**Raytracing Software for the Simulation of the Solar Corona: User’s Guide**

1. **Latest News**

The latest version of SCRaytrace can be downloaded [here](https://lasco-www.nrl.navy.mil/)

1. **Overview**

Ray-tracing is a computer technique for generating an image by tracing the path of light through pixels in an image plane and simulating the effects of what it encounters with virtual objects.[[1]](#One)

The software implements:

* Thomson scattering for the simulation of the K corona[[2]](#Two)
* Mie scattering [[3]](#Three)[[4]](#Four)
* Volume scattering function [[5]](#Five)
* UV emission

The engine of the software and the models are written in C++ for processing efficiency. The software can be run using IDL command line or with a graphic user interface (GUI), which gives a quick way to visualize and fit simple CME models directly to data images.

1. **Install**
2. **Overview**

For the users using SolarSoft (<http://stereo-ssc.nascom.nasa.gov/software.shtml>), providing that one of the available binary file matches your computer architecture, you should be all set and ready to use SCRaytrace ([Section 2, “Running SCRaytrace from SolarSoft”](#section3_2)).

One caveat of using the SolarSoft precompiled version is that it only runs in single-thread mode. For people that want to speed up the ray-tracing, a multi-threaded version of the code could be used but it would need to be re-compiled. The multi-threaded version uses the boost library which has to be installed before compiling SCRaytrace ([Section 3.1, “Boost library for multi-threading”](#section3_3_1)). If you plan to modify the code, it is recommended to install the CppUnit library ([Section 3.2, “CppUnit library for unit testing”](#section3_3_2)). The C++ source code is available either from the CVS SSW repository ([Section 5, “Checking out the code from the CVS SolarSoft secchi branch repository”](#section3_5)) or as a tarball package ([Section 4, “Compiling the C++ code from the tarball package”](#section3_4)).

1. **Running SCRaytrace from SolarSoft**

For the people having SolarSoft installed (<http://stereo-ssc.nascom.nasa.gov/software.shtml>), SCRaytrace is available in the secchi branch. In order for SCRaytrace to work fully, it requires a binary file that should be compiled for your architecture. Precompiled libraries are also available in SolarSoft and should be automatically detected (see rtinitenv.pro). In order to quickly check if SCRaytrace will work on your machine you can try running

IDL> **rttest**

in IDL. If the program does not throw any error, you are all set and you can start to have some fun: [Mini Tutorial](#section4).

1. **Installing optional libraries**
   1. **Boost library for multi-threading**
      1. **Why using multi-threading?**

If you have a multi-core computer you might want to enable multi-threading since it can significantly speed up the raytracing. We measured a factor ~6 speed up from using a single core to an 8-core processor. To enable this feature, the boost library has to be installed and the C++ code has to be recompiled.

* + 1. **Installing boost**

Boost is a free peer-reviewed portable C++ source library. Package and install instructions are available at <http://www.boost.org/>. If you use Linux, you might be able to install it with the package manager that comes with your distribution.

* 1. **CppUnit library for unit testing**

CppUnit is a C++ unit testing framework. It is needed if you want to run a make check after compilation, or if you plan to contribute to the C++ code and test your code. Sources and instructions for installing that library can be found at <http://apps.sourceforge.net/mediawiki/cppunit/index.php?title=Main_Page>.

* 1. **Automake and libtool**

For people planning to modify SCRaytrace, the project uses Automake (<http://www.gnu.org/software/automake/>)and Libtool (<http://www.gnu.org/software/libtool/>). These [GNU](http://www.gnu.org/) softwares generally come standard with any Linux distribution.

* 1. **Doxygen for code documentation**

The C++ code is documented using Doxygen: <http://www.stack.nl/~dimitri/doxygen/>. It is strongly recommended that people planning to contribute to SCRaytrace install that software and also document their code.

1. **Compiling the C++ code from the tarball package**

The install follows the classical Unix/Linux install steps: **./configure**, **make**, **make install**. Again, if you want to use the multi-threading capability of the code you will have to first install the boost library: [Section 3, “Installing optional libraries”](#section3_3).

* 1. Download the tarball package: go to <http://secchi.nrl.navy.mil/wiki/pmwiki.php?n=Main.DataProcessingAndAnalysis> and scroll down to Solar Corona Ray-Tracing Software section.
  2. Expand the package and move to the created directory:

[dave@HAL]$ **tar -xzf scraytrace-0.1.tar.gz**

[dave@HAL]$ **cd scraytrace-0.1**

* 1. Configure:

[dave@HAL]$ **./configure**

* 1. Use *--help* to list the **./configure** options. If some libraries are not located in the standard Linux path, you can use the following flags: *CPPFLAGS*"=-I/non/standard/dir/include - L/non/standard/dir/libs" and *LDFLAGS*=-L/non/standard/dir/libs. As well, if you installed boost in a non standard location, you can use the *--with-boost* to specify its location.

* 1. Optionally, you can check if the compiled programs and libraries work, but this requires that you have the CPPUNIT library installed: [Section 3.2, “CppUnit library for unit testing”](#section3_3_2)

[dave@HAL]$ **make check**

* 1. The compilation creates two shared object libraries:libraytrace.so and libraytracethread.so. They are located in the src/.libs. In order for IDL to see these files, you will have to set the path using rtinitenv.pro as explained in [Section 7, “Setting the path of the compiled libraries for the IDL routines”.](#section7)

1. **Checking out the code from the CVS SolarSoft secchi branch repository**

Instructions to check out the secchi branch can be found at <http://sohowww.nascom.nasa.gov/solarsoft/stereo/secchi/doc/SECCHI_IDL_Guidelines.html>

1. **Compiling the C++ checked out of the CVS repository**

In order to be able to compile the source code coming from the SolarSoft CVS repository you will need the libtool and automake GNU softwares: see [Section 3.3, “Automake and Libtool”](file:///C:\Users\m232166\Downloads\doc\doc\install.html#automakelibtool). As well, boost and CppUnit are strongly recomended: see [Section 3, “Installing optional libraries”](file:///C:\Users\m232166\Downloads\doc\doc\install.html#install.optional)

1. Move to the scraytrace directory of the secchi branch:

[dave@HAL]$ **cd secchi/cpp/scraytrace**

1. Generate configure file:

Optionally, before the **aclocal**, you might have to update the Libtool files by making:

[dave@HAL]$ **libtoolize --copy --force**

Then proceed to:

[dave@HAL]$ **aclocal**

[dave@HAL]$ **autoconf**

[dave@HAL]$ **automake -a**

1. configure and compile

[dave@HAL]$ **./configure**

[dave@HAL]$ **make**

1. If you have [CppUnit](http://apps.sourceforge.net/mediawiki/cppunit/index.php?title=Main_Page) library installed, then you can test if the compilation was successful doing:

[dave@HAL]$ **make check**

...

OK (1)

PASS: testphysics

===================

All 11 tests passed

===================

...

[dave@HAL]$

1. Generate the code documentation:

[dave@HAL]$ **doxygen Doxyfile**

The compiled libraries libraytrace.so and libraytracethread.so are located in the subdirectory src/.libs.

1. **Setting the path of the compiled libraries for the IDL routines**
2. Set the location of the shared libraries. You can use one of the two following options:
   * Set the environment variable RT\_FORCELIBFILE to the location of the shared library libraytrace.so.

[dave@HAL]$ **setenv RT\_FORCELIBFILE /path/to/the/lib/libraytrace.so**

You can include these lines in your .tcshrc or .bashrc, depending on your shell.

If you compiled with boost you also have to set the RT\_FORCELIBTHREAD to the location of the shared library libraytracethread.so.

[dave@HAL]$ **setenv RT\_FORCELIBTHREAD /path/to/the/lib/libraytracethread.so**

Note again that both libraytrace.so and libraytracethread.so are located in the src/.libs directory of the C++ package once you have compiled the code.

* + Run IDL and use the **rtinitenv.pro** to set the shared library path

IDL> **rtinitenv, forcelibfile='/path/to/the/lib/libraytrace.so',**

**forcelibthread='/path/to/the/lib/libraytracethread.so'**

1. In order to check that everything works, run **rttest** under IDL. If the test is successful, you are all set and ready to have fun.

IDL> **rttest**

...

Test looks good !

IDL>

1. **Mini Tutorial**
2. **Using the Command Line**

The raytracing can be performed via the command line of IDL. It is of course less friendly than a graphic interface front end but it provides a way more flexibility, and simulations can be included in a batch script for example. The two main routines are **raytracewl** and **rtraytracewcs**. The following few examples show how to use these routines. Please refer to the program headers, they should be up to date with all the parameters and keyword description.

**1.1 raytracewl**

**1.1.1. No parameter passing**

**Example 4.1. Slab model simulation**

; -- simulation of a slab, edge-on

raytracewl,sbt1,sbp1,sne1,imsize=[256,256],losrange=[-30,30],$

losnbp=128L,modelid=14,neang=[0.,-39.5,-90-12]\*!dtor,/c3

wnd,0,alog10(sbt1.im > 1e-14)

; -- simulation of a slab, face-on

raytracewl,sbt2,sbp2,sne2,imsize=[256,256],losrange=[-30,30],$

losnbp=128L,modelid=14,neang=[90.,-39.5,-90-12]\*!dtor,/c3

wnd,1,alog10(sbt2.im > 1e-14)

##### **Inputs**

*sbt1*

Total brightness structures containing the image, the simulation parameters and a FITS header.

*sbp1*

Polarized brightness structures containing the image, the simulation parameters and a FITS header.

*sne1*

Integrated Ne structures containing the image, the simulation parameters and a FITS header.

##### **Outputs**

*imsize*

Size of the image in pixels.

*losrange*

LOS integration range in R\_Sun, with the origin at the impact distance. Use the *frontinteg* keyword to change the origin to the observer.

*losnbp*

Number of integration points along the LOS, within the LOS range.

*modelid*

ID of the model.

*neang*

Rotation angles applied to the density model, in radian. Rotation order is Z, Y then X. X points up in the image, Y points on the right, and Z is the optical axis. (X,Y,Z) is direct.

*/c3*

Set to the predefined LASCO C3 FOV.

* + 1. **Rotation order convention**

Rotation order convention for the variables obsang and neang is: 1st: Oz, 2nd: Oy, 3rd: Ox.

* + 1. **With parameter passing**

**Example 4.2 CME model simulation**

raytracewl,sbt1,sbp1,sne1,imsize=[256,256],losrange=[-30,30],$

losnbp=128L,modelid=54,modparam=[2.,30.\*!dtor,12.,0.2,1e6,0,0,0,0.8,0.5],neang=[50.,-39.5,-90-12]\*!dtor,/c3

wnd,0,bytscl(alog10(sbt1.im > 1e-20),-13,-9),/tv

##### **Inputs**

*modparam*

Array of parameter corresponding to the model 54. See models51to60.cpp [here](http://secchi.nrl.navy.mil/synomaps/scraytrace/doxy/index.html) for more details.

#### **1.1.4. With a Density Cube**

The model 26 is useful to do a ray-tracing through an electron density cube. The format of the *modparam* parameter array is the following (see also the source code):

1. x size (sx)
2. y size (sy)
3. z size (sz)
4. xc Sun center in pix
5. yc Sun center in pix
6. zc Sun center in pix

|  |  |
| --- | --- |
|  | **Note** |
| (0,0,0) is at the first vertice of the density cude. |

1. voxel size in rsun, same for the 3 directions of space
2. data cube in lexicographical order (x,y,z)

We can build a fake electron density cube and make the raytracing for the example. Here we use a parallelopiped slab. Note that model 26 uses trilinear interpolation between neighbor voxels. For no interpolation, use model 25.

; ---- build the fake density cube

cube=fltarr(64,64,64)

cube[32:\*,32:\*,32:38]=1e4

; ---- build the parameter array

modparam=[64,64,64,32,32,32,0.8,reform(cube,64L\*64L\*64L)]

; ---- generate the view

raytracewl,sbt,imsize=[256,256],losnbp=64L,losrange=[-20,20],$$

modelid=26,modparam=modparam,neang=[90,-80,30]\*!dtor,/cor2

wnd,0,alog10(sbt.im >1e-12 )

#### **1.1.5. With Source Surface Field Map**

For model #11, a Carrington map with the position of the neutral line have to be used. Electron density is placed depending on the distance to the neutral line and radial distance to the Sun. This can be used to reproduce the shape of the streamer belt.

; ---- download the CR2012 SSFM from WSO

rdtxtmagmap,nsheetmap,crot=2012

; ---- format parameter array in single row vector

modparam=reform(nsheetmap,n\_elements(nsheetmap))

; ---- run raytracing

raytracewl,sbt,imsize=[256,256],losnbp=64L,losrange=[-20,20],modelid=11,modparam=modparam,neang=[0,90,0.]\*!dtor,/cor2

; ---- display total brightness

wnd,0,alog10(sbt.im > 1e-14)

#### **1.1.6. Overploting Radial Distance**

In this example, we show how to plot the radial distance on simulated images. The raytracing program returns an extra parameter *rho* which is the impact distance for each LOS/pixel of the image. **oplotimpactgrid** is then used to overplot that distance on the simulated image.

raytracewl,sbt1,sbp1,sne1,imsize=[256,256],losrange=[-30,30],$$

losnbp=128L,modelid=33,modparam=[0.7,2.55,30.,10,.2],neang=[0.,-39.5,-90-12]\*!dtor,/c3,/fake,rho=rho

wnd,0,alog10(sbt1.im > 1e-14)

oplotimpactgrid,rho,levels=[1,(findgen(7)+2)\*4],$$

c\_labels=replicate(1,10),ystyle=5,xstyle=5,c\_linestyle=1

### **1.2. Using WCS information from a SECCHI data header: rtraytracewcs**

#### **1.2.1. Simulation of a CME seen in SECCHI COR2-A**

For this example, we take the SECCHI COR2-A event of 2007/05/15 23:52:30. A pre-event image taken on 2007/05/15 18:52:30 is subtracted and **secchi\_prep** is used to calibrate both the pre-event and the event images from a triplet of polarized images. The IDL routine **rtraytracewcs** takes a SECCHI FITS header in input to set the simulation parameters: detector FOV, position and attitude. In the example code shown below, we use the GCS CME model, modelid #54. The 3 parameters of the variable *neang* allow to set respectively the Carington longitude, latitude and the tilt angle of the CME. The data image and the corresponding simulated image is given in [Figure 4.1, “Simulation of a CME as seen from SECCHI COR2-A”](#figure4_1)

eventtriplet = path + ['secchi/lz/L0/a/seq/cor2/20070515/20070515\_235230\_s4c2A.fts',$

'secchi/lz/L0/a/seq/cor2/20070515/20070515\_235243\_s4c2A.fts',$

'secchi/lz/L0/a/seq/cor2/20070515/20070515\_235256\_s4c2A.fts']

preevtriplet = path + ['secchi/lz/L0/a/seq/cor2/20070515/20070515\_215230\_s4c2A.fts',$

'secchi/lz/L0/a/seq/cor2/20070515/20070515\_215243\_s4c2A.fts',$

'secchi/lz/L0/a/seq/cor2/20070515/20070515\_215256\_s4c2A.fts']

secchi\_prep,eventtriplet,hdrevent,imevent,/polariz\_on,/rotate\_on,/PRECOMMCORRECT\_ON,/rotinterp\_on,/silent

secchi\_prep,preevtriplet,hdrpreev,impreev,/polariz\_on,/rotate\_on,/PRECOMMCORRECT\_ON,/rotinterp\_on,/silent

; ---- display the data image

m=get\_smask(hdrevent)

wnd,0,alog10(m\*(imevent-impreev) > 1e-12 < 1e-10),.25

; -- get the default parameters for model 54

rtraytracewcs,sbt,sbp,sne,imsize=[512,512],modelid=54,scchead=hdrevent,losnbp=2,losrange=[-20,20],$

modparam=mp,/usedefault

; -- replace parameters with those from fit

mp[1]=0.4

mp[2]=4.

; ---- generate image of model corresponding to the event

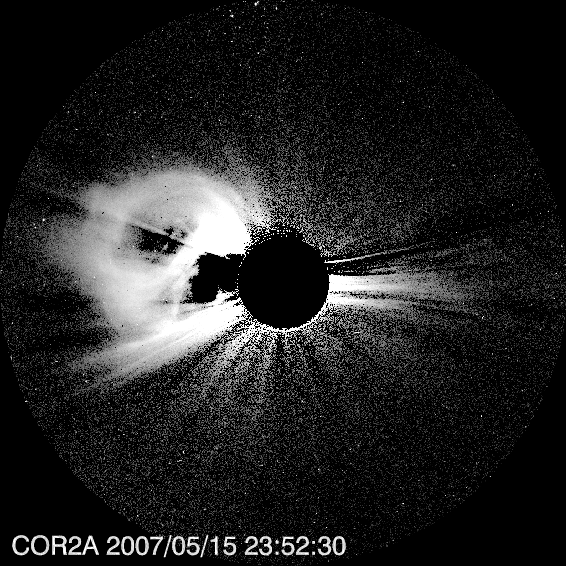
rtraytracewcs,sbt,sbp,sne,imsize=[512,512],modelid=54,scchead=hdrevent,losnbp=128,losrange=[-15,15],$

modparam=mp,neang=[80.,13.,60]\*!dtor

; ---- display the result

wnd,1,alog10(sbt.im > 1e-11)

**Figure 4.1.****Simulation of a CME as seen from SECCHI COR2-A**



#### 

#### **1.2.2. Simulation of a comet nucleus using WCS and SPICE**

In this example, we show how to position a celestial body in space and simulate how it is viewed from a given instrument. In this case, we use comet Encke, which was visible in SECCHI HI1-A images, like in 20070425\_121000\_s4h1A.fts for example.

To access the ephemeris of comet Encke, we use a SPICE kernel that can be downloaded at the NAIF web site. The code sequence needed to initialized the SPICE software, load the Encke ephemeris kernel and calculate the position of its nucleus in Carrington coordinates is shown below:

load\_stereo\_spice, errmsg=message, \_extra=\_extra

; ---- load the solar system small body ephemeris kernel

cspice\_furnsh,'/user/albert/comets/SPK/wld4826.15'

; ---- convert image data image date into SPICE compatible format

utc = anytim2utc(date, /ccsds)

; ---- define the frames

origin\_to='Sun'

frame\_from='IAU\_SUN'

correction='None'

origin\_from='Encke'

; ---- compute position

cspice\_spkezr, origin\_to, et, frame\_from,correction, origin\_from, origin, ltime

; ---- compute position in spherical coordinates

cspice\_reclat,-origin[0:2],rad,lon,lat

; ---- format output

lonlatrad=[lon,lat,rad/oneau('km')\*oneau('rsun')]

The model number 58 used here is a simple sphere of density that can positioned whereever needed, using the Carrington position of its center (here passed in the variable lonlatrad). So we simulate the position of the comet with a sphere of electron density and using Thomson scattering: this does not make sense from a physics point of view but this is just to illustrate how to position and render small features with the software.

; ---- prep the data image and get its header

secchi\_prep,'20070425\_121000\_s4h1A.fts',hdrhi1a,imhi1a,/PRECOMMCORRECT\_ON

; ---- set the parameters of the model

mp=[1.,0.5,lonlatrad]

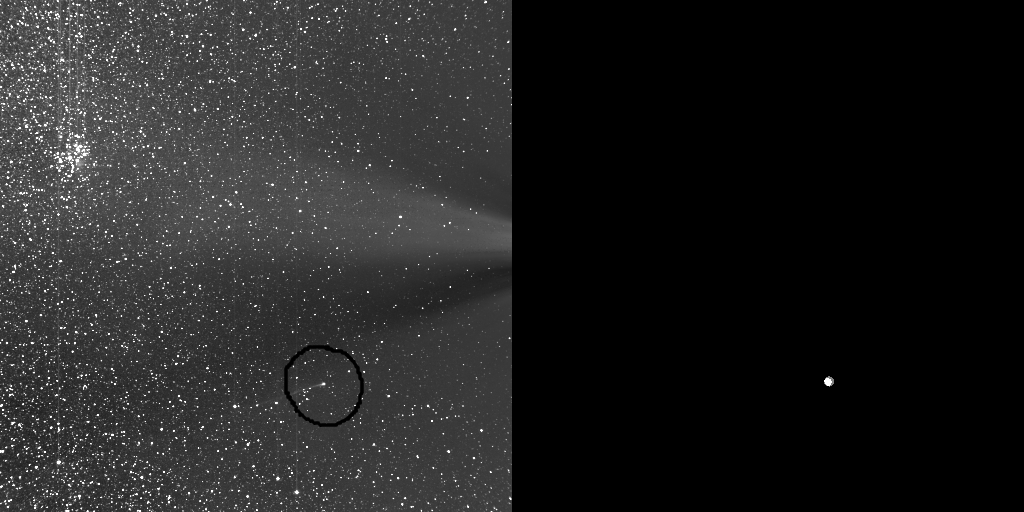
; ---- simulate comet nucleus image as seen from SECCHI HI1A

rtraytracewcs,sbt,sbp,sne,imsize=[512,512],modelid=58,scchead=hdrhi1a,losnbp=500,losrange=[-130,130],modparam=mp

; ---- display

wnd,0,sne.im

**Figure 4.2. Simulation of the comet Encke nucleus image as seen from HI1-A**

****

**1.2.3. FITS header of the simulated data**

A FITS header corresponding to the simulated image can be returned by the software. This header is (normally) compliant with the FITS-WCS standard [b2006SPD370307T]. Note that an observation date must be provided in order for the program to be able to calculate the position in different coordinate systems.

**1.2.4. Projection Type**

The following projection types are implemented: ARC, SIN, TAN, AZP. See [b2006SPD370307T] and [b2002AA3951077].

**1.2.5. Setting positions in Carrington coordinates**

It is possible to set the position of the electron density and the observer using the Carrington coordinate system. hlonlat is used to set the position of the electron density model and obslonlat is used to set the position of the observer.

**1.3. Electron density cube generation: buildcloud**

**buildcloud** generates a density cube for a given model. The cube can be saved in a text file or a binnary file. The example below shows how to build a density cube for model 14. It creates a 64 x 64 x 64 cube of 60 x 60 x 60 R\_Sun, the Sun center being at the center of the cube.

; ---- generate the density cube

buildcloud,14,cubesidenbpix=64L,cubesidersun=60.,outputtype=2

; ---- read the cube just created

rtreadbincube,'cube14.dat',c,szs,orig

; ---- raytrace it and display

raytracewl,sbt,imsize=[256,256],losnbp=64L,losrange=[-20,20],modelid=25,modparam=c,neang=[90,-80,30]\*!dtor,/c3

wnd,0,alog10(sbt.im > 1e-14)

**2. Fit of a flux rope on SECCHI data using the IDL-GUI**

We present here rtsccguicloud.pro, an IDL routine that allows to fit manually the GCS CME model to an event observed by STEREO SECCHI instruments.

**2.1. Demo mode**

The example described hereafter can be reproduced simply by using the demo mode of rtsccguicloud:

rtsccguicloud,/demo

**2.2. Preparing the data**

In this example, we will use the CME event of May 15 2007. The fist step is to prepare the images with secchi\_prep.

; ---- init the event image filenames

; -- A

eventtripa=['20070515\_235230\_s4c2A.fts','20070515\_235243\_s4c2A.fts','20070515\_235256\_s4c2A.fts']

preevtripa=['20070515\_182230\_s4c2A.fts','20070515\_182243\_s4c2A.fts','20070515\_182256\_s4c2A.fts']

; -- B

eventtripb=['20070515\_235230\_s4c2B.fts','20070515\_235243\_s4c2B.fts','20070515\_235256\_s4c2B.fts']

preevtripb=['20070515\_185230\_s4c2B.fts','20070515\_185243\_s4c2B.fts','20070515\_185256\_s4c2B.fts']

; -- EUVI images

euvia=['20070515\_235215\_n4euA.fts']

euvib=['20070515\_235215\_n4euB.fts']

; ---- prep the images

secchi\_prep,eventtripa,hdreventa,imeventa,/polariz\_on,/rotate\_on,/PRECOMMCORRECT\_ON,/rotinterp\_on,/silent

secchi\_prep,preevtripa,hdrpreeva,impreeva,/polariz\_on,/rotate\_on,/PRECOMMCORRECT\_ON,/rotinterp\_on,/silent

secchi\_prep,eventtripb,hdreventb,imeventb,/polariz\_on,/rotate\_on,/PRECOMMCORRECT\_ON,/rotinterp\_on,/silent

secchi\_prep,preevtripb,hdrpreevb,impreevb,/polariz\_on,/rotate\_on,/PRECOMMCORRECT\_ON,/rotinterp\_on,/silent

; -- get nice mask

ma=get\_smask(hdreventa)

mb=get\_smask(hdreventb)

secchi\_prep,eveuvia,heuvia,imeuvia,/PRECOMMCORRECT\_ON

secchi\_prep,eveuvib,heuvib,imeuvib,/PRECOMMCORRECT\_ON

; ---- format image for display

ima=bytscl(rebin(alog10(ma\*(imeventa-impreeva) > 1e-12 < 1e-10),512,512))

imb=bytscl(rebin(alog10(mb\*(imeventb-impreevb) > 1e-12 < 1e-10),512,512))

imea=alog10(rebin(imeuvia,512,512) > 1)

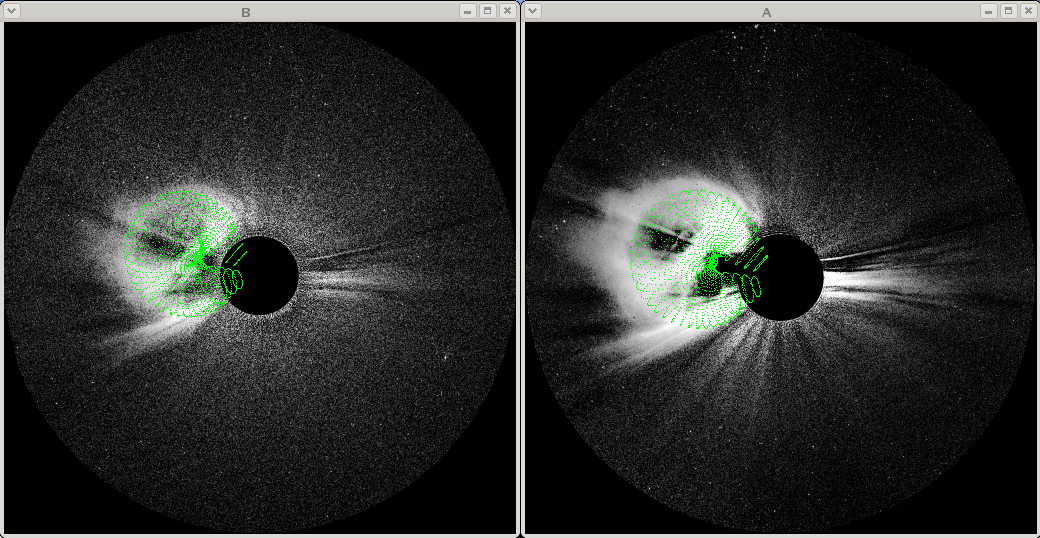
imeb=alog10(rebin(imeuvib,512,512) > 1)

; ---- run the GUI

rtsccguicloud,ima,imb,hdreventa,hdreventb,imeuvia=imea,hdreuvia=heuvia,imeuvib=imeb,hdreuvib=heuvib

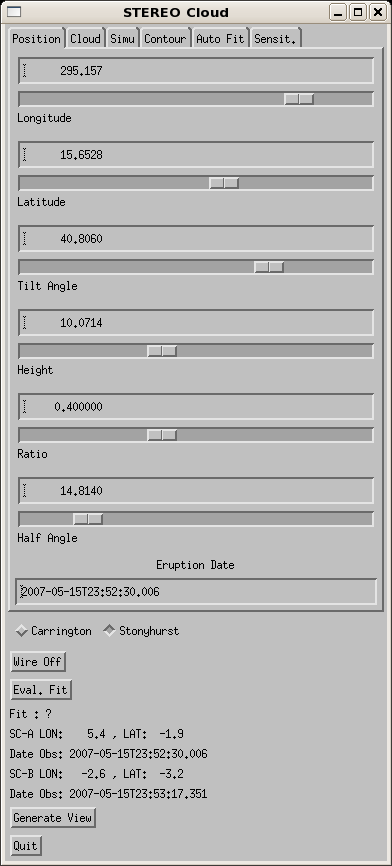
**2.3. Manual fitting of the flux rope**

**Figure 4.3. CME event of 2007/05/15 viewed in COR2-A and COR2-B.**

****

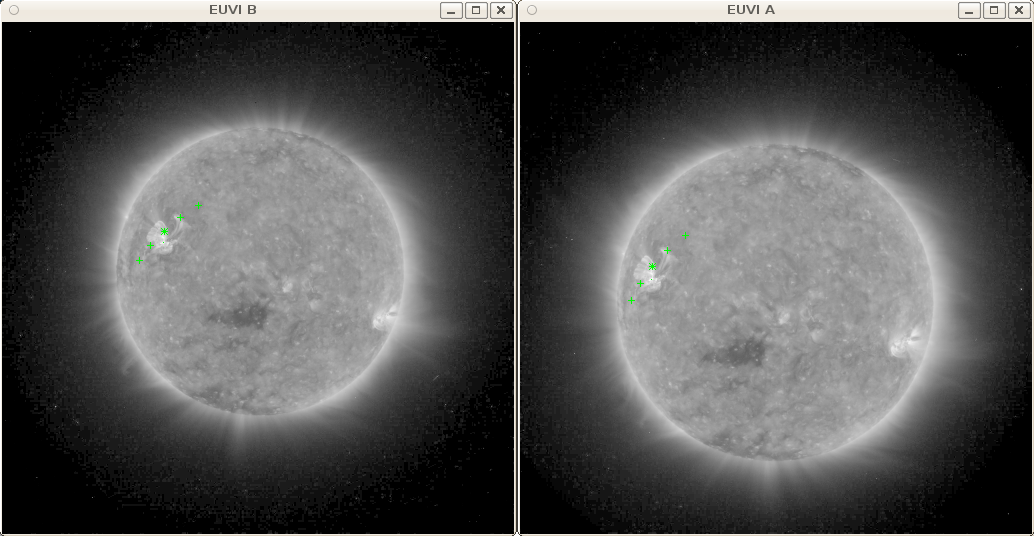
The green cloud represents the GCS CME model

**Figure 4.4. Slider GUI tab**

****

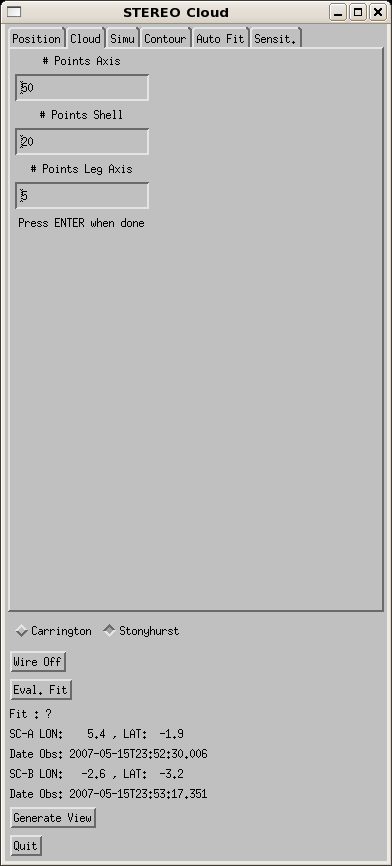
Various sliders allow to change the model parameters in order to fit the observations

**Figure 4.5. Source region position on EUVI-A and B.**



The source region position corresponding to the modeled CME is indicated by the star symbol. The plus signs show the orientation of the CME, and its angular extent. The points are plotted in green when they are on the visible side of the Sun, and in gray when they are located in the back side.

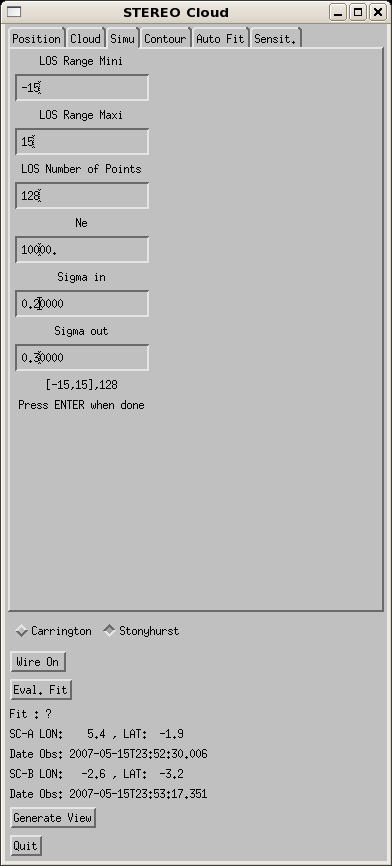
**Figure 4.6. Cloud parameters GUI tab.**



This GUI allows to change the number of points displayed in the cloud.

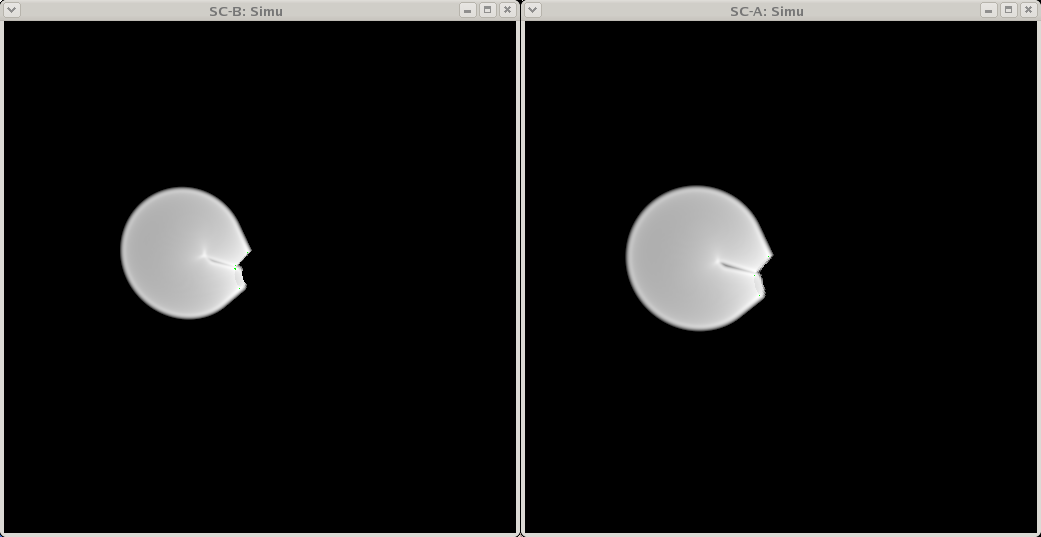
### **2.4. Generating Thomson scattering views**

**Figure 4.7. Model parameters GUI tab.**



This GUI allows to set the parameters of the Thomson scattering renderer. The integration range is in units of Rsun, and its origin is at the impact distance. The LOS number of points are the number of integration points used for each LOS in the integration range: the larger that number, the slower the rendering, but the better the image quality. Finally, the electron density, and the model shell inner and outer thickness allow to set the constant electron density in the CME skin, and the thickness of the skin.

**Figure 4.8. Simulated total brightness Thomson scattering images for COR2-A and COR2-B.**



### **2.5. Saving the parameters and images of the fit**

Different keywords of the rtsccguicloud routine allow to access the parameters of the fit, as well as the wireframe and simulated images. Here is an example of call to the program followed by a description of the different keywords.

rtsccguicloud,ima,imb,hdra,hdrb,ssim=ssim,sgui=sgui,swire=swire,ocout=oc

ssim

Structure containing the rendered images.

sgui

Structure containing all the parameters of the program, especially the GUI parameters.

swire

Structure containing the rendered wireframes, as well as all the parameters used to compute the rendered views.

ocout

Contains the 3D point positions of the wireframe.

### **2.6. Automatic fit of the model position.**

It is possible to determine semi-automaticaly the position and direction of the CME, assuming the GCS morphology of the CME. The first step consists in defining the contour of the observed CME leading edge. This is the purpose of the Contour tab of the GUI. Press the Draw Contour button in this tab and then start by drawing the contour of the CME seen in A, using the mouse. Once the contour is satisfiying, press the right button of the mouse to exit. You can then draw the contour of the CME viewed from B now. Press the right button to exit once done.

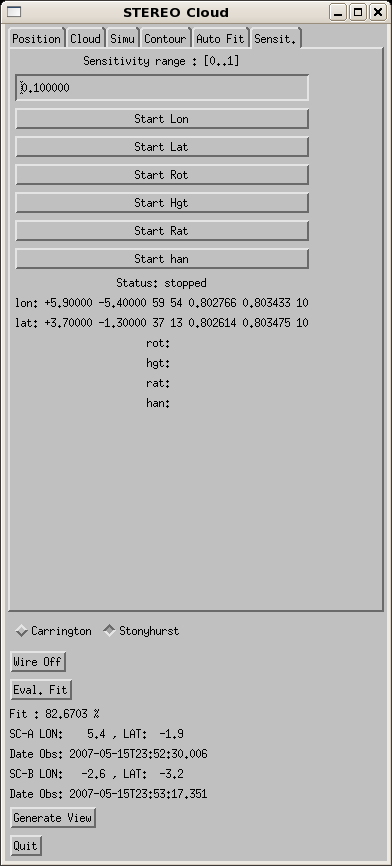
The Eval. Fit button computes the merit function that compares the model contour to the user drawn contour. If there is a perfect match, the merit function gives 100%, and if there is no match, it gives 0%. The result is displayed below the button and two windows are open showing the relative positions of the different contours.

The Auto Fit tab allows you to run the optimizer. Note that the optimizer plays only on the longitude, latitude and height of the model, the other parameters remain fixed. Indeed, the tilt angle, aspect ratio and half angle parameters are not sensitive enough (large deviation -> small change of the merit function) so the optimizer will not converge quickly enough and/or the solution will not be unique.

### **2.7. Sensitivity analysis.**

The tab Sensit. allows you to perform a sensitivity analysis on the different model parameters.

**Figure 4.9.****GUI tab used to perform sensitivity analysis.**



The edit box allows you to enter the range of variation for the analysis. In the example of [Figure 4.9, “GUI tab used to perform sensitivity analysis.”](#figure4_9), it is set to plus or minus 10%. Each following button allows you to run the analysis for each of the model parameters. In the example, it has been run for the longitude and the latitude. The outcome of the analysis is given below. The first and second displayed numbers are the deviation, in units of the corresponding parameter. In the case of the longitude and the latitude it is then in degrees. The two following numbers are the number of steps. The step is hard wired in the software, it cannot be changed, for now. For the longitude and latitude, the step is set to 0.1 degrees. The two following numbers are the value of the merit function, in each direction. It should be equal approximately to the max of the merit function minus the percentage set in the sensitivity range. Finally, the last number shows the sensitivity range, which in this case is 10%.

## **3. Software concept overview**

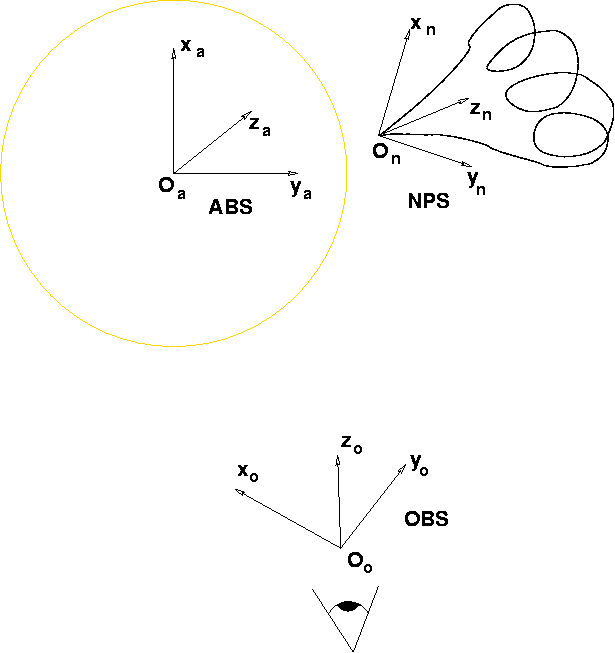
### **3.1. Units**

SCRaytrace measures the distances in Solar radii, and the angles in radians. One solar radius is taken to be 6.955x108 meters.

### **3.2. Coordinate Systems**

All the coordinate systems are direct and orthonormal. [Figure 4.10, “Overview of the three coordinate systems used in SCRaytrace. ”](#figure4_10) shows a summary of the 3 coordinate systems used in the software. ABS is the absolute coordinate system. It is placed at the center of the Sun, represented by the yellow circle here. NPS is the coordinate system of the model and OBS is the one for the observer.

**Figure 4.10.****Overview of the three coordinate systems used in SCRaytrace.**



#### **3.2.1. Absolute coordinate system: ABS**

This is the coordinate system that is the reference for all the other systems described below: translation and rotation of the other coordinate systems will be calculated with respect to this one. The origin is at the center of the Sun.

Ox

Points to Solar North.

Oy

Points to Carrington longitude and latitude (270°,0).

Oz

Points to Carrington longitude and latitude (0,0).

#### **3.2.2. Model coordinate system: NPS**

Defines the position and orientation of a model, like an electron density cube for example, with respect to the absolute coordinate system ABS.

#### **3.2.3. Observer coordinate system: OBS**

Defines the position and orientation of the observer with respect to the absolute coordinate system. We use the convention of optics for the axis orientation:

Ox

Vertical axis of the camera detector.

Oy

Horizontal axis of the camera detector.

Oz

Optical axis of the telescope, going out of the camera.

### **3.3. The Sun**

No spectrum is used and its radiance is supposed to be unity. For the Thomson scattering physics the default limb darkening coefficient is set to 0.58, but it can be overwritten by the user.

### **3.4. Observer**

The main intrinsic parameters of the observing instrument are the size in pixel of the virtual detector and the angular resolution of one pixel or plate-scale. A projection type can also be chosen. The position and the orientation in space of the instrument also have to be defined.

### **3.5. Model Representation**

The software can use two types of model representations: geometrical functions and density cubes.

#### **3.5.1. Geometric Model**

The model is defined by a geometrical function that gives the electron density (in the case of Thomson Scattering physics for example) for a given coordinate position x,y,z of space ([Equation 4.1, “Geometric Model”](#equation4_1)). The geometrical function generally will have some extra parameters, noted p, in order to be able to modify the shape and density of the model for example.

**Equation 4.1.****Geometric Model**

Ne = f(x,y,z,p)

#### **3.5.2. Density Cube**

The electron density is given by a three-dimensional density cube. Each voxel of the cube contains the value of the electron density for that region of space. The advantage of the density cube is that it can contain the representation of a complex structure which could not be simply defined by a geometrical function. On the other hand, if a higher resolution is needed, to represent the fine details of a coronal structure for example, the amount of memory necessary to store the density cube could be a problem.

### **3.6. Physical process**

Thomson scattering physics needs the electron density in electrons.cm-3. For the other types of physics, please refer to the source code documentation available at <http://secchi.nrl.navy.mil/synomaps/scraytrace/doxy/annotated.html>. The name of the classes for the various physical processes implemented are PhysicsName, with Name the name of the physics.

#### **3.6.1. Thomson Scattering**

The K corona component we observe in white light is the result of Thomson scattering of the photospheric light by the electrons of the corona [Minaert], [Van De Hulst], [Billings]. That light is polarized. The equations [Equation 4.2, “Total brightness”](#equation4_2) and [Equation 4.3, “Polarized brightness”](#equation4_3) give the total and polarized brightness scattered from the solar photosphere by a localized electron density and integrated along the lines of sight (LOS) of an observer. SCRaytrace provides an implementation of these equations.

**Equation 4.2.****Total brightness**



**Equation 4.3.****Polarized brightness**



1. **Implementing New Models**

## **1. Introduction**

The creation of a new electron density model involves editing several source files.

## **2. Implementation of a new model**

### **2.1. Adding the model code in the C++ file**

The models are in the files modelsXXtoYY.cpp, with [XX,YY] the range of model numbers. To avoid having a big file that would require a too long compilation time, the models are grouped 10 by 10. In this example we will add the model 57. The first step is to edit the file models51to60.cpp. We can add the following code:

//! Density 57: constant and uniform density

float CModel57::Density(const Cvec &v)

{

return unifdens;

}

float CModel57::Density(const Cvec &v,float &temperature)

{

temperature=uniftemp;

return unifdens;

}

//! Inititialization of the parameters for the Model 57

void CModel57::initParam(float\* pparam)

{

unifdens=pparam[0]; // - constant density, specified by the user

}

//! Returns the default parameters of the model

void CModel57::dumpDefaultParamForIDL(std::vector<moddefparam>& vp,int& flagcase)

{

flagcase=0;

vp.push\_back(moddefparam("","constant and uniform density","",""));

vp.push\_back(moddefparam("unifdens","1.","Constant density","electron/cm^3"));

return;

}

and then edit also the header file models51to60.h:

//! Constant and uniform density

class CModel57 : public CModelBase

{

public:

float Density(const Cvec &v);

float Density(const Cvec &v,float &temperature);

void initParam(float\* pparam);

void dumpDefaultParamForIDL(std::vector<moddefparam>&,int&);

protected:

float unifdens;

float uniftemp;

};

The new density model class CModel57 should derive from the virtual class CModelBase. Density is the density model method that returns the electron density depending on the point of space noted v. pparam is an array that can contain extra parameters to compute the density. A temperature can also be passed and used for the calculation: this parameter has been implemented for compatibility with a raytracing software in the radiometric spectrum range. Note that the density model is the simplest your can defined: it just returns a user defined constant density, whatever the position in space. Note also that the initParam method is useful when some preliminary calculation has to be done a single time at the initialization of the model (instantiation of the class). The results can be re-used each time the model is called, avoiding recalculating the parameters. The dumpDefaultParamForIDL method returns the default parameters for IDL: see [Section 3, “Defining the default parameters in the C code”](#section5_3) for a complete explanation.

### **2.2. Registering the model**

We need now to register the model in the modelselect method of the CModelBase class. Edit CModelBase.cc and insert the lines shown in between the /////// comment separators.

case 56 :

CModel56 \*pmodel56;

pmodel56 = new CModel56;

pmod= (CModelBase\*) pmodel56;

return pmod;

break;

///////////////////////////////////////////////////////

case 57 :

CModel57 \*pmodel57;

pmodel57 = new CModel57;

pmod= (CModelBase\*) pmodel57;

return pmod;

break;

///////////////////////////////////////////////////////

// -----------------------------------

// | REGISTER NEW DENSITIES HERE |

// -----------------------------------

default :

std::cout << "Model ID out of range: model 1 used by default." << std::endl;

pmodel1 = new CModel01;

pmod= (CModelBase\*) pmodel1;

...

We register the model classes in this modelselect method because the main raytracing program engine only knows the CModelBase class. The user selects the model by passing the model registration number (here 57) to modelselect which returns a pointer to the corresponding density model Density method. This step is just done once at the initialization and avoid that selection step each time Density is called.

### **2.3. Compile and Test**

You can now compile the C++ code using **make** and try to use the new model using the following:

IDL> **print,getdensity([4.,0,0],57,modparam=[1.])**

% Compiled module: GETDENSITY.

Seconds ellapsed :

0.011713028

1.00000

The program getdensity returns the value of the density at a point of the space, here [4.,0,0] in this case. The value returned is 1 since we passed 1 in the model parameter array *modparam*.

## **3.****Defining the default parameters in the C code**

The default parameters of a model can be set by redefining the virtual method dumpDefaultParamForIDL, derived from CModelBase class. We show here 3 examples of implementation.

### **3.1. No parameters needed**

The code in [Example 5.1, “No parameters needed.”](#example5_1) is from models01to10.cc. That model does not require any parameter. Only a description of the model is passed in the second field of structure moddefparam.

|  |  |
| --- | --- |
|  | **Important** |
| The model description is optional. If implemented, it should always be in the first element of the array vp, and the first field of structure moddefparam should be empty. |

The variable flagcase is a binary flag: LSB set to 1: the parameters are undefined: that's the default value if dumpDefaultParamForIDL is not overwritten. Bit 1 set to 1: won't be included in the frontend: can be useful if the model is obsolete or is still under construction. Bit 2 set to 1: No parameters are needed: modparam can remain undefined.

**Example 5.1.****No parameters needed.**

void CModel01::dumpDefaultParamForIDL(std::vector<moddefparam>& vp,int& flagcase) {

flagcase=0x4;

vp.push\_back(moddefparam("","M.Guhathakurta model, frozen parameters.","",""));

return;

}

### **3.2. Default parameters needed**

[Example 5.2, “Definition of default parameters needed.”](#example5_2) shows code from density model 33, defined in models31to40.cc. The first element of array vp is the model description. The rest of the rows defines the default parameters. Field 1 of moddefparam is the parameter name. Field 2 is the default value, which should always be a float or an array of float. Field 3 is a description of the parameter. Field 4 is the units of the parameters. See also moddefparam definition in the code.

**Example 5.2.****Definition of default parameters needed.**

void CModel33::dumpDefaultParamForIDL(std::vector<moddefparam>& vp,int& flagcase) {

flagcase=0;

vp.push\_back(moddefparam("","Tube shell model.","",""));

vp.push\_back(moddefparam("d0","0.7","FULL thickness of shell.","Rsun"));

vp.push\_back(moddefparam("rb","2.55","Dist to bottom of structure.","Rsun"));

vp.push\_back(moddefparam("alpha","0.52","Angle between axis and foot.","rad"));

vp.push\_back(moddefparam("rf","10","Dist junction line-circle.","Rsun"));

vp.push\_back(moddefparam("ratio","0.2","ratio of tube radius to height","Rsun"));

return;

}

### **3.3. Default parameters generated by a program**

[Example 5.3, “The default parameters are generated by a sequence of IDL instructions.”](#example5_3) if from density model 25 in models21to30.cc. As explain above, the first element of vp is the description of the density. The following elements (in [Example 5.3, “The default parameters are generated by a sequence of IDL instructions.”](#example5_3) there is only one) defines parameters and variables. Field 1 is the name of the variable. Field 2 can be either hard coded value or an IDL instruction, as in this example. Field 3 is a description of the parameter. Field 4 is not used.

The line following the parameter definition gives the name of the variable that will contain the model parameter array. Field 1 is that variable name. Field 2 must be empty in order that the parser interprets properly that line. Fields 3 and 4 are unused.

The following lines contain a sequence of IDL code that generates the model parameter array. As explained in the previous paragraph, there should be an assignment of the variable given in the line preceeding the code sequence: in [Example 5.3, “The default parameters are generated by a sequence of IDL instructions.”](#example5_3) it is mp. Parameters that has been assigned in the parameter definition section (see two paragraph above) of the vp array can be included in the code sequence. They must be preceeded by the dollar sign so it will be interpreted properly by the model default parameter array parser parsemoddefparam.pro. For example, the parameter cubefile is passed to the loaddenscube in [Example 5.3, “The default parameters are generated by a sequence of IDL instructions.”](#example5_3).

**Example 5.3.****The default parameters are generated by a sequence of IDL instructions.**

void CModel25::dumpDefaultParamForIDL(std::vector<moddefparam>& vp,int& flagcase) {

flagcase=0;

vp.push\_back(moddefparam("","Density cube.","",""));

vp.push\_back(moddefparam("cubefile","getenv('RT\_PATH')+get\_delim()+'testcube.fts'","Filename of the density cube",""));

vp.push\_back(moddefparam("mp","","",""));

vp.push\_back(moddefparam("","mp=loaddenscube($cubefile)","Load the density cube.",""));

return;

}

## **4. Getting the model default parameters from IDL**

### **4.1. The simple way**

The simplest way to assign the default parameters corresponding to a model is to use the */usedefault* keyword to the raytracewl IDL function. The following example shows the method for the model 25:

IDL> **raytracewl,modelid=25,/usedefault,modparam=mp**

The variable mp will contain the model default parameters and can be reused in another call to raytracewl.

### **4.2. The detailed way**

Building the default parameters from IDL is done using two programs: getmoddefparam.pro and parsemoddefparam.pro. The [Example 5.4, “Getting the default parameters from IDL.”](#example5_4) shows the code to build the default parameters for the model 25. The parameter vector from the C code is given in [Example 5.3, “The default parameters are generated by a sequence of IDL instructions.”](#example5_3). getmoddefparam just fetch the model parameters from the C routine and store it in the structure s. Then parsemoddefparam parse that structure s and build the model parameter array noted mparam in the example. The rest of the code [Example 5.3, “The default parameters are generated by a sequence of IDL instructions.”](#example5_3) build an image and display it.

The keyword filepro for function parsemoddefparam is optional. You can set that keyword to the filename of the IDL function that will build the model parameter array:

mparam=buildmodel25param(sv)

The output variable sv is a structure that contains the different parameters needed to build mparam. You can modify the values of that structure and reuse buildmodel25param to generate mparam with a different set of parameters. In the case of [Example 5.3, “The default parameters are generated by a sequence of IDL instructions.”](#example5_3), it would load a different density cube for example.

**Example 5.4.****Getting the default parameters from IDL.**

getmoddefparam,25,s

mparam=parsemoddefparam(s,sv,filepro='buildmodel25param.pro')

raytracewl,sbt,modparam=mparam,modelid=25,imsize=[128,128],/c2,$

neang=[30.,0,20]\*!dtor,losrange=[-10,10],losnbp=128

wnd,0,alog10(sbt.im > 1e-12)

1. **Quick Reference to Raytrace Programs**

For a description of the program parameters, please refer to the program listing header. This list is non-exhaustive.

buildcloud

Build a density cube file from a model in different formats.

getdensity

Get the density of a model at a given point of space.

loaddenscube

Load and format a fits file density cube. The output is a modparam formatted density cube.

oplotimpactgrid

Overplot impact parameter or r parameter on an image from raytracing.

raytracewl

Main raytracing program to generate an image. Does not use WCS standard. For raytracing that complies with WCS use rtraytracewcs.

rtraytracewcs

Main raytracing program to generate an image. Uses WCS standard. Useful to generate simulated images for SECCHI data.

rtdisp

To display an image from raytrace.

rtdumpversion

Simply prints the compilation time and version of the library.

rtcloud

Compute the projection of a set of points on the image plane.

rtsccguicloud

GUI useful to fit SECCHI data.

rtreadbincube

Read a binary density cube generated by buildcloud.

rttest

Runs simple test to check if SCRaytrace is properly installed.

1. **Bibliography**
2. Wikipedia contributors. (2021, June 1). Ray tracing (graphics). In *Wikipedia,*

*The Free Encyclopedia*. Retrieved 15:16, June 1, 2021

2) D. E. Billings. *A guide to the solar corona*. New York: Academic Press. 1966.

3) H. C. van de Hulst. *Light Scattering by Small Particles*. Dover. 1981.

4) C. F. Bohren and D. R. Huffman. *Absorption and Scattering of Light by Small*

*Particles*. Wiley-VCH. 2004.

5) P. L. Lamy and J.-M. Perrin. *Volume scattering function and space distribution*

*of the interplanetary dust cloud*. A and A. 269-286. 1986.

6)

7)

8)

9)