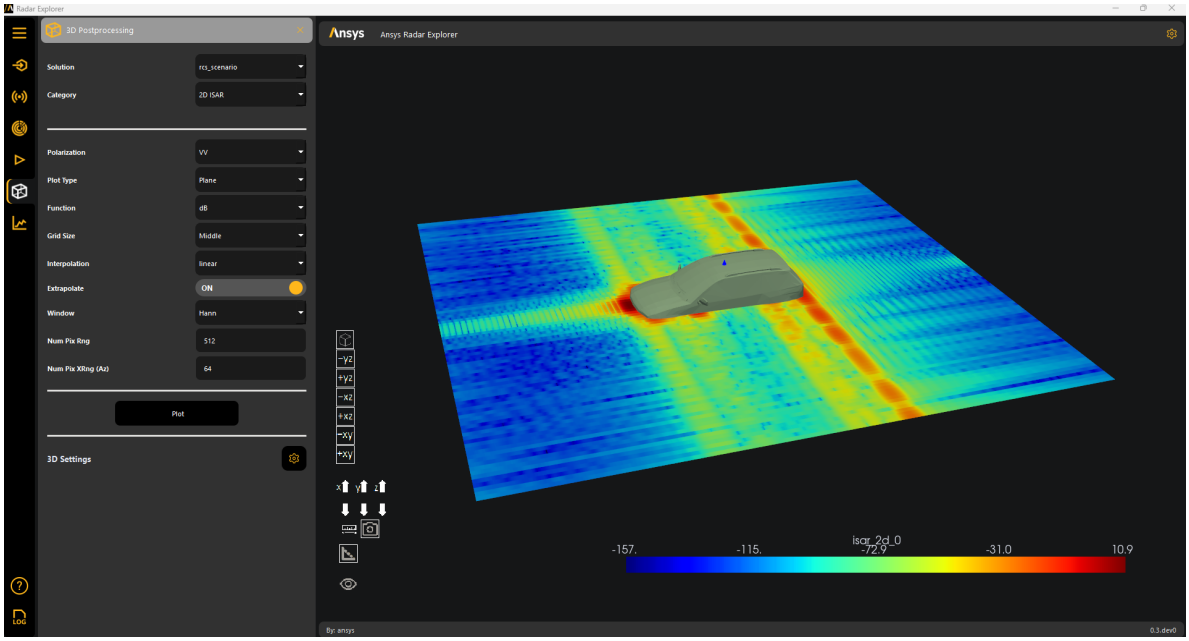


2.13 ISAR 2D

1. Go to the **3D Postprocessing** panel from the left sidebar. In the following image, **2D ISAR** is the selected postprocessing category.

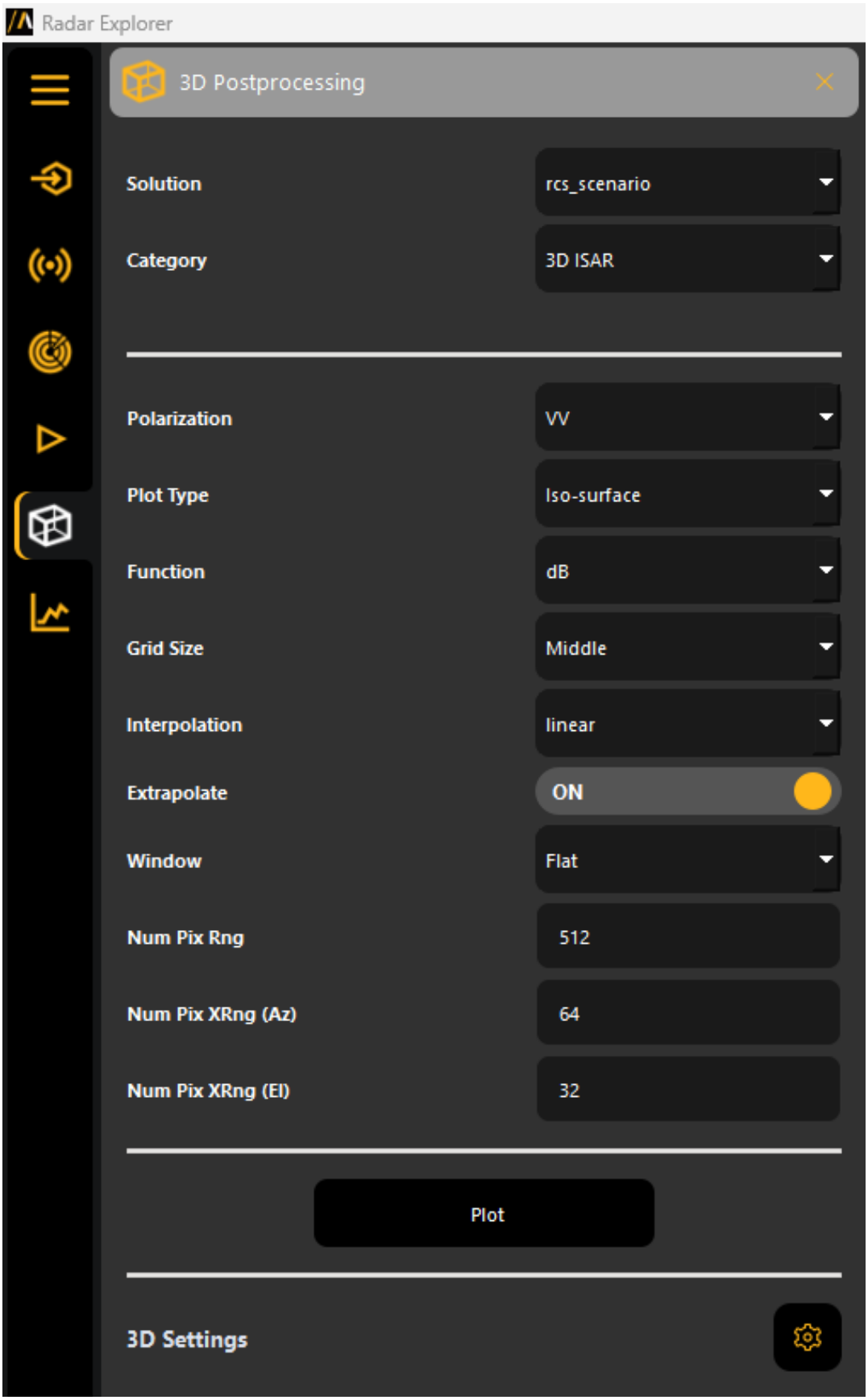

Radar Explorer
3D Postprocessing
Solution
rcs_scenario
Category
2D ISAR
Polarization
VV
Plot Type
Plane
Function
dB
Grid Size
Middle
Interpolation
linear
Extrapolate
ON
Window
Flat
Num Pix Rng
512
Num Pix XRng (Az)
64
Plot
3D Settings

2. Click **Plot** to generate a 3D plot based on the specified settings.

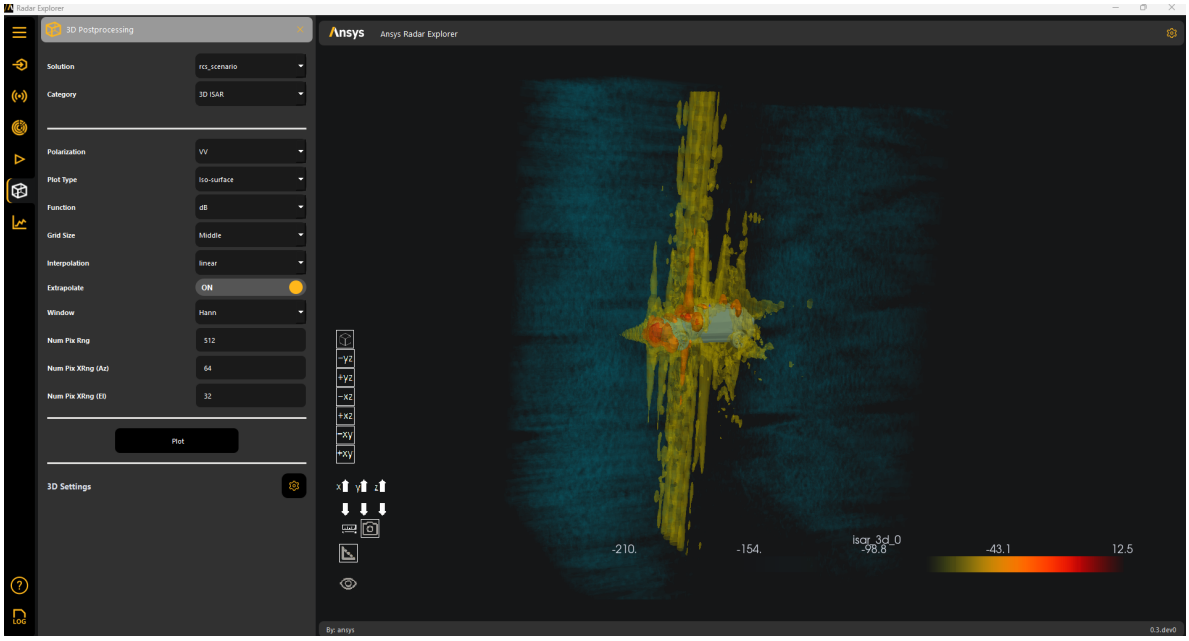


2.14 ISAR 3D

1. Go to the **3D Postprocessing** panel from the left sidebar. In the following image, **3D ISAR** is the selected postprocessing category.


Radar Explorer
3D Postprocessing
Solution
rcs_scenario
Category
3D ISAR
Polarization
VV
Plot Type
Iso-surface
Function
dB
Grid Size
Middle
Interpolation
linear
Extrapolate
ON
Window
Flat
Num Pix Rng
512
Num Pix XRng (Az)
64
Num Pix XRng (El)
32
Plot
3D Settings

2. Click **Plot** to generate a 3D plot based on the specified settings.



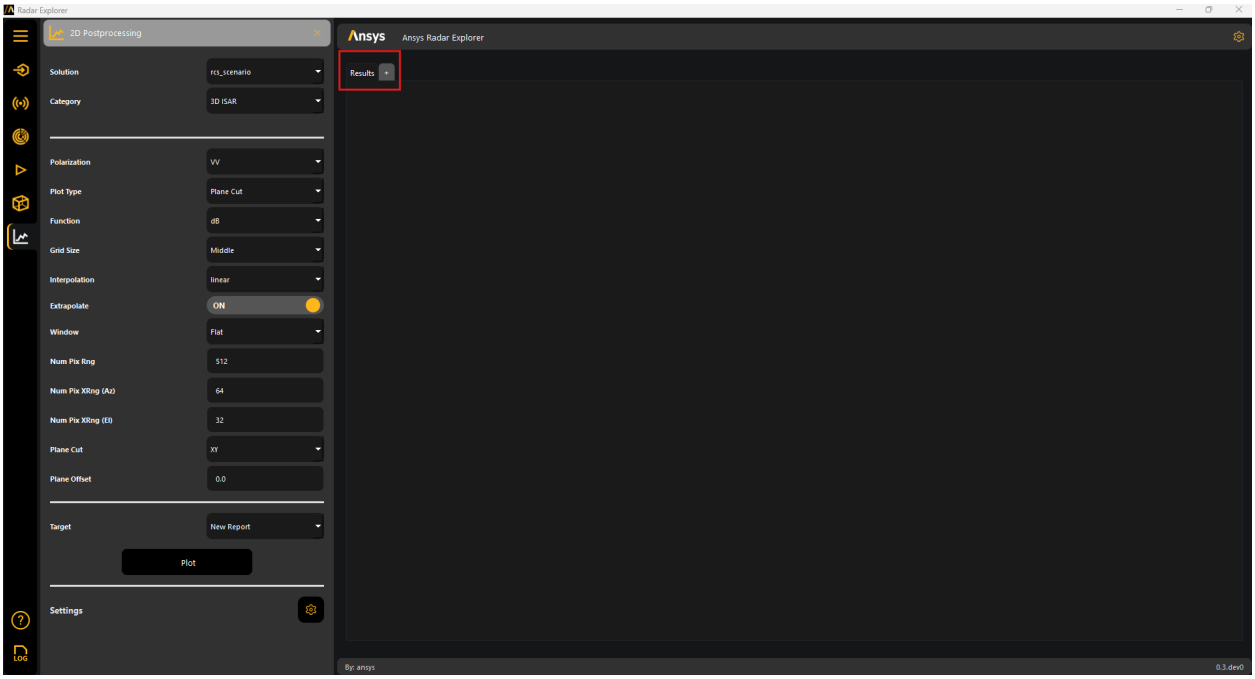
2.15 2D Postprocessing

The menu only displays the **2D Postprocessing** panel if the design is solved and results are extracted by the toolkit.

You have four 2D postprocessing categories:

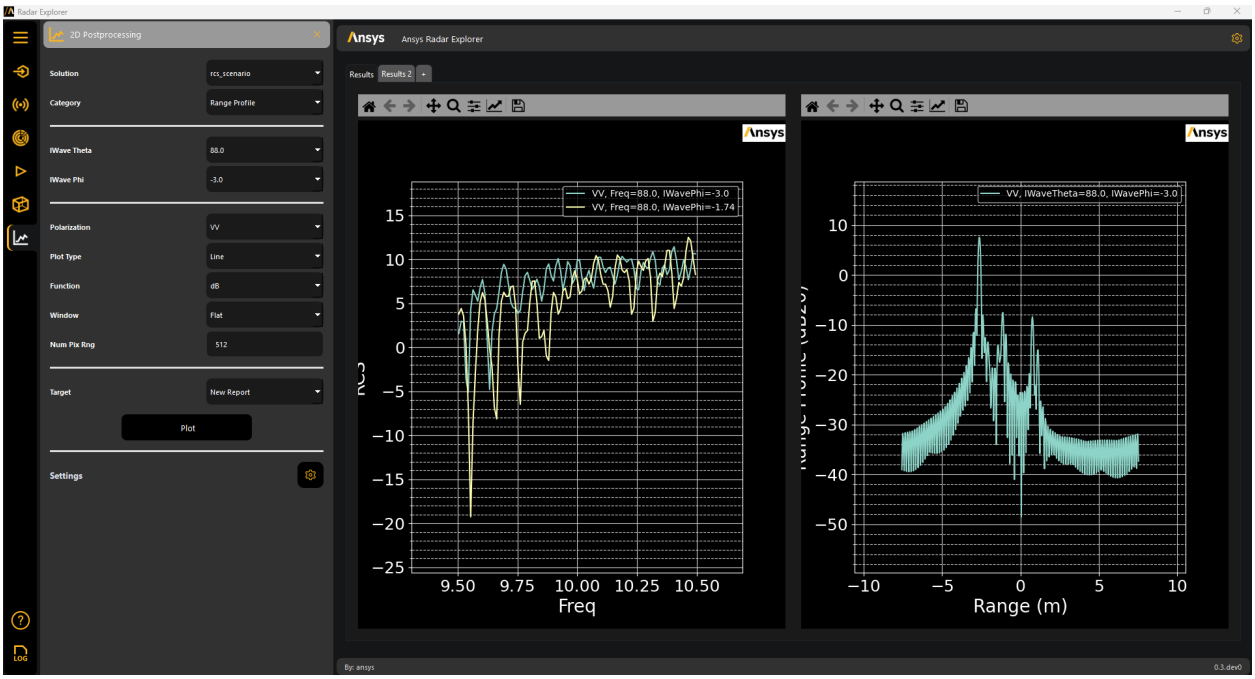
- RCS
- Range profile
- Waterfall
- ISAR 2D

The **2D Postprocessing** panel lets you plot multiple plots in one single result tab, in the same tab, or even in the same plot.



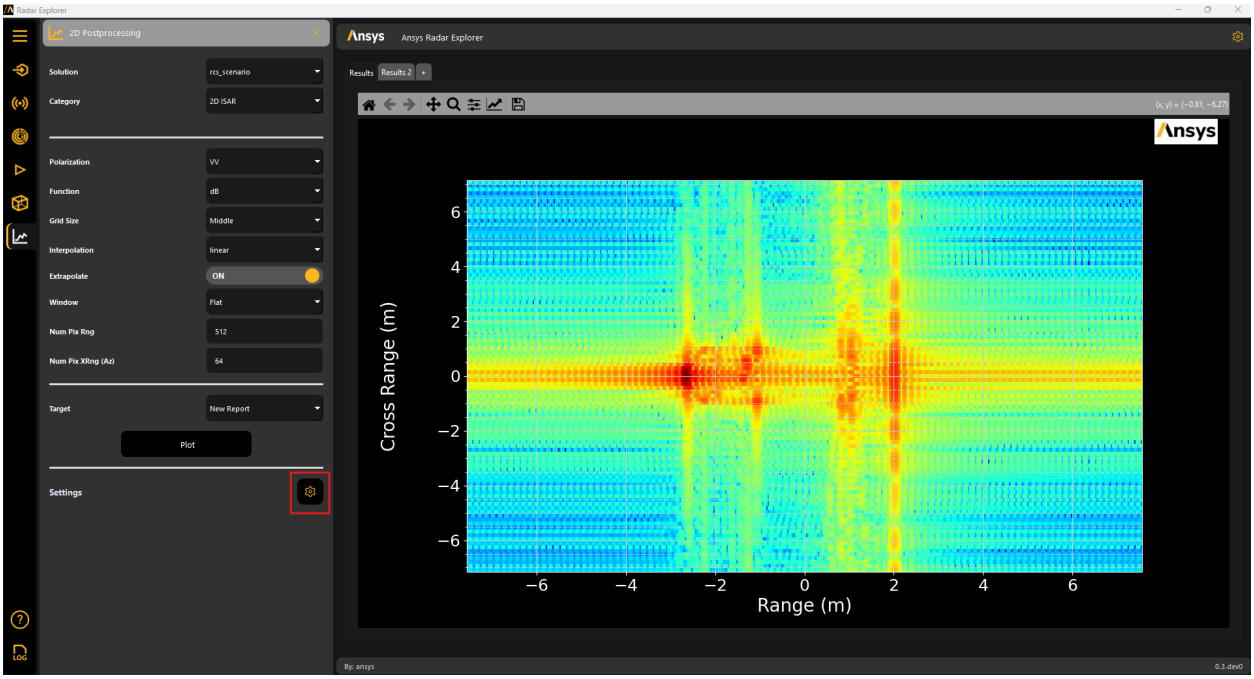
> **Note**
>
> You can access *2D settings* to change the plot settings.

Clicking **Plot** generates 2D plots based on the specified settings.

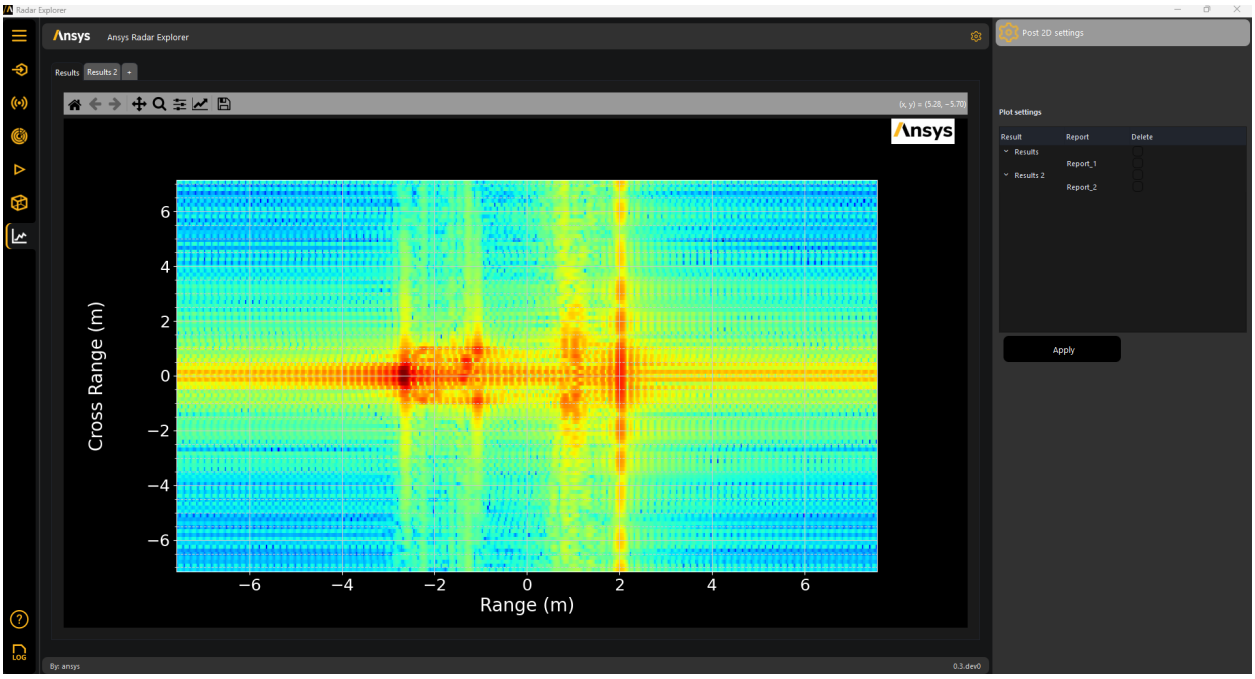


2.16 2D settings

1. Under **Settings**, click the gear icon to display the **Post 2D settings** panel for the generated plots:



1. View, modify, and delete plots if necessary.



CHAPTER

THREE

API REFERENCE

The Radar Explorer Toolkit API contains the `ToolkitBackend` class, which provides methods for controlling the toolkit workflow. This API provides methods for synthesizing and creating an antenna. You use this toolkits API at the toolkit level.

Note

The PyAEDT Common Toolkit provides the common methods for creating an AEDT session or connecting to an existing AEDT session.

You use the Radar Explorer Toolkit API to perform end-to-end workflows or directly postprocess radar results to display graphics objects and plot data.

3.1 Visualization

The Radar Explorer Toolkit API offers sophisticated tools for advanced monostatic RCS postprocessing. It contains two complementary classes: `MonostaticRCSData` and `MonostaticRCSPlotter`.

- `MonostaticRCSData`: Focuses on the direct access and processing of RCS solution data. It supports a comprehensive set of postprocessing operations, from visualizing radiation patterns to computing key performance metrics.
- `MonostaticRCSPlotter`: Focuses on the postprocessing of RCS solution data.

MonostaticRCSData	Provides monostatic RCS data.
MonostaticRCSPlotter	Provides monostatic RCS plot functionalities.

3.1.1 ansys.aedt.toolkits.radar_explorer.rcs_visualization.MonostaticRCSData

class ansys.aedt.toolkits.radar_explorer.rcs_visualization.**MonostaticRCSData**(*input_file*)

Provides monostatic RCS data.

This class reads the monostatic RCS metadata in a JSON file generated by the PyAEDT `MonostaticRCSExporter` class and returns the Python interface to plot and analyze the RCS data.

Parameters

input_file
[str] Metadata information in a JSON file.

Examples

```
>>> from ansys.aedt.core import Hfss
>>> from ansys.aedt.toolkits.radar_explorer.rcs_visualization import␣
↪MonostaticRCSData
>>> app = Hfss(version="2025.1", design="Antenna")
>>> data = app.get_rcs_data()
>>> metadata_file = data.metadata_file
>>> app.release_desktop()
>>> rcs_data = MonostaticRCSData(input_file=metadata_file)
```

__init__(*input_file*)

Methods

__init__(input_file)	
phase_shift(data, slice_idx, num_after_fft, ...)	Apply phase shift to the data if the output has an even number of points in the slice_idx direction.
window_function([window, size])	Apply a window function.

Attributes

aspect_range	Aspect range for ISAR.
available_incident_wave_phi	Available incident wave phi.
available_incident_wave_theta	Available incident wave theta.
data_conversion_function	RCS data conversion function.
extrapolate	Extrapolation flag.
frequencies	Available frequencies.
frequency	Active frequency.
frequency_units	Frequency units.
gridsize	Grid size for ISAR.
incident_wave_phi	Active incident wave phi.
incident_wave_theta	Active incident wave theta.
input_file	Input file.
interpolation	Interpolation method.
isar_2d	ISAR 2D.
isar_3d	ISAR 3D.
metadata	Antenna metadata.
name	Data name.
range_profile	Range profile.
raw_data	Antenna data.
rcs	RCS data for active frequency, theta, and phi.
rcs_active_frequency	RCS data for active frequency.
rcs_active_phi	RCS data for active incident wave phi.
rcs_active_theta	RCS data for active incident wave theta.
rcs_active_theta_phi	RCS data for active theta and phi.
solution	Data solution name.
upsample_azimuth	Upsample azimuth for ISAR.
upsample_elevation	Upsample elevation for ISAR.
upsample_range	Upsample range for ISAR.

continues on next page

Table 3 – continued from previous page

`waterfall`	Waterfall.
`window`	Window function.
`window_size`	Window size.

3.1.2 ansys.aedt.toolkits.radar_explorer.rcs_visualization.MonostaticRCSPlotter

class ansys.aedt.toolkits.radar_explorer.rcs_visualization.**MonostaticRCSPlotter**(*rcs_data:* MonostaticRCSData | *None* = *None*)

Provides monostatic RCS plot functionalities.

Parameters

rcs_data
[ansys.aedt.toolkits.radar_explorer.rcs_visualization, default: None]
Monostatic RCS data object.

Examples

```
>>> from ansys.aedt.core import Hfss
>>> from ansys.aedt.toolkits.radar_explorer.rcs_visualization import␣
↪MonostaticRCSData
>>> from ansys.aedt.toolkits.radar_explorer.rcs_visualization import␣
↪MonostaticRCSPlotter
>>> app = Hfss(version="2025.1", design="Antenna")
>>> data = app.get_rcs_data()
>>> metadata_file = data.metadata_file
>>> app.release_desktop()
>>> rcs_data = MonostaticRCSData(input_file=metadata_file)
>>> rcs_plotter = MonostaticRCSPlotter(rcs_data)
```

__init__(*rcs_data:* MonostaticRCSData | *None* = *None*)

Methods

__init__([rcs_data])	
add_incident_isar_2d_settings(phi_span, num_phi)	Add incident wave arrow setting for ISAR 2D scene.
add_incident_isar_3d_settings(theta_span, ...)	Add incident wave arrow setting for ISAR 3D scene.
add_incident_range_profile_settings([...])	Add incident wave arrow setting for range profile scene.
add_incident_rcs_settings(theta_span, ...[, ...])	Add incident wave arrow setting for RCS scene.
add_incident_waterfall_settings(phi_span, ...)	Add incident wave arrow setting for waterfall scene.
add_isar_2d([plot_type, color_bar])	Add the ISAR 2D.

continues on next page

Table 4 – continued from previous page

add_isar_2d_settings([size_range, ...])	Add a preview frame of 2D ISAR visualization to the current 3D scene.
add_isar_3d([plot_type, color_bar, ...])	Add the ISAR 3D plot to the 3D scene.
add_isar_3d_settings([size_range, ...])	Add a preview frame of 3D ISAR visualization to the current 3D scene.
add_range_profile([plot_type, color_bar])	Add the 3D range profile.
add_range_profile_settings([size_range, ...])	Add a 3D range profile setting representation to the current scene.
add_rcs([color_bar])	Add a 3D RCS representation to the current scene.
add_waterfall([color_bar])	Add the 3D waterfall.
add_waterfall_settings([aspect_ang_phi, ...])	Add a 3D waterfall setting representation to the current scene.
clear_scene([first_level, second_level, name])	Clear actors from the visualization scene.
get_plane_cut_from_3d_isar(shape, ...)	Extract a 2D plane cut from 3D ISAR data based on the specified plane.
plot_isar_2d([title, output_file, show, ...])	Create a 2D pcolor plot of the 2D ISAR image.
plot_isar_3d([plane_cut, plane_offset, ...])	Create a 2D pcolor plot of a 3D-ISAR cut.
plot_range_profile([title, output_file, ...])	Create a 2D plot of the range profile.
plot_rcs([primary_sweep, secondary_sweep, ...])	Create a 2D plot of the monostatic RCS.
plot_rcs_3d([title, output_file, show, size])	Create a 3D plot of the monostatic RCS.
plot_scene([show])	Plot the 3D scene including models, annotations, and results.
plot_waterfall([title, output_file, show, ...])	Create a 2D contour plot of the waterfall.
rotate_point(x1, y1, z1, rotation)	Rotate a set of 3D points by the specified azimuth, elevation, and twist angles.
stretch_data(data, scaling_factor, offset)	Stretches and scales the input data to a specified range.

Attributes

all_scene_actors	All scene actors.
center	Geometry center.
extents	Geometry extents.
model_info	Geometry information.
model_units	Model units.
num_contours	Number of contours.
radius	Geometry radius.
rcs_data	RCS data object.

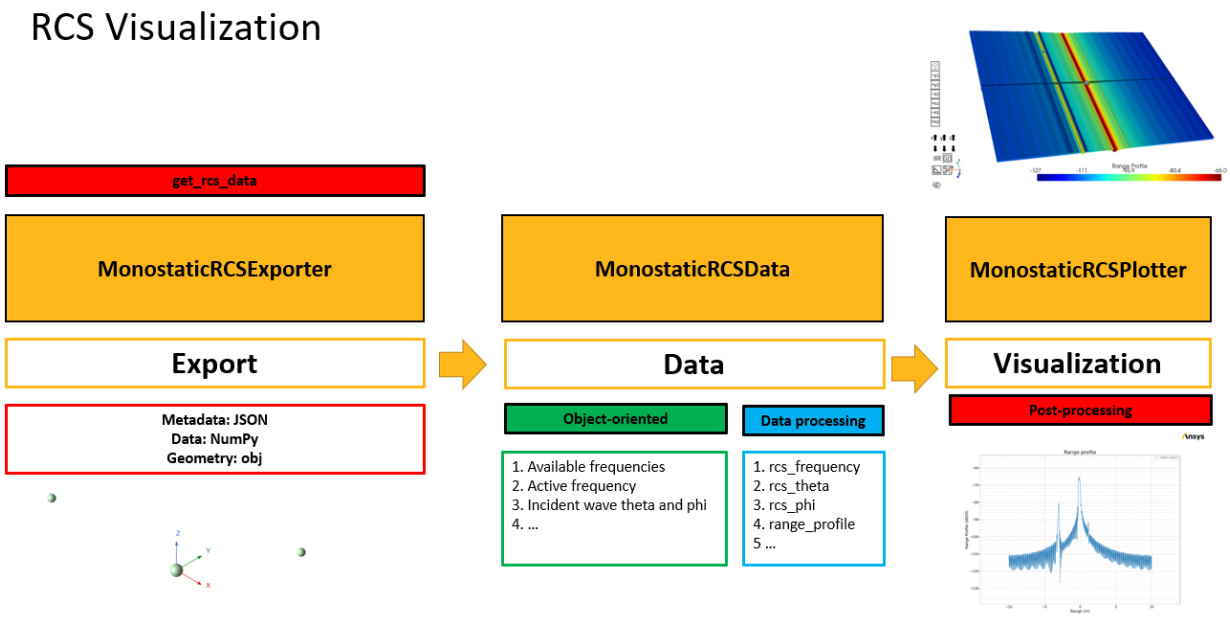
This code shows how to get RCS data and perform some postprocessing:

```
from ansys.aedt.core import Hfss
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSPlotter

app = Hfss()
rcs_object = app.get_rcs_data()
rcs_plotter = MonostaticRCSPlotter(rcs_data=rcs_object.rcs_data)
rcs_plotter.plot_rcs()
```

If you exported the RCS data previously, you can directly get this data:

```
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSPlotter
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSData

input_file = r"path_to_data\pyaedt_rcs_metadata.json"
rcs_data = MonostaticRCSData(input_file)
rcs_plotter = MonostaticRCSPlotter(rcs_data)
rcs_plotter.plot_cut()
```

The following diagram shows how both classes work. You can use them independently or from the `get_rcs_data()` method.



3.2 Workflow

To perform an end-to-end workflow, use this class:

ToolkitBackend()	API to control the toolkit workflow.

3.2.1 ansys.aedt.toolkits.radar_explorer.backend.api.ToolkitBackend

class ansys.aedt.toolkits.radar_explorer.backend.api.**ToolkitBackend**

API to control the toolkit workflow.

This class provides methods to connect to a selected design and create geometries.

Examples

```
>>> from ansys.aedt.toolkits.radar_explorer.backend.api import ToolkitBackend
>>> toolkit_api = ToolkitBackend()
>>> toolkit_api.launch_aedt()
>>> toolkit_api.wait_to_be_idle()
```

__init__()

Initialize the `toolkit` class.

Methods

__init__()	Initialize the `toolkit` class.
add_plane_wave([name, polarization])	Insert plane wave.
add_setup([name])	Insert setup.
aedt_sessions()	Get information for the active AEDT sessions.
analyze()	Analyze the design.
connect_aedt()	Connect to an existing AEDT session.
connect_design([app_name])	Connect to an application design.
duplicate_sbr_design([name])	Duplicate an existing SBR+ design.
export_aedt_model([obj_list, export_path, ...])	Export the model in the OBJ format and then encode the file if the `encode` parameter is enabled.
export_rcs([excitation, expression, encode])	Get RCS data.
generate_3d_component()	Generate a 3D component from current design.
get_design_names()	Get the design names for a specific project.
get_materials()	Get available materials.
get_plane_waves()	Get plane waves.
get_project_name(project_path)	Get the project name from the project path.
get_properties()	Get the toolkit properties.
get_setups()	Get setups.
get_sweeps()	Get sweeps.
get_thread_status()	Get the toolkit thread status.
insert_cad_sbr([name])	Insert CAD in the SBR+ design.
insert_sbr_design(input_file[, name])	Insert SBR+ design and insert a component if passed.
installed_aedt_version()	Get the installed AEDT versions.
is_aedt_connected()	Check if AEDT is connected.
is_sbr_design()	Check if the design is the SBR+ solution type.
launch_aedt()	Launch AEDT.
launch_thread(process, *args)	Launch the thread.
open_project([project_name])	Open an AEDT project.
release_aedt([close_projects, close_on_exit])	Release AEDT.
save_project([project_path, release_aedt])	Save the project.
serialize_obj_base64(file_path)	Encode a bytes-like object.
set_properties(data)	Assign the passed data to the internal data model.
update_isar_2d_properties([range_is_system, ...])	Update radar setup properties for the range profile.
update_range_profile_properties([is_system])	Update radar setup properties for range profile.
update_rcs_properties([range_is_system, ...])	Update radar setup properties for RCS.
update_waterfall_properties([...])	Update radar setup properties for waterfall.
wait_to_be_idle([timeout])	Wait for the thread to be idle and ready to accept a new task.

Here is an example of how you use it:

```
# Import required modules for the example
import time
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSData
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import (
    MonostaticRCSPlotter,
```

(continues on next page)

(continued from previous page)

```
)

# Import backend
from ansys.aedt.toolkits.template.backend.api import ToolkitBackend

# Initialize generic service
toolkit_api = ToolkitBackend()

# Load default properties from a JSON file
properties = toolkit_api.get_properties()

# Set properties
new_properties = {
    "num_phi": 101,
    "aspect_ang_phi": 80.0,
    "num_theta": 51,
    "aspect_ang_theta": 2.0,
}
flag4, msg4 = toolkit_api.set_properties(new_properties)
properties = toolkit_api.get_properties()

# Update RCS properties
toolkit_api.update_rcs_properties(
    range_is_system=True, azimuth_is_system=True, elevation_is_system=True
)

# Launch AEDT
thread_msg = toolkit_api.launch_thread(toolkit_api.launch_aedt)

# Wait until thread is finished
idle = toolkit_api.wait_to_be_idle()
if not idle:
    print("AEDT not initialized.")
    sys.exit()

# Create 3D component
component_file = toolkit_api.generate_3d_component()

# Insert SBR+ design
_ = toolkit_api.insert_sbr_design(component_file, name="Trihedral_RCS")

# Assign excitation
v_plane_wave = toolkit_api.add_plane_wave(name="IncWaveVpol", polarization="Vertical")
h_plane_wave = toolkit_api.add_plane_wave(name="IncWaveHpol", polarization="Horizontal")
plane_wave_names = [v_plane_wave, h_plane_wave]

# Create setup
setup_name = toolkit_api.add_setup()

# Analyze
toolkit_api.analyze()
toolkit_api.save_project()
```

(continues on next page)

(continued from previous page)

```
# Get RCS data

rcs_metadata_vh = toolkit_api.export_rcs(
    h_plane_wave, "ComplexMonostaticRCSTheta", encode=False
)
rcs_metadata_hh = toolkit_api.export_rcs(
    h_plane_wave, "ComplexMonostaticRCSPhi", encode=False
)

# Load RCS data

rcs_data_vv = MonostaticRCSData(metadata_file)
rcs_data_vh = MonostaticRCSData(rcs_metadata_vh)

# Load RCS Plotter

rcs_data_vv_plotter = MonostaticRCSPlotter(rcs_data_vv)
rcs_data_vh_plotter = MonostaticRCSPlotter(rcs_data_vh)

# Select cut

primary_sweep = "IWavePhi"
secondary_sweep_value_vv = rcs_data_vv_plotter.rcs_data.incident_wave_theta
secondary_sweep_value_vh = rcs_data_vh_plotter.rcs_data.incident_wave_theta

# Plot RCS

plot_vv = rcs_data_vv_plotter.plot_rcs(
    primary_sweep=primary_sweep, secondary_sweep_value=secondary_sweep_value_vv
)
plot_vh = rcs_data_vh_plotter.plot_rcs(
    primary_sweep=primary_sweep, secondary_sweep_value=secondary_sweep_value_vh
)

plot_vh_freq = rcs_data_vh_plotter.plot_rcs(
    primary_sweep="Freq", secondary_sweep="IWavePhi"
)

# Release AEDT
toolkit_api.release_aedt()
```

CHAPTER

FOUR

EXAMPLES

End-to-end examples show how to use the Radar Explorer Toolkit.

4.1 Export RCS

Duplicate an HFSS-driven modal design to an SBR+ design, apply an RCS setup, analyze, and perform postprocessing:

4.1.1 Import CAD and create ISAR 2D setup

This example demonstrates how to use the `ToolkitBackend` class. It initiates AEDT through PyAEDT, opens an SBR+ design, creates the setup, and analyzes it.

Perform required imports

[1]:
```
from pathlib import Path
import shutil
import sys
import tempfile
import time
```

[2]:
```
from ansys.aedt.toolkits.radar_explorer.backend.api import ToolkitBackend
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSData
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSPlotter
```

Set AEDT version

[3]:
```
aedt_version = "2025.1"
```

Set non-graphical mode

[4]:
```
non_graphical = True
```

Set number of cores

[5]:
```
cores = 4
```

Create temporary directory

[6]:
```
temp_dir = tempfile.TemporaryDirectory(suffix="_ansys")
```

Get example project

[7]:
```
car_original = r"example_models\geometries\car_stl.stl"
car = Path(temp_dir.name) / "car.stl"
shutil.copy(car_original, car)
```

[7]:
```
WindowsPath('C:/Users/ansys/AppData/Local/Temp/tmprwmf4sw__ansys/car.stl')
```

Initialize toolkit

[8]:
```
toolkit_api = ToolkitBackend()
```

Get toolkit properties

[9]:
```
properties_from_backend = toolkit_api.get_properties()
```

Set properties

Set non-graphical mode.

[10]:
```
set_properties = {"non_graphical": non_graphical, "aedt_version": aedt_version}
flag_set_properties, msg_set_properties = toolkit_api.set_properties(set_properties)
```

```
INFO - Updating internal properties.
```

Initialize AEDT

Launch a new AEDT session in a thread.

[11]:
```
thread_msg = toolkit_api.launch_thread(toolkit_api.launch_aedt)
```

```
PyAEDT INFO: Python version 3.12.10 (tags/v3.12.10:0cc8128, Apr  8 2025, 12:21:36) [MSC␣
↪v.1943 64 bit (AMD64)].
PyAEDT INFO: PyAEDT version 0.24.1.
PyAEDT INFO: Initializing new Desktop session.
```

Wait for the toolkit thread to be idle

Wait for the toolkit thread to be idle and ready to accept a new task.

[12]:
```
idle = toolkit_api.wait_to_be_idle()
if not idle:
    print("AEDT not initialized.")
    sys.exit()
```

```
Exception in thread Toolkit_Thread:
Traceback (most recent call last):
  File "C:\actions-runner\_work\_tool\Python\3.12.10\x64\Lib\threading.py", line 1075,␣
↪in _bootstrap_inner
    self.run()
  File "C:\actions-runner\_work\_tool\Python\3.12.10\x64\Lib\threading.py", line 1012,␣
↪in run
    self._target(*self._args, **self._kwargs)
  File "C:\actions-runner\_work\ansys-aedt-toolkits-radar-explorer\ansys-aedt-toolkits-
↪radar-explorer\.venv\Lib\site-packages\ansys\aedt\toolkits\common\backend\thread_
↪manager.py", line 55, in process_exe
    process(*args)
  File "C:\actions-runner\_work\ansys-aedt-toolkits-radar-explorer\ansys-aedt-toolkits-
↪radar-explorer\.venv\Lib\site-packages\ansys\aedt\toolkits\common\backend\api.py",␣
↪line 429, in launch_aedt
    self.desktop = ansys.aedt.core.Desktop(**desktop_args)
                   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: __init__() should return None, not 'bool'
ERROR - Toolkit has crashed and is not functional.
```

Connect design

Connect an existing design or create a new design.

[13]:
```
toolkit_api.connect_design("HFSS")
```

```
ERROR - Process ID is not defined.
```

[13]:
```
False
```

Import CAD

[14]:
```
toolkit_api.properties.cad.input_file = [car]
toolkit_api.properties.cad.material = ["pec"]
toolkit_api.properties.cad.position = [[0.0, 0.0, 0.0]]
```

Set calculation type

[15]:
```
new_properties = {"calculation_type": "2D ISAR"}
```

[16]:
```
flag3, msg3 = toolkit_api.set_properties(new_properties)
```

```
INFO - Updating internal properties.
```

[17]:
```
new_properties = {"ray_density": 0.1, "ffl": True, "num_cores": cores}
flag4, msg4 = toolkit_api.set_properties(new_properties)
```

```
INFO - Updating internal properties.
```

Set ISAR 2D properties

[18]:
```
new_properties = {
    "range_max_az": 14.3,
    "range_res_az": 0.15,
    "range_max": 15.0,
    "range_res": 0.15,
    "sim_freq_lower": 9.5e9,
    "sim_freq_upper": 10.5e9,
}
flag5, msg5 = toolkit_api.set_properties(new_properties)
```

```
INFO - Updating internal properties.
```

[19]:
```
properties1 = toolkit_api.get_properties()
```

```
C:\actions-runner\_work\ansys-aedt-toolkits-radar-explorer\ansys-aedt-toolkits-radar-
↪explorer\.venv\Lib\site-packages\pydantic\main.py:464: UserWarning: Pydantic␣
↪serializer warnings:
  PydanticSerializationUnexpectedValue(Expected `str` - serialized value may not be as␣
↪expected [field_name='input_file', input_value=WindowsPath('C:/Users/ans...rwmf4sw__
↪ansys/car.stl'), input_type=WindowsPath])
  return self.__pydantic_serializer__.to_python(
```

[20]:
```
toolkit_api.update_isar_2d_properties(range_is_system=False, azimuth_is_system=False)
```

Check how `aspect_ang_phi` and `num_phi` has changed.

[21]:
```
properties2 = toolkit_api.get_properties()
```

Connect design and load project information

[22]:
```
toolkit_api.launch_aedt()
```

```
PyAEDT INFO: Python version 3.12.10 (tags/v3.12.10:0cc8128, Apr  8 2025, 12:21:36) [MSC␣
↪v.1943 64 bit (AMD64)].
PyAEDT INFO: PyAEDT version 0.24.1.
PyAEDT INFO: Initializing new Desktop session.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[22], line 1
----> 1 toolkit_api.launch_aedt()

File C:\actions-runner\_work\ansys-aedt-toolkits-radar-explorer\ansys-aedt-toolkits-
↪radar-explorer\.venv\Lib\site-packages\ansys\aedt\toolkits\common\backend\api.py:429,␣
↪in AEDTCommon.launch_aedt(self)
    427     desktop_args["new_desktop"] = False
    428     desktop_args["aedt_process_id"] = self.properties.selected_process
--> 429 self.desktop = ansys.aedt.core.Desktop(**desktop_args)
    431 if not self.desktop:  # pragma: no cover
    432     msg = "AEDT not launched"

TypeError: __init__() should return None, not 'bool'
```

Insert CAD

[23]:
```
toolkit_api.insert_cad_sbr()
```

```
ERROR - Process ID is not defined.
ERROR - Toolkit cannot connect to AEDT.
INFO - AEDT is released.
```

[23]:
```
False
```

Assign excitation

[24]:
```
v_plane_wave = toolkit_api.add_plane_wave(name="IncWaveVpol", polarization="Vertical")
```

```
ERROR - Process ID is not defined.
ERROR - Toolkit can not connect to AEDT.
INFO - AEDT is released.
```

Create setup

[25]:
```
setup_name = toolkit_api.add_setup()
```

```
ERROR - Process ID is not defined.
ERROR - Toolkit can not connect to AEDT.
INFO - AEDT is released.
```

Get toolkit properties

[26]:
```
properties = toolkit_api.get_properties()
```

Analyze

[27]:
```
toolkit_api.analyze()
toolkit_api.save_project()
```

```
ERROR - Process ID is not defined.
ERROR - Toolkit can not connect to AEDT.
INFO - AEDT is released.
ERROR - Process ID is not defined.
ERROR - Project is not saved.
```

[27]:
```
False
```

Get RCS data

[28]:
```
rcs_metadata_vv = toolkit_api.export_rcs(v_plane_wave, "ComplexMonostaticRCSTheta",␣
↪encode=False)
```

```
ERROR - Process ID is not defined.
ERROR - Toolkit cannot connect to AEDT.
INFO - AEDT is released.
```

Save and release AEDT

[29]:
```
toolkit_api.release_aedt(True, True)
```

```
ERROR - Process ID is not defined.
INFO - AEDT is released.
```

[29]:
```
True
```

Load RCS data

[30]:
```
rcs_data_vv = MonostaticRCSData(rcs_metadata_vv)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[30], line 1
----> 1 rcs_data_vv = MonostaticRCSData(rcs_metadata_vv)

File C:\actions-runner\_work\ansys-aedt-toolkits-radar-explorer\ansys-aedt-toolkits-
↪radar-explorer\.venv\Lib\site-packages\ansys\aedt\toolkits\radar_explorer\rcs_
↪visualization.py:103, in MonostaticRCSData.__init__(self, input_file)
    102 def __init__(self, input_file):
--> 103     input_file = Path(input_file)
    104     # Public
    105     self.output_dir = input_file.parent

File C:\actions-runner\_work\_tool\Python\3.12.10\x64\Lib\pathlib.py:1162, in Path.__
↪init__(self, *args, **kwargs)
   1159     msg = ("support for supplying keyword arguments to pathlib.PurePath "
   1160            "is deprecated and scheduled for removal in Python {remove}")
   1161     warnings._deprecated("pathlib.PurePath(**kwargs)", msg, remove=(3, 14))
-> 1162 super().__init__(*args)

File C:\actions-runner\_work\_tool\Python\3.12.10\x64\Lib\pathlib.py:373, in PurePath.__
↪init__(self, *args)
    371             path = arg
    372         if not isinstance(path, str):
--> 373             raise TypeError(
    374                 "argument should be a str or an os.PathLike "
    375                 "object where __fspath__ returns a str, "
    376                 f"not {type(path).__name__!r}")
    377         paths.append(path)
    378 self._raw_paths = paths

TypeError: argument should be a str or an os.PathLike object where __fspath__ returns a␣
↪str, not 'bool'
```

Load RCS plotter

[31]:
```
rcs_data_vv_plotter = MonostaticRCSPlotter(rcs_data_vv)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[31], line 1
```

(continues on next page)

(continued from previous page)

```
----> 1 rcs_data_vv_plotter = MonostaticRCSPlotter(rcs_data_vv)

NameError: name 'rcs_data_vv' is not defined
```

Set ISAR 2D

[32]:
```
rcs_data_vv_plotter.plot_isar_2d()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[32], line 1
----> 1 rcs_data_vv_plotter.plot_isar_2d()

NameError: name 'rcs_data_vv_plotter' is not defined
```

Plot ISAR

[33]:
```
rcs_data_vv_plotter.clear_scene()
rcs_data_vv_plotter.add_isar_2d()
rcs_data_vv_plotter.plot_scene()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[33], line 1
----> 1 rcs_data_vv_plotter.clear_scene()
      2 rcs_data_vv_plotter.add_isar_2d()
      3 rcs_data_vv_plotter.plot_scene()

NameError: name 'rcs_data_vv_plotter' is not defined
```

[34]:
```
# Wait 3 seconds to allow AEDT to shut down before cleaning the temporary directory.
time.sleep(3)
```

Clean temporary directory

[35]:
```
temp_dir.cleanup()
```

4.1.2 Generate an ISAR 2D plot from RCS metadata

This example demonstrates how to use the `ToolkitBackend` class. It initiates AEDT through PyAEDT, opens an HFSS design, and proceeds to get the antenna data.

Perform required imports

[1]:
```
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSData
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSPlotter
```

Get metadata

[2]:
```
metadata_vv = r"..\example_models\isar_2d_data\HH_Spheres_ISAR2D.json"
```

Load RCS

[3]:
```
rcs_data_vv = MonostaticRCSData(metadata_vv)
```

Get ISAR 2D data

[4]:
```
rcs_data_vv.upsample_azimuth = 201
data = rcs_data_vv.isar_2d
```

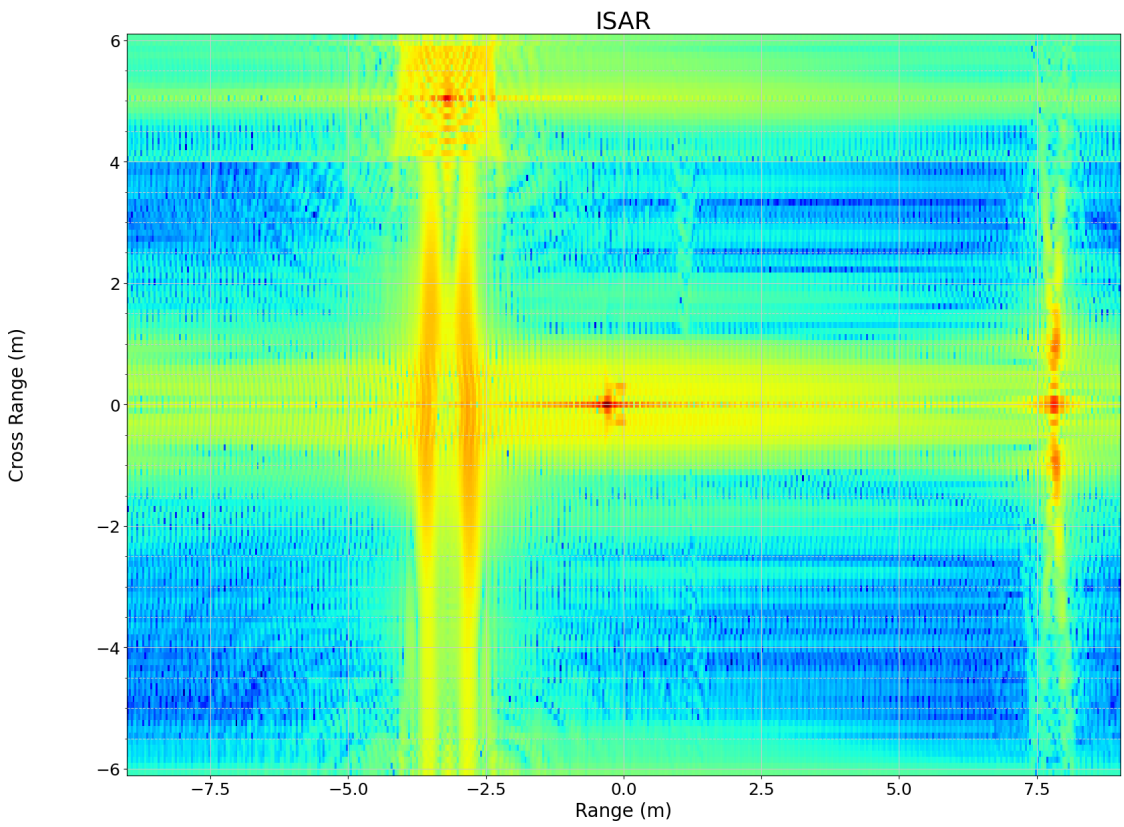
Load RCS plotter

[5]:
```
rcs_data_vv_plotter = MonostaticRCSPlotter(rcs_data_vv)
```

Plot 2D ISAR data

[6]:
```
rcs_data_vv_plotter.plot_isar_2d()
```

[6]:
```
Class: ansys.aedt.core.visualization.plot.matplotlib.ReportPlotter
```



Plot 2D ISAR settings

[7]:
```
rcs_data_vv_plotter.add_isar_2d_settings(size_range=18, size_cross_range=12)
rcs_data_vv_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot ISAR

[8]:
```
rcs_data_vv_plotter.clear_scene()
rcs_data_vv_plotter.add_isar_2d()
rcs_data_vv_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot ISAR 2D with relief

[9]:
```
rcs_data_vv_plotter.clear_scene()
rcs_data_vv_plotter.add_isar_2d(plot_type="Relief")
rcs_data_vv_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot ISAR 2D with projection

rcs_data_vv_plotter.clear_scene() rcs_data_vv_plotter.add_isar_2d() rcs_data_vv_plotter.add_isar_2d(plot_type=Projection) rcs_data_vv_plotter.plot_scene()

4.1.3 Read RCS metadata

This example demonstrates how to use the `ToolkitBackend` class. It initiates AEDT through PyAEDT, opens an HFSS design, and proceeds to get the RCS data.

Perform required imports

[1]:
```
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSData
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSPlotter
```

Get metadata

[2]:
```
metadata = r"..\example_models\rcs_data\VV_Trihedral_RCS.json"
```

Load RCS

[3]:
```
rcs_data = MonostaticRCSData(metadata)
```

Get RCS data

[4]:
```
data_active_frequency = rcs_data.rcs_active_frequency
data_active_theta = rcs_data.rcs_active_theta
data_active_phi = rcs_data.rcs_active_phi
data_active_theta_phi = rcs_data.rcs_active_theta_phi
data = rcs_data.rcs
```

Load RCS plotter

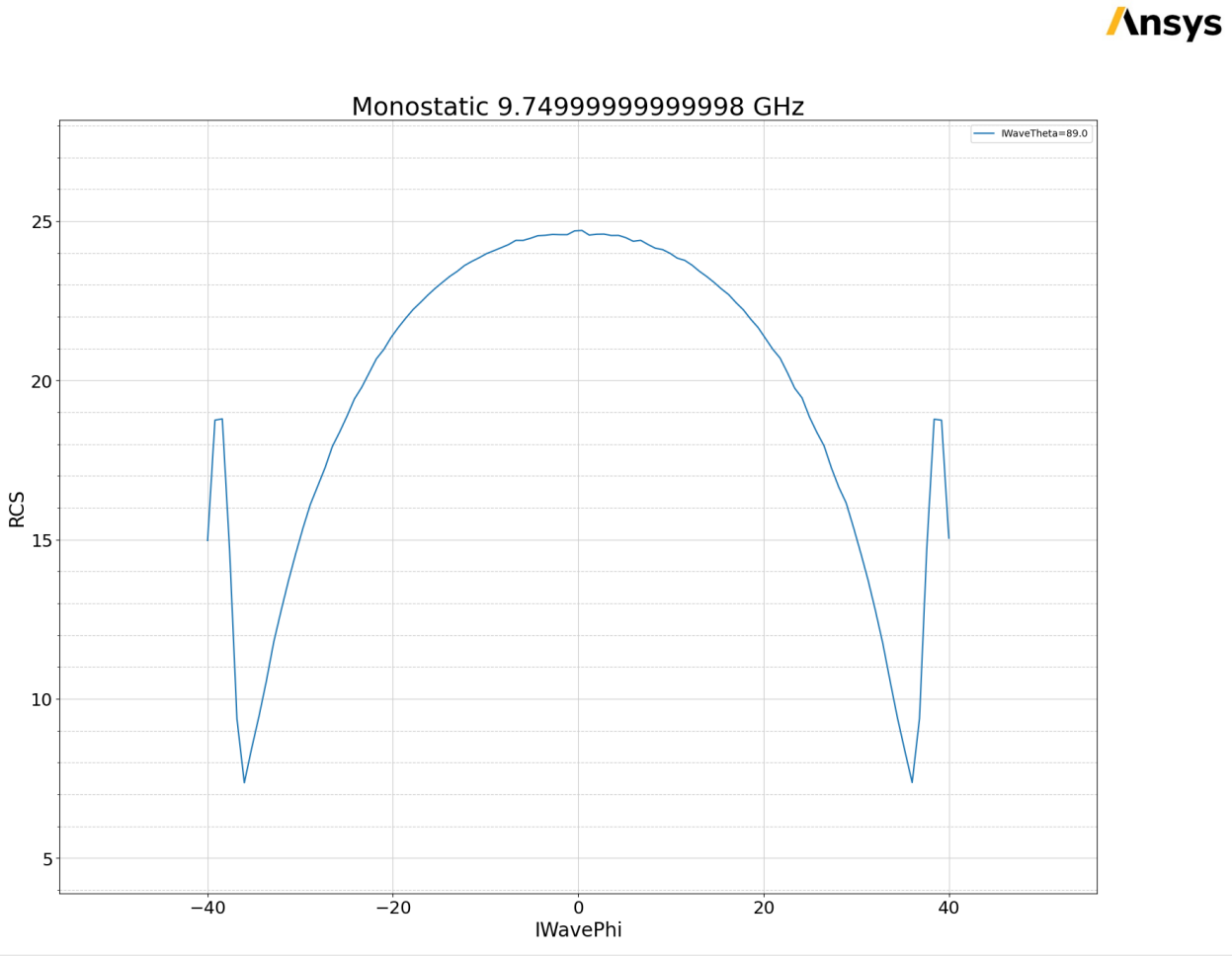
[5]:
```
rcs_data_plotter = MonostaticRCSPlotter(rcs_data)
```

Plot incident first incident wave theta

[6]:
```
primary_sweep = "IWavePhi"
secondary_sweep_value = rcs_data.available_incident_wave_theta[0]
```

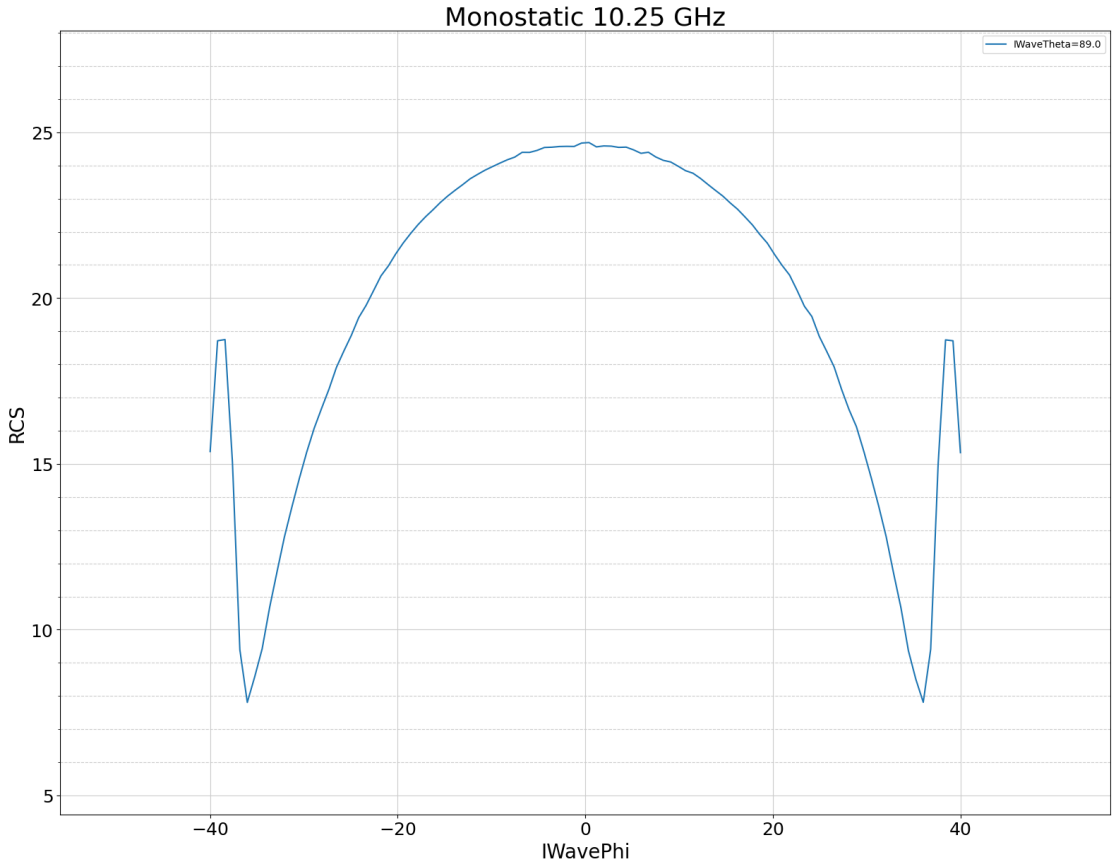
Plot RCS

[7]:
```
freq = rcs_data_plotter.rcs_data.frequency
rcs_plot_1 = rcs_data_plotter.plot_rcs(
    primary_sweep=primary_sweep, secondary_sweep_value=secondary_sweep_value, title=f
↪"Monostatic {freq} GHz", show=False
)
```



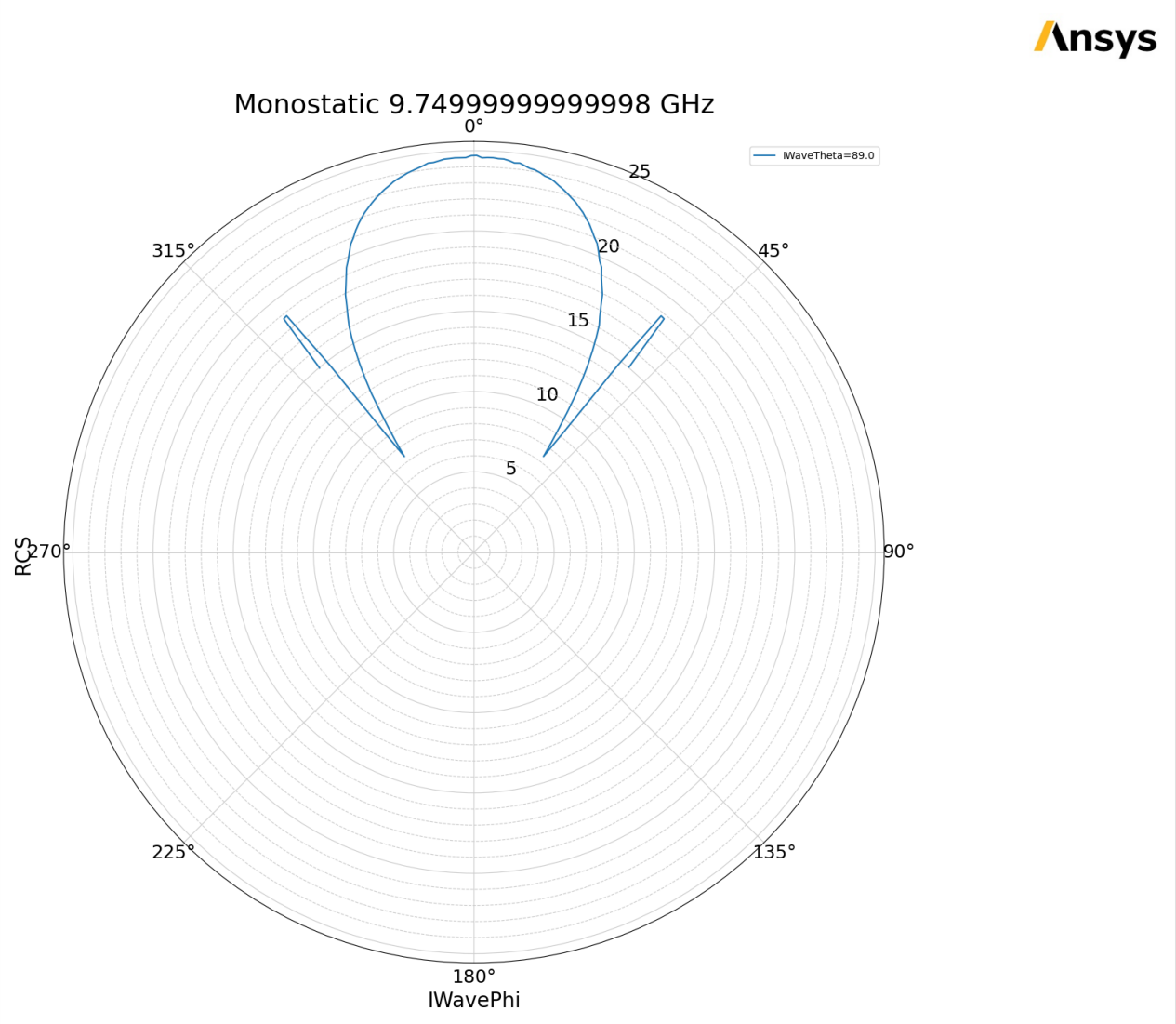
Plot last frequency RCS

[8]:
```
rcs_data_plotter.rcs_data.frequency = rcs_data_plotter.rcs_data.frequencies[-1]
freq = rcs_data_plotter.rcs_data.frequency
rcs_plot_2 = rcs_data_plotter.plot_rcs(
    primary_sweep=primary_sweep, secondary_sweep_value=secondary_sweep_value, title=f
↪"Monostatic {freq} GHz"
)
```



Plot polar RCS

[9]:
```
rcs_data_plotter.rcs_data.frequency = rcs_data_plotter.rcs_data.frequencies[0]
freq = rcs_data_plotter.rcs_data.frequency
rcs_plot_polar = rcs_data_plotter.plot_rcs(
    primary_sweep=primary_sweep,
    secondary_sweep_value=secondary_sweep_value,
    is_polar=True,
    title=f"Monostatic {freq} GHz",
)
```

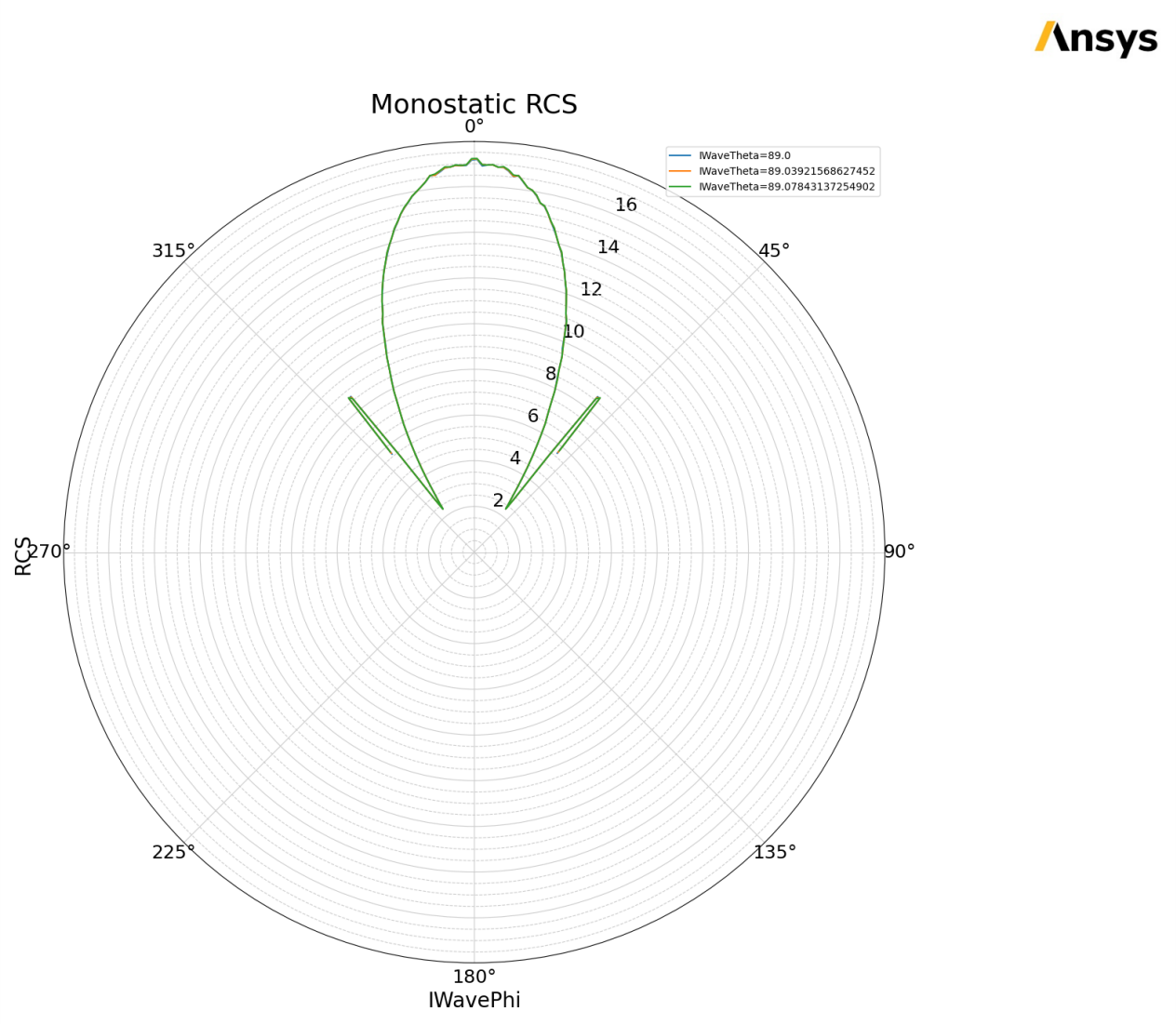


Plot three incident wave theta

[10]:
```
primary_sweep = "IWavePhi"
secondary_sweep_value = rcs_data_plotter.rcs_data.available_incident_wave_theta[0:3]
rcs_data_plotter.rcs_data.data_conversion_function = "abs"
```

Plot RCS

[11]:
```
rcs_plot_3 = rcs_data_plotter.plot_rcs(
    primary_sweep=primary_sweep, secondary_sweep_value=secondary_sweep_value, is_
↪polar=True
)
```

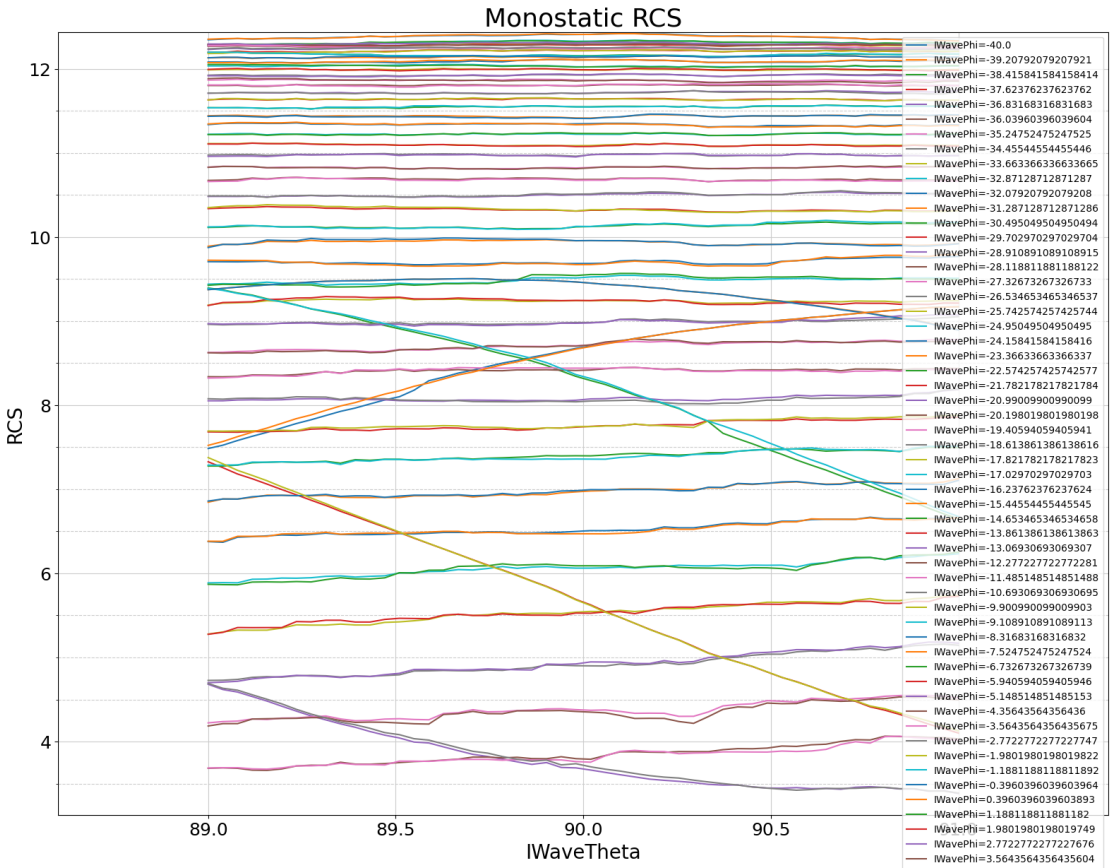


Plot all incident incident wave phi

[12]:
```
primary_sweep = "IWaveTheta"
```

Plot RCS

[13]:
```
rcs_data_plotter.rcs_data.data_conversion_function = "dB10"
rcs_plot_4 = rcs_data_plotter.plot_rcs(primary_sweep=primary_sweep, secondary_sweep_
↪value="all")
```

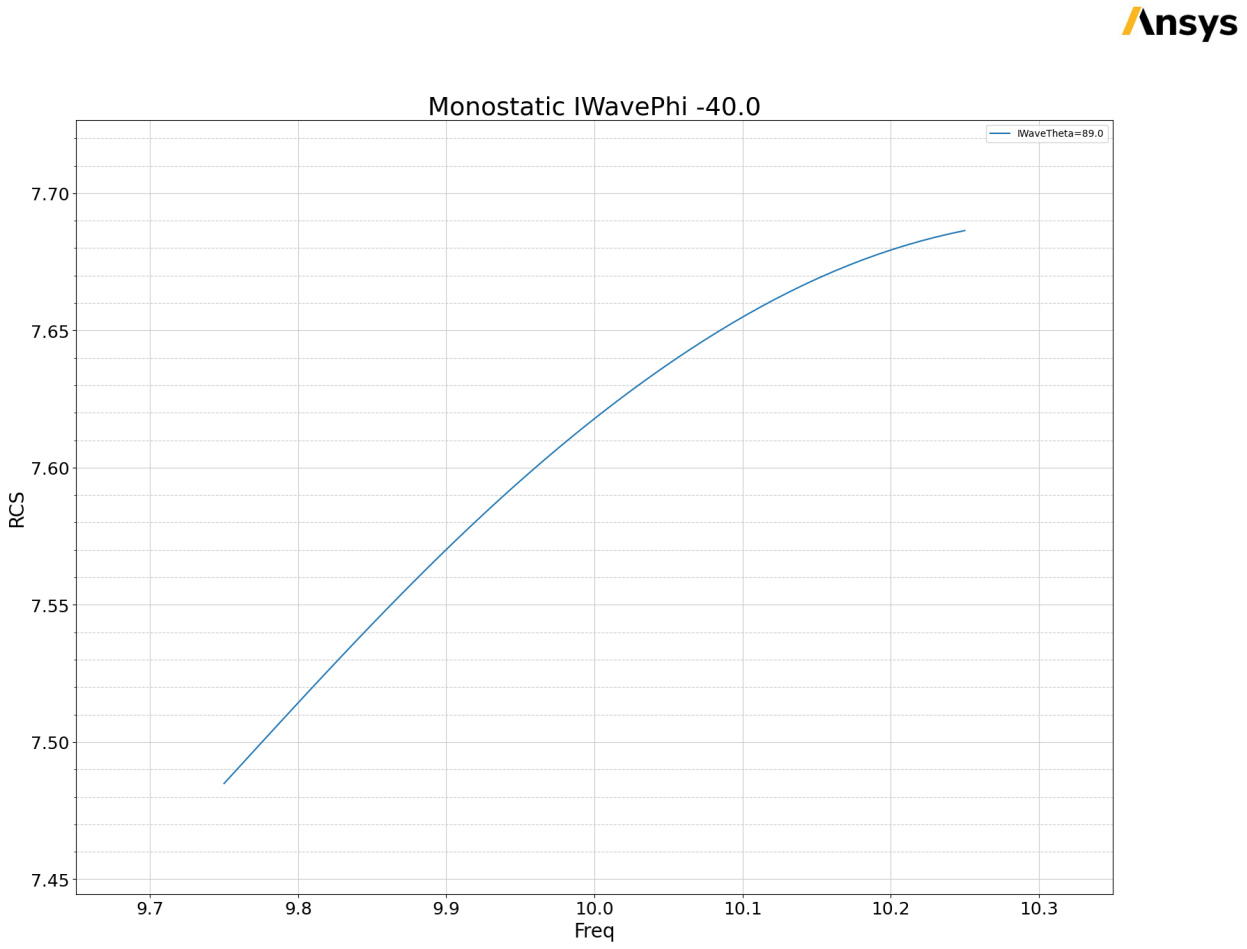

Monostatic RCS
RCS
IWaveTheta
12
10
8
6
4
89.0
89.5
90.0
90.5

Plot over frequency

[14]:
```
primary_sweep = "Freq"
```

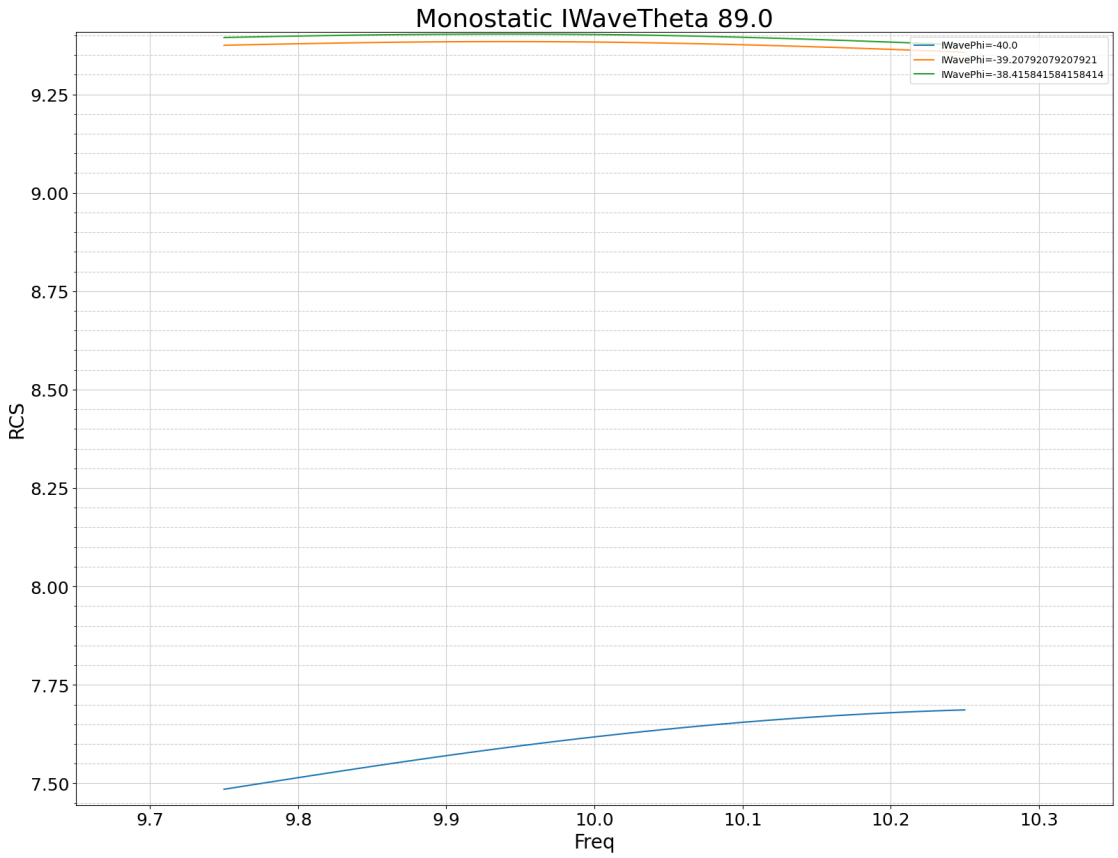
Plot RCS

[15]:
```
iwavephi = rcs_data_plotter.rcs_data.incident_wave_phi
rcs_plot_5 = rcs_data_plotter.plot_rcs(
    primary_sweep=primary_sweep,
    title=f"Monostatic IWavePhi {iwavephi}",
)
```



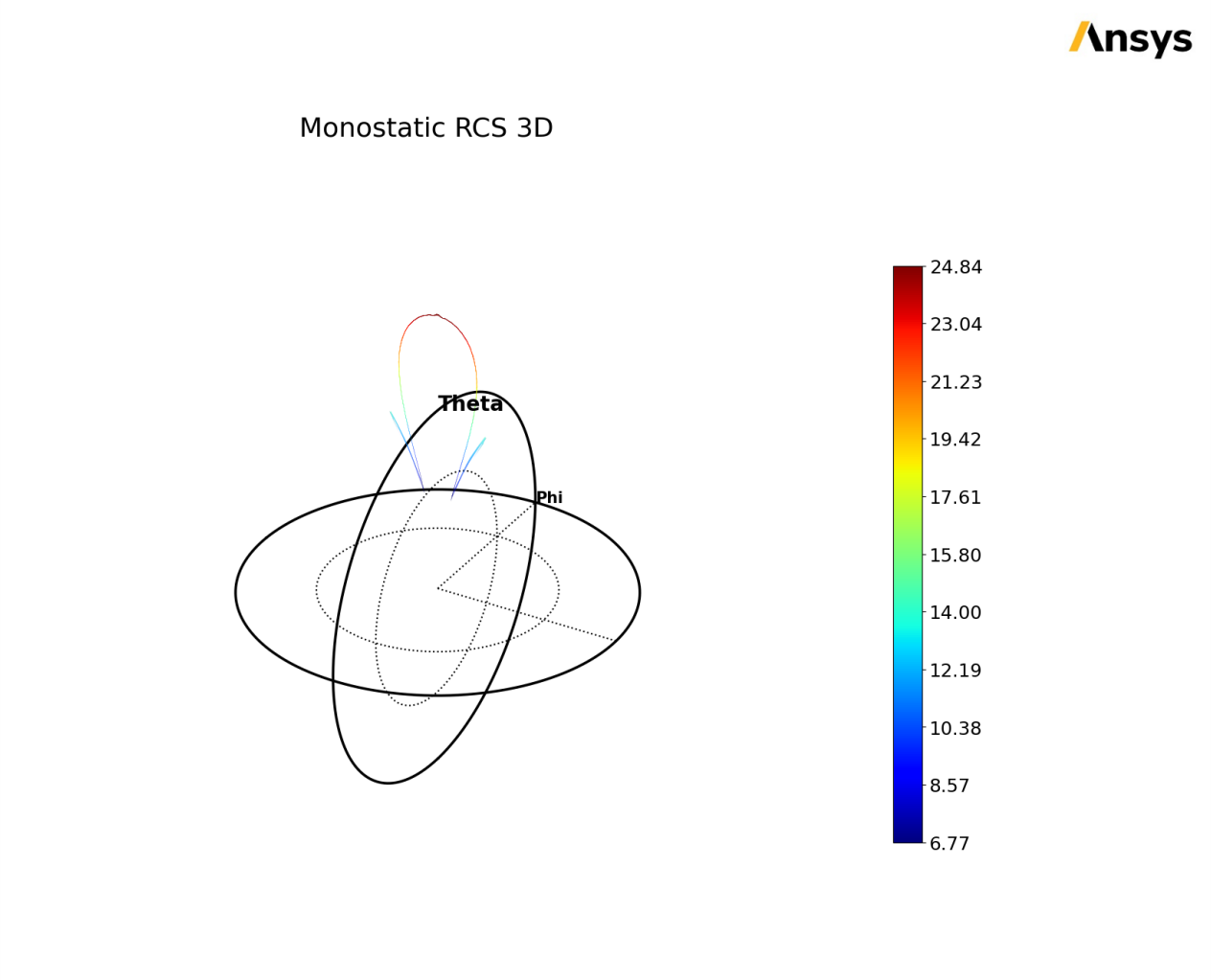
Plot RCS

[16]:
```
iwavetheta = rcs_data_plotter.rcs_data.incident_wave_theta
secondary_sweep_value = rcs_data_plotter.rcs_data.available_incident_wave_phi[0:3]
rcs_plot_6 = rcs_data_plotter.plot_rcs(
    primary_sweep=primary_sweep,
    secondary_sweep="IWavePhi",
    secondary_sweep_value=secondary_sweep_value,
    title=f"Monostatic IWaveTheta {iwavetheta}",
)
```



Plot 3D RCS with Matplotlib

[17]:
```
rcs_data_plotter.rcs_data.data_conversion_function = "dB20"
rcs_plot_7 = rcs_data_plotter.plot_rcs_3d(show=True)
```



Plot RCS in scene

[18]:
```
rcs_data_plotter.rcs_data.data_conversion_function = "dB20"
rcs_data_plotter.clear_scene()
rcs_data_plotter.show_geometry = True
rcs_data_plotter.add_rcs()
rcs_data_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

4.1.4 Generate a range profile plot from RCS metadata

This example demonstrates how to use the `ToolkitBackend` class. It initiates AEDT through PyAEDT, opens an HFSS design, and proceeds to get the antenna data.

Perform required imports

[1]:
```
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSData
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSPlotter
```

Get metadata

[2]:
```
metadata_vh = r"..\example_models\range_profile_data\VH_Spheres_Range.json"
metadata_hh = r"..\example_models\range_profile_data\HH_Spheres_Range.json"
metadata_vv = r"..\example_models\range_profile_data\VV_Spheres_Range.json"
metadata_hv = r"..\example_models\range_profile_data\HV_Spheres_Range.json"
```

Load RCS

[3]:
```
rcs_data_vh = MonostaticRCSData(metadata_vh)
rcs_data_hh = MonostaticRCSData(metadata_hh)
rcs_data_vv = MonostaticRCSData(metadata_vv)
rcs_data_hv = MonostaticRCSData(metadata_hv)
```

Get range profile

[4]:
```
data_vh = rcs_data_vh.range_profile
data_hh = rcs_data_hh.range_profile
data_vv = rcs_data_vv.range_profile
data_hv = rcs_data_hv.range_profile
```

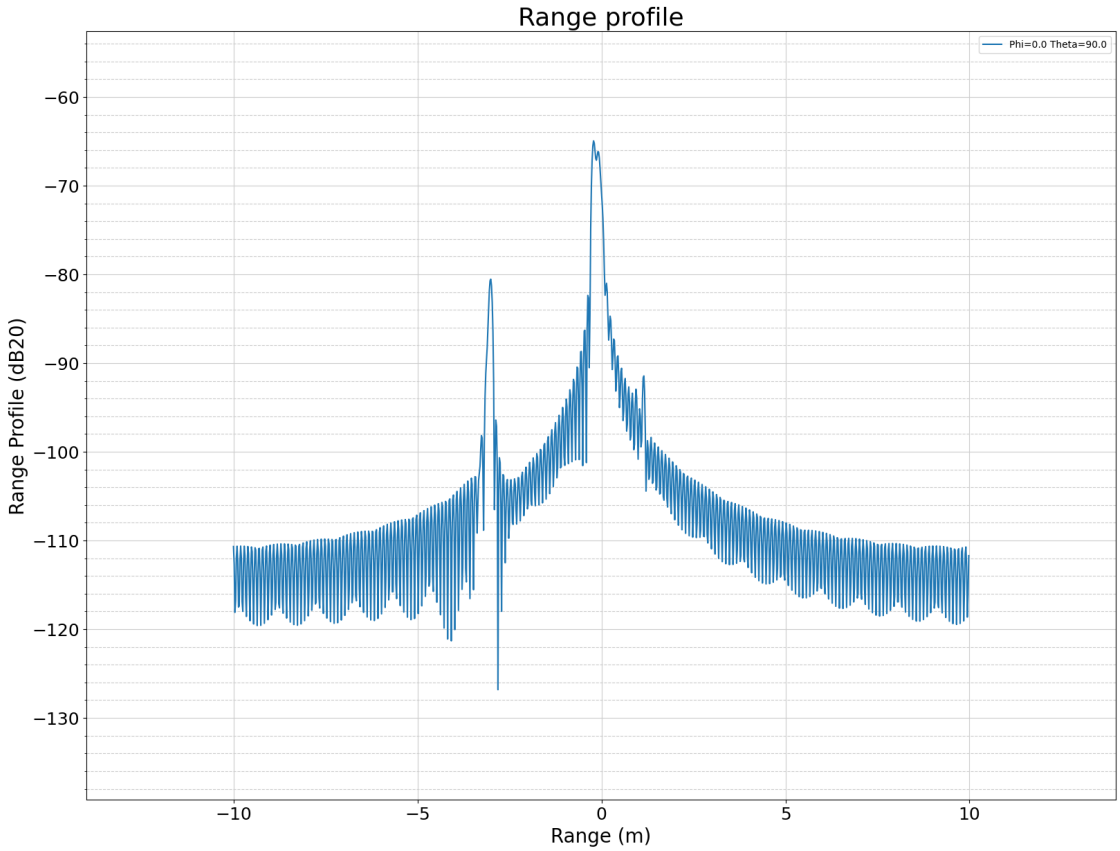
Load RCS plotter

[5]:
```
rcs_data_vh_plotter = MonostaticRCSPlotter(rcs_data_vh)
```

Plot 2D range profile

[6]:
```
rcs_data_vh_plotter.plot_range_profile()
```

[6]:
```
Class: ansys.aedt.core.visualization.plot.matplotlib.ReportPlotter
```



Plot scene with geometry

[7]:
```
rcs_data_vh_plotter.show_geometry = True
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot range profile settings using the internal Plotter

[8]:
```
rcs_data_vh_plotter.add_range_profile_settings(size_range=20, range_resolution=1)
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot range profile

[9]:
```
rcs_data_vh_plotter.show_geometry = True
rcs_data_vh_plotter.add_range_profile(color_bar="red")
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot range profile results disabling one plot

[10]:
```
for range_profile_actors in rcs_data_vh_plotter.all_scene_actors["annotations"]["range_
↪profile"].values():
    range_profile_actors.custom_object.show = False
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Clear scene and plot range profile using internal plotter

[11]:
```
rcs_data_vh_plotter.clear_scene()
rcs_data_vh_plotter.show_geometry = False
rcs_data_vh_plotter.add_range_profile()
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot some range profile results

[12]:
```
range_profile_name = list(rcs_data_vh_plotter.all_scene_actors["results"]["range_profile
↪"].keys())[0]
rcs_data_vh_plotter.clear_scene(first_level="results", second_level="range_profile",␣
↪name=range_profile_name)
rcs_data_vh_plotter.add_range_profile(plot_type="ribbon")
rcs_data_vh_plotter.add_range_profile(plot_type="Plane H")
rcs_data_vh_plotter.add_range_profile(plot_type="Plane V")
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot some range profile projection

[13]:
```
rcs_data_vh_plotter.clear_scene()
rcs_data_vh_plotter.add_range_profile(plot_type="Projection")
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot some range profile rotated

[14]:
```
rcs_data_vh_plotter.clear_scene()
rcs_data_vh_plotter.show_geometry = True
rcs_data_vh_plotter.add_range_profile(plot_type="Rotated")
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot some range profile extruded

[15]:
```
rcs_data_vh_plotter.clear_scene()
rcs_data_vh_plotter.show_geometry = True
rcs_data_vh_plotter.add_range_profile(plot_type="Extruded")
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Move plot in Z

Clean results.

[16]:
```
rcs_data_vh_plotter.clear_scene()
rcs_data_vh_plotter.show_geometry = True
```

Add range profiles.

[17]:
```
rcs_data_vh_plotter.add_range_profile(color_bar="red")
rcs_data_vh_plotter.add_range_profile(color_bar="red")
```

Get second plot and change properties.

[18]:
```
range_profile_result = rcs_data_vh_plotter.all_scene_actors["results"]["range_profile"][
↪"range_profile_1"]
range_profile_result.custom_object.color = None
range_profile_result.custom_object.color_map = "jet"
```

Move plot.

[19]:
```
range_profile_result.custom_object.z_offset = 5.0
```

Plot scene.

[20]:
```
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Scale curve

Clean results.

[21]:
```
rcs_data_vh_plotter.clear_scene()
rcs_data_vh_plotter.show_geometry = True
```

Add range profiles.

[22]:
```
rcs_data_vh_plotter.add_range_profile()
```

Get second plot and change properties

[23]:
```
range_profile_result = rcs_data_vh_plotter.all_scene_actors["results"]["range_profile"][
↪"range_profile_0"]
range_profile_result.custom_object.color = None
range_profile_result.custom_object.color_map = "jet"
```

Move plot.

[24]:
```
range_profile_result.custom_object.z_offset = 5.0
```

Scale plot.

[25]:
```
range_profile_result.custom_object.scale_factor = 3.0
```

Plot scene.

[26]:
```
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Move plot.

[27]:
```
range_profile_result.custom_object.z_offset = 1.0
```

Plot scene

[28]:
```
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Reset plot.

[29]:
```
range_profile_result.custom_object.reset_scene()
```

Plot scene.

[30]:
```
rcs_data_vh_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

4.1.5 Generate a waterfall plot from RCS metadata

This example demonstrates how to use the `ToolkitBackend` class. It initiates AEDT through PyAEDT, opens an HFSS design, and proceeds to get the antenna data.

Perform required imports

[1]:
```
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSData
from ansys.aedt.toolkits.radar_explorer.rcs_visualization import MonostaticRCSPlotter
```

Get metadata

[2]:
```
metadata_vv = r"..\example_models\waterfall_data\VV_Spheres_Waterfall.json"
```

Load RCS

[3]:
```
rcs_data_vv = MonostaticRCSData(metadata_vv)
```

Get waterfall data.

[4]:
```
data = rcs_data_vv.waterfall
```

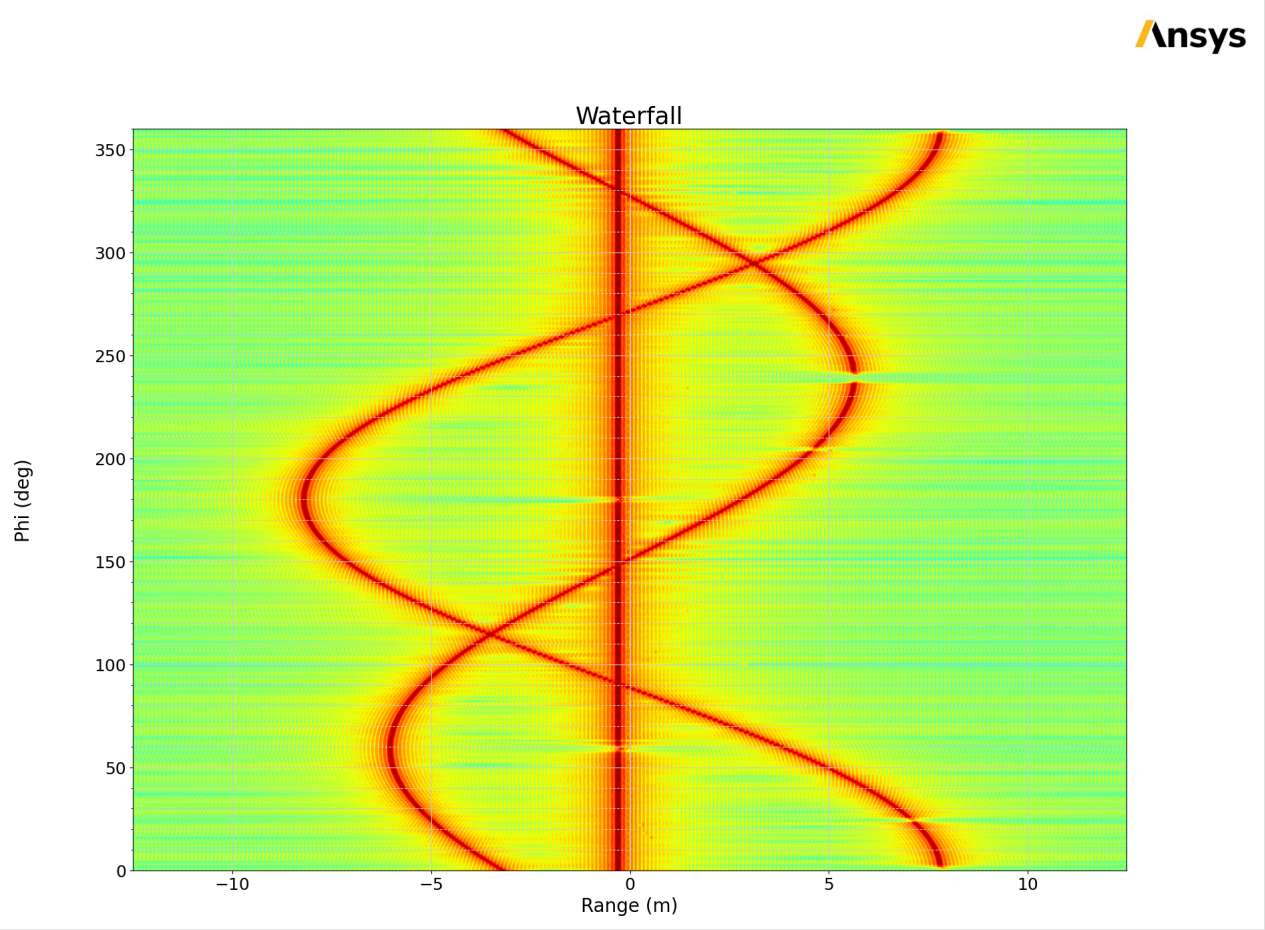
Load RCS plotter

[5]:
```
rcs_data_vv_plotter = MonostaticRCSPlotter(rcs_data_vv)
```

Plot 2D range profile

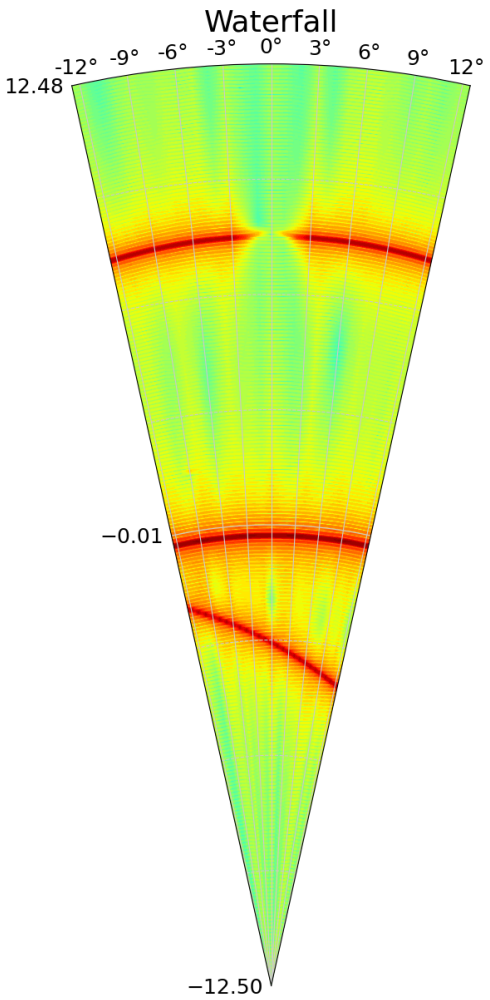
[6]:
```
rcs_data_vv_plotter.plot_waterfall()
```

[6]:
```
Class: ansys.aedt.core.visualization.plot.matplotlib.ReportPlotter
```



[7]:
```
rcs_data_vv_plotter.plot_waterfall(is_polar="True")
```

[7]:
```
Class: ansys.aedt.core.visualization.plot.matplotlib.ReportPlotter
```



Plot range profile settings using internal plotter

[8]:
```
rcs_data_vv_plotter.add_waterfall_settings()
rcs_data_vv_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot range profile results using external plotter

[9]:
```
rcs_data_vv_plotter.show_geometry = True
rcs_data_vv_plotter.add_waterfall()
rcs_data_vv_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Plot range profile results using external plotter disabling one plot

[10]:
```
for range_profile_actors in rcs_data_vv_plotter.all_scene_actors["annotations"][
↪"waterfall"].values():
    range_profile_actors.custom_object.show = False
rcs_data_vv_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

Clear scene and plot range profile using internal plotter

[11]:
```
rcs_data_vv_plotter.clear_scene()
rcs_data_vv_plotter.show_geometry = False
rcs_data_vv_plotter.add_waterfall()
rcs_data_vv_plotter.plot_scene()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

CHAPTER

FIVE

CONTRIBUTE

Overall guidance on contributing to a PyAnsys repository appears in Contributing in the *PyAnsys developers guide*. Ensure that you are thoroughly familiar with this guide before attempting to contribute to PyAEDT or its toolkits.

The following contribution information is specific to PyAEDT toolkits.

You can be up and running with four lines of code:

```
git clone https://github.com/ansys/ansys-aedt-toolkits-radar-explorer
cd ansys-aedt-toolkits-radar-explorer
pip install -e .
```

Run it with this code:

```
run_toolkit
```

5.1 Developer installation

1. Clone the repository:

   ```
   git clone https://github.com/ansys/ansys-aedt-toolkits-radar-explorer
   ```

2. Create a fresh-clean Python environment and activate it as shown in the following code. If you require additional information, see the venv documentation in the Python documentation.

   ```
   # Create a virtual environment
   python -m venv .venv
   # Activate it in a POSIX system
   source .venv/bin/activate
   # Activate it in Windows CMD environment
   .venv\Scripts\activate.bat
   # Activate it in Windows Powershell
   .venv\Scripts\Activate.ps1
   ```

3. Install the project in editable mode:

   ```
   pip install -e .[tests,doc]
   ```

4. Verify your development installation:

   ```
   pytest tests -v
   ```

5.2 Style and testing

This project uses pre-commit. Install it:

```
pip install pre-commit
run pre-commit install
```

With each commit you make, ``pre-commit``runs to ensure that you have followed project style guidelines. The output looks like this:

```
git commit -am 'fix style'
isort....................................................................Passed
black....................................................................Passed
blacken-docs.............................................................Passed
flake8...................................................................Passed
codespell................................................................Passed
pydocstyle...............................................................Passed
check for merge conflicts................................................Passed
debug statements (python)................................................Passed
check yaml...............................................................Passed
trim trailing whitespace.................................................Passed
Validate GitHub Workflows................................................Passed
```

Run this command if you need to run `pre-commit` again on all files and not just staged files:

```
pre-commit run --all-files
```

5.3 Local build

You can deploy this application as a *frozen* application using PyInstaller:

```
pip install -e .[freeze]
run pyinstaller frozen.spec
```

This generates application files at `dist/ansys_python_manager`. You can run the application locally by executing the `Ansys Python Manager.exe` file.

5.4 Documentation

For building documentation, you can run the usual rules provided in the Sphinx Makefile:

```
pip install -e .[doc]
doc/make.bat html
# subsequently open the documentation with:
<your_browser_name> doc/html/index.html
```

CHAPTER

SIX

RELEASE NOTES

This document contains the release notes for the project.

6.1 0.1.1 - February 08, 2026

Dependencies

Bump actions/setup-python from 6.0.0 to 6.1.0	#3
Bump actions/download-artifact from 6.0.0 to 7.0.0	#7
Bump actions/upload-artifact from 5.0.0 to 6.0.0	#8
Bump codecov/codecov-action from 5.5.1 to 5.5.2	#16, #18
Bump actions/checkout from 6.0.1 to 6.0.2	#19
Bump actions/setup-python from 6.1.0 to 6.2.0	#20
Bump ansys/actions from 10.2.3 to 10.2.4	#21

Fixed

Compatibility with pyaedt 0.24	#17
Update installer metadata for ansys-aedt-toolkits-radar-explorer	#24
Update installer to include PyVista and its dependencies	#25

Maintenance

Update CHANGELOG for v0.1.0	#10
Update version to ``0.2.dev0`` and modify its declaration in project	#11
Use use-pull-request-title as use-conventional-commits is deprecated.	#12
Update copyright headers to reference 2026.	#13
Clean changelog	#14
Add actions/check-vulnerabilities	#22

6.2 0.1.0 - December 22, 2025

Added

Initial commit.	#1

Dependencies

Bump actions/download-artifact from 5.0.0 to 6.0.0	#213
Bump actions/upload-artifact from 4.6.2 to 5.0.0	#215
Bump actions/checkout from 5.0.0 to 6.0.1	#219
Bump softprops/action-gh-release from 2.3.3 to 2.5.0	#220
Bump ansys/actions from 10.1.4 to 10.2.3	#221

Documentation

Fix vale warning	#210

Fixed

Workflows	#6

Maintenance

Restore pypi release step	#4
Update CHANGELOG for v0.4.0	#207
Fix permissions for doc deploy	#208
Bump 0.5.dev0	#209
Add action security checking	#211
Update license to Apache-2.0	#216
Fix condition in ``block-dependabot`` check	#222
Review permissions for jobs in workflows and provide comments where needed	#223

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- search