B IMPLEMENTATION OF THE GRADUATED CYLINDRICAL SHELL MODEL IN PYTHON

The graduated cylindrical shell model (GCS, Thernisien et al., 2006; Thernisien, 2011) is an empirical model that is commonly used to represent the three-dimensional structure of flux rope coronal mass ejections (CMEs) near the Sun. It defines a croissant-like 3D shape with conical legs whose ends are anchored to the center of the Sun, as shown in Figure 14.

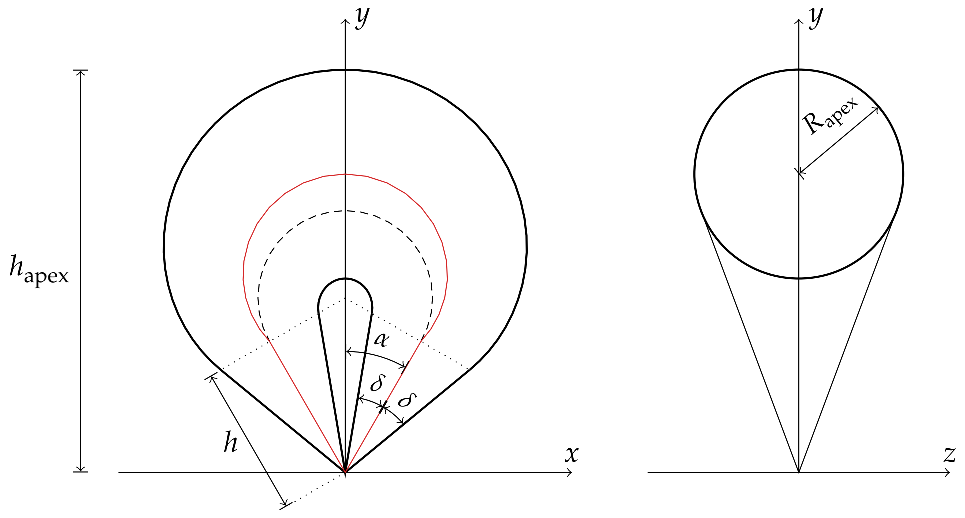


Figure 14: Illustration of the GCS model and definition of parameters $h, h_{\text{apex}}, \alpha, \delta$ and R_{apex}, based on Thernisien (2011). In this example, the parameters are set to $\alpha = 30°$ and $\kappa = 0.35$. The left panel shows a side view of the CME in the xy plane, where the thick black line marks the outer contour of the flux rope and the red line corresponds to its central axis. The dotted lines mark the boundary between the front section and the legs. The dashed line is a circular arc around the central point, showing that the front section does not have a constant radius. The right panel shows a cut in the perpendicular yz plane, where the cross section of the front is marked with a thick circle and the conical legs are indicated using the thin lines.

The GCS geometry is constrained using three main parameters: The CME apex height h_{apex} (or, alternatively, the leg height h), the angular half width α of the CME, and the so-called aspect ratio κ, which corresponds to the half angle δ of the leg cones:

$$\kappa = \sin \delta \tag{10}$$

The origin of the coordinate system shown in Figure 14 is fixed to the center of the Sun, with the y axis defining the propagation direction of the CME. Three additional parameters describe its orientation: The heliographic latitude θ and longitude ϕ (typically given in Stonyhurst or Carrington coordinates), and the tilt angle

γ, which defines the rotation around the y axis in Figure 14. For a detailed description of the mathematical derivation of the GCS model, please refer to Thernisien (2011).

The GCS model is typically employed in a forward modelling approach, i.e., the model is visually compared to coronagraph observations of a CME and the input parameters are then iteratively adjusted by the scientist to achieve a good fit. This manual fitting process is ideally applied simultaneously to coronagraph images from multiple viewpoints, such as from the Solar and Heliospheric Observatory (SOHO)/Large Angle and Spectrometric Coronagraph Experiment (LASCO) and Solar Terrestrial Relations Observatory (STEREO)/Sun Earth Connection Coronal and Heliospheric Investigation (SECCHI) coronagraphs, to avoid ambiguity due to the line of sight effect. The resulting GCS parameters for the best fit can then be used for further evaluation, e.g. as input parameters for modeling the subsequent CME propagation. Additional properties of the flux rope, such as the radius at the apex R_{apex} (see Figure 14) can also be calculated from these parameters, as derived by (Thernisien, 2011). When applied to a sequence of consecutive images, the CME kinematics can also be reconstructed.

The original implementation of the GCS model in the Interactive Data Language (IDL)[1] and a corresponding graphical user interface (GUI) were developed by Thernisien et al. (2006) and are included in the SolarSoft software package (Freeland and Handy, 1998) under the name `scraytrace`[2]. SolarSoft is a collection of IDL software libraries that was originally developed in the 1990s by members of the Yohkoh and SOHO mission teams and the NASA Solar Data Analysis Center (SDAC), and some tools from other missions such as STEREO were also included later. Using this GCS implementation requires a license of IDL, a local installation of SolarSoft and the corresponding database (SSWDB), which includes coronagraph images and calibration data. Obtaining and installing all these components is quite involved for scientists that are not familiar with IDL and SolarSoft. Additionally, the GCS implementation is only partially documented and not very flexible, as it was initially hard-coded to work with only STEREO-A and -B data, with support for SOHO being manually added later.

As described e.g. in the detailed review by Burrell et al. (2018), the Python programming language is becoming increasingly popular in the solar and heliospheric physics community, and consequently, various open source software libraries to assist with the associated data analysis are available. Python is a modern, general-purpose object-oriented programming language that is easy to learn and emphasizes code readability. According to the TIOBE Programming Community Index[3], it has recently surpassed Java as the second most popular programming language in the world, and in contrast to IDL, it is open source software (OSS) and available free of charge on all major operating systems.

SunPy (The SunPy Community et al., 2020), a library for working with solar images from various missions, is one of the most widely-used Python toolkits for solar physics. However, it does not yet provide any models for CME reconstruction

1 https://www.l3harrisgeospatial.com/Software-Technology/IDL
2 https://hesperia.gsfc.nasa.gov/ssw/stereo/secchi/idl/scraytrace
3 https://www.tiobe.com/tiobe-index/

in coronagraph images. Thus, an open source Python implementation of the GCS model and a simple corresponding GUI application based on *SunPy* have been developed during the course of the study presented in Freiherr von Forstner et al. (2021). It can be used both as as a standalone application as well as integrated into existing Python-based plotting routines. The source code is available on GitHub at `https://github.com/johan12345/gcs_python`, and is also mirrored at Kiel University under `https://gitlab.physik.uni-kiel.de/ET/gcs_python`. It can be easily installed with Python's `pip` package manager as follows:

```
pip3 install git+https://github.com/johan12345/gcs_python.git
```

(provided that Python 3.7 or above is already installed).

The following sections will describe the design and usage of this software package, and its validation against the original IDL version.

B.1 GCS GEOMETRY

The basic GCS geometry is implemented in the `gcs.geometry` module. This code is a close translation of the corresponding IDL routines from SolarSoft. Two basic functions are provided to calculate the geometry of the GCS structure based on the input parameters: The `skeleton` function (based on `shellskeleton.pro` in SolarSoft) calculates the shape of the central axis of the flux rope (thin solid line in Figure 14), which consists of two straight segments in the legs and a curved segment in the front. The desired resolution, i.e. the number of points along each part of the curve, can be passed to the function. This central axis, which lies in the xy plane, then needs to be used to generate the outer shell of the flux rope, which consists of circles that are perpendicular to the tangent vector at each point. For this purpose, the `skeleton` function also provides the orientation of this tangent vector as well as the radius of each circle. The `gcs_mesh` function (based on `cmecloud.pro` in SolarSoft) then uses the output of the `skeleton` function to construct a 3D mesh by generating these circles around the central axis with the appropriate radius and orientation. The parameters of the `gcs_mesh` function are the half angle α, the CME height h_{apex}, and the aspect ratio κ, as well as the desired numbers of points along the straight segments, along the front, and along each circle in the mesh.

In addition to these basic routines to construct the GCS mesh, there is also a function `gcs_mesh_rotated`, which uses the three angles θ, ϕ, γ to rotate the CME cloud in 3D space, as well as the function `gcs_mesh_sunpy`, which converts the rotated GCS mesh into a *SunPy* `SkyCoord` object. This object then contains the necessary metadata about the coordinate system so that the model can directly be integrated into a *SunPy* plot.

B.2 GRAPHICAL USER INTERFACE

In addition to the implementation of the GCS geometry, a convenient GUI was created that can be used to fit the GCS model to coronagraph images. The GUI is based upon *SunPy*, which already provides functions to obtain coronagraph images and

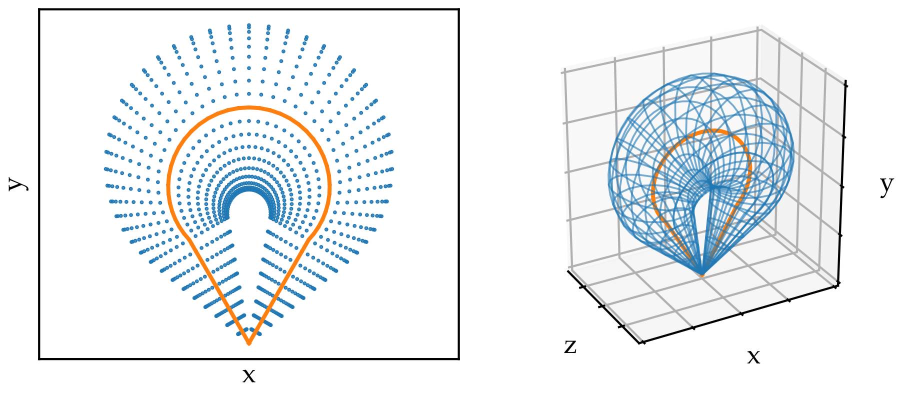


Figure 15: Results of the functions from the GCS Python implementation. The left panel shows the central axis of the flux rope, calculated using the `skeleton` function, in orange, and the surrounding circles generated by the `gcs_mesh` function in blue. The right panel shows a 3D wireframe representation of the same data. The input parameters α and κ are the same as in Figure 14.

to perform the coordinate transformations necessary to overplot arbitrary points in space on these images. Thus, its implementation was relatively straightforward.

For starting the GUI, a command line interface is provided. For example,

```
gcs_gui "2013-05-13 16:54" STB SOHO STA
```

starts the GCS GUI with the closest available coronagraph images to the given date and time from STEREO-B, SOHO and STEREO-A. Additional command line options are available to set which coronagraph should be used (`-soho C2` or `C3`, and `-stereo COR1` or `COR2`) and whether to use running difference images (`-rd`) or direct images. Running difference images are calculated by subtracting a previous image (e.g. 1 h before) from the current one, so that moving features are highlighted.

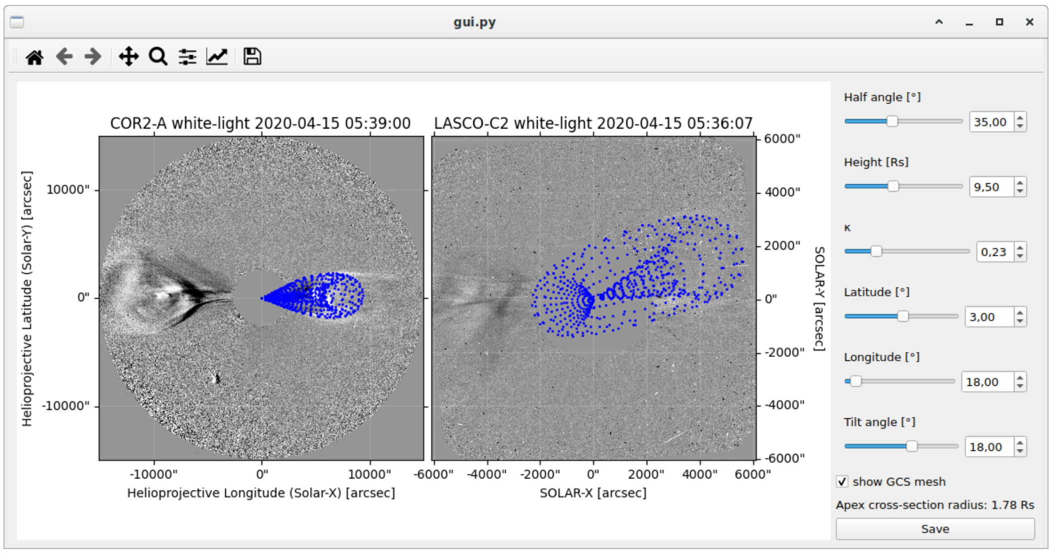
The GUI components, e.g. sliders and text boxes for each input parameter, were implemented using the *PyQt5* library[4], which provides Python bindings for the popular GUI framework *Qt*. The GUI is defined in the `gcs.gui` module as the `GCSGui` class. For embedding the solar images into the Qt window, it uses a plotting canvas provided by the *matplotlib* plotting package (`FigureCanvasQTAgg`).

When the user starts the GUI, it first retrieves the desired coronagraph images through the Internet. This is done using the Helioviewer.org Application Programming Interface (API) [5], which directly provides images in JPEG2000 format to which all necessary calibration and background subtraction routines were already applied on the server side. This drastically simplifies and speeds up the process compared to the IDL version, where images in FITS format need to be downloaded manually from the respective mission sites, and where the calibration procedure needs to be applied locally (requiring an installation of SSWDB). The JPEG2000 images provided by Helioviewer.org also include additional metadata about the observer location and field of view, which are copied from the original FITS file

4 https://riverbankcomputing.com/software/pyqt/
5 https://api.helioviewer.org/docs/v2/

and are necessary to plot the images in the correct coordinate system (solar latitude and longitude).

When the images are downloaded, the GCS GUI plots them using *SunPy* and displays the result in the plotting canvas (see Figure 16). The GCS croissant mesh is then plotted on top of these images and the user can adjust the GCS parameters interactively with the six sliders and numerical input boxes on the right side of the window. Three additional controls are provided: A checkbox to show or hide the GCS mesh, a text view showing the calculated apex radius of the flux rope (R_{apex}, cf. Figure 14), and a button to save the GCS parameters to a file. These data are stored in the JavaScript Object Notation (JSON) format, a general-purpose data format that is human-readable and can be easily handled with most modern programming languages. On the upper edge of the GCS window, the standard *matplotlib* controls are also shown to allow zooming and panning in the images as well as saving the current set of images to a file.


gui.py
COR2-A white-light 2020-04-15 05:39:00
LASCO-C2 white-light 2020-04-15 05:36:07
Helioprojective Latitude (Solar-Y) [arcsec]
10000"
0"
-10000"
-10000"
0"
10000"
Helioprojective Longitude (Solar-X) [arcsec]
6000"
4000"
2000"
0"
-2000"
-4000"
-6000"
-6000"
-4000"
-2000"
0"
2000"
4000"
6000"
SOLAR-X [arcsec]
SOLAR-Y [arcsec]
Save

Half angle [°] 35,00

Height [Rs] 9,50

κ 0,23

Latitude [°] 3,00

Longitude [°] 18,00

Tilt angle [°] 18,00

✓ show GCS mesh

Apex cross-section radius: 1.78 Rs

Figure 16: Screenshot of the GCS Python GUI. The left part of the window shows the plotting canvas with running difference coronagraph images of the April 15, 2020 CME (see Freiherr von Forstner et al., 2021). The user controls for the GCS parameters are shown on the right.

During the development of the GCS GUI, two problems with the data provided to *SunPy* by the Helioviewer.org Application Programming Interface (API) were discovered: First, the STEREO-B COR2 images were temporarily not available through the API in October 2020, an issue which has since been resolved by the Helioviewer.org team[6]. Second, there was an issue with the metadata included in the SOHO/LASCO files: The SOHO spacecraft performs a roll maneuver every three months to keep its high gain antenna oriented towards Earth. The files provided by Helioviewer.org already take this into account by rotating the images accordingly, so that the solar north pole is always pointing upwards. However, the metadata in the JPEG2000 files are not adjusted accordingly[7], so *SunPy* still in-

6 https://github.com/Helioviewer-Project/helioviewer.org/issues/288
7 https://github.com/sunpy/sunpy/issues/4553

terprets these images as though they were rotated by 180°. This obviously caused the location of the GCS model results in the LASCO images to be incorrect. Thus, a workaround was implemented into the GCS GUI to reset the rotation metadata of the LASCO files and also submitted a patch[8] to the *SunPy* project to address this issue, which is included in version 2.1 of *SunPy*.

Possible future improvements to the GCS GUI could include adding an option to calculate base difference images as an alternative to the current options of direct and running difference images, i.e. allowing the user to specify a fixed point in time that should be used for the subtraction. It may also be helpful to include additional controls for the user to adjust the contrast of the displayed images — this is already possible in the IDL implementation, but only before starting the fitting process, it cannot be changed interactively while already working on the fit. In addition, further tools could be provided to facilitate the fitting of time series so that the user can simply provide a start and end time, and the GUI would directly provide one image after another and store the fitting results for each time step in one file. The toolkit may also be easily extended to include support for additional data sources as soon as they are implemented in *SunPy*, such as the Wide-Field Imager onboard Parker Solar Probe (WISPR, Vourlidas et al., 2016), the Metis coronagraph onboard Solar Orbiter (Antonucci et al., 2020) and the Solar Orbiter Heliospheric Imager (SoloHI, Howard et al., 2020).

B.3 VALIDATION

To validate that the Python implementation of GCS yields the correct results, a set of CMEs were re-plotted with the Python GCS GUI that have previously been fitted using the IDL version to compare the resulting plots. One example of this is shown in Figure 17 for a CME launched on May 13, 2013 (originally reconstructed by Gou et al., 2020, Figure 2). The corresponding input parameters are shown in Table 3.

Parameter	Value
Stonyhurst Longitude ϕ [°]	270
Heliographic Latitude θ [°]	19
Tilt angle γ [°]	35
Half angle α [°]	28
Apex height R_{apex} [$R_\odot$]	10.8
Aspect ratio κ	0.35

Table 3: GCS parameters for the May 13, 2013 CME shown in Figure 17.

Similar comparisons were performed for other CMEs with different characteristics: The June 21, 2011 CME (originally reconstructed by Heinemann et al., 2019, Table 1), the May 25, 2014 CME (originally reconstructed by Dumbović et al., 2018a,

8 https://github.com/sunpy/sunpy/pull/4561

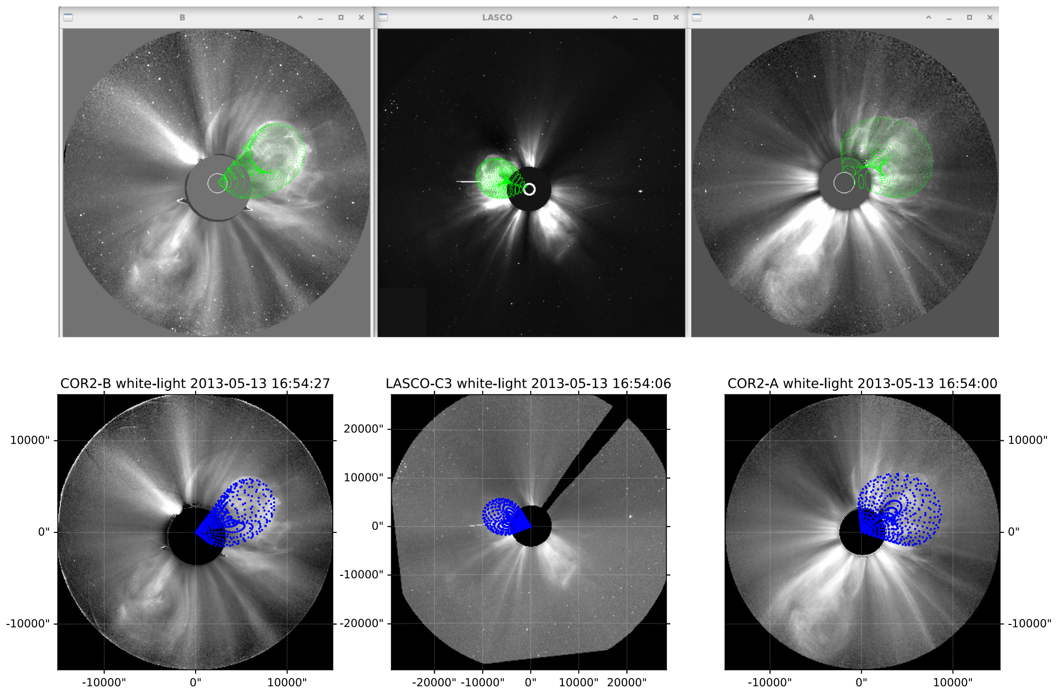

B
LASCO
A
COR2-B white-light 2013-05-13 16:54:27
LASCO-C3 white-light 2013-05-13 16:54:06
COR2-A white-light 2013-05-13 16:54:00
10000"
0"
-10000"
20000"
10000"
0"
-10000"
-20000"
-20000" -10000" 0" 10000" 20000"

Figure 17: Validation of the GCS Python implementation. Both panels show the same CME on May 13, 2013, with the GCS parameters listed in Table 3. The top image was generated by the IDL implementation of GCS, while the bottom panel shows the result of the new Python implementation.

Figure 5b), as well as the April 15, 2020 CME studied by Freiherr von Forstner et al. (2021). In all cases, the GCS 3D meshes generated by the Python implementation match the CME signatures in the coronagraph images as well as the original results from the IDL implementation. This shows that the Python version is implemented correctly and can be used for scientific purposes.