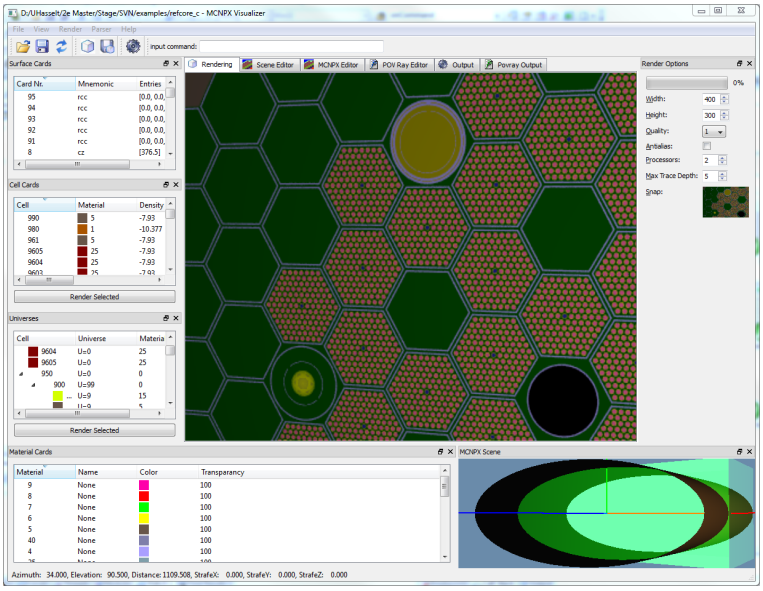
SCK•CEN - Belgian Nuclear Research Centre

User's Manual

MCNPX Visualizer

Author:
Nick Michiels

Mentors:
Gert Van den Eynde
Simon Vanmaercke



Report as an account of work for an internship at the Nuclear Research Centre in Mol

2011

About

This report was prepared as an account of work for an internship at the Nuclear Research Centre in Mol (**SCK•CEN**) [2]. The manual describes the usage of the developped **MCNPX Visualizer** software. The *MCNPX Visualizer* is a visualization tool for MCNPX input files. It parses the geometry of a MCNPX input file into an appropriate **POV-Ray** [1] file. When the parsing is complete, the POV-Ray file can be rendered in the same visualizer. There are several possibilities to edit the rendered scene, i.e. camera placement, quality settings, cross sections, ... It uses a preliminary preparser to visualize the cards (cells, surfaces, materials, ...) defined in the MCNPX input file.

Contents

Chapter 1

Introduction

The *MCNPX Visualizer* is a graphical user interface for rendering the geometry of a MCNPX file. It parses the geometry out of the file and renders it with the *POV-Ray* raytracer. The parsing and rendering are executed as transparent as possible by command line calls of respectively the **Python parser** and the **POV-Ray software**. The Python parser is open source, so it can be extended or debugged manually. An example of the GUI is shown in Figure 1.1.

This manual will help you installing the *MCNPX Visualizer* and will help you to get started with its functionality. The first chapter gives an overview of the basic installation steps for Linux and Windows operating systems. The second chapter describes how to open, parse and render MCNPX files. Every widget of the GUI will be explained here. The chapter mentions how to manipulate the scene parameters. It is for example possible to intuitively place the camera around the scene with a small OpenGL widget. In addition the chapter explains how to use the cross section functionality where the section planes could be manipulated by the OpenGL Widget as well. Finally, in the last section, there is a discussion of the constraints and pitfalls of the system. In Appendix A you find an explanation of the errors that can occur and how you can prevent them. Appendix B illustrates the simplified UML class diagram of the global system.

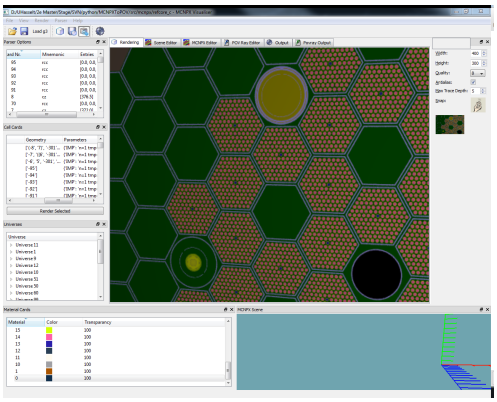


Figure 1.1: Example of the MCNPX Visualizers GUI.

Chapter 2

Installation

This chapter describes the installation steps for *Linux* and *Windows* operating systems. The software requires a working **Python** environment with the **Sympy** extension and a **POV-Ray** installation. The GUI itself is a C++/Qt implementation.

2.1 Linux

The following steps are tested on Ubuntu 10.04.1 (AMD-64).

Configuring Python

Python is a standard Linux environment. The MCNPX-parser needs an extra plugin library for Python, called Sympy. It can be installed from `http://code.google.com/p/sympy/` or use the packet manager:

```
sudo apt-get install python-sympy
```

Configuring POV-Ray

POV-Ray is a raytracer that is widly used for rendering complex scenes with high quality. It can be found at `http://www.povray.org/` or downloaded from the packet manager:

```
sudo apt-get install povray
```

Attention, the MCNPX Visualizer is tested on POV-Ray version 3.6. It is recommended to use this version. Normally version 3.7 will work properly, but in some rare cases it might fail.

Installing MCNPX Visualizer

Now install the complete software package wherever you want. After installation, you can startup the software and it should look like Figure 2.1a.

2.2 Windows

Configuring Python

Dowload Python for Windows from `http://www.python.org/download/`. The software is currently running on version 2.7.1. After the installation, make sure that you add Python to the environment variables of the system. Go to *this computer*, *properties*, *advanced system settings*, *Environment Variables*. Add the path of your installed Python library to the PATH variable (i.e. `C:\Python27\`). You may need to reboot your system. Python is configured correctly if the command is recognized by the command window.

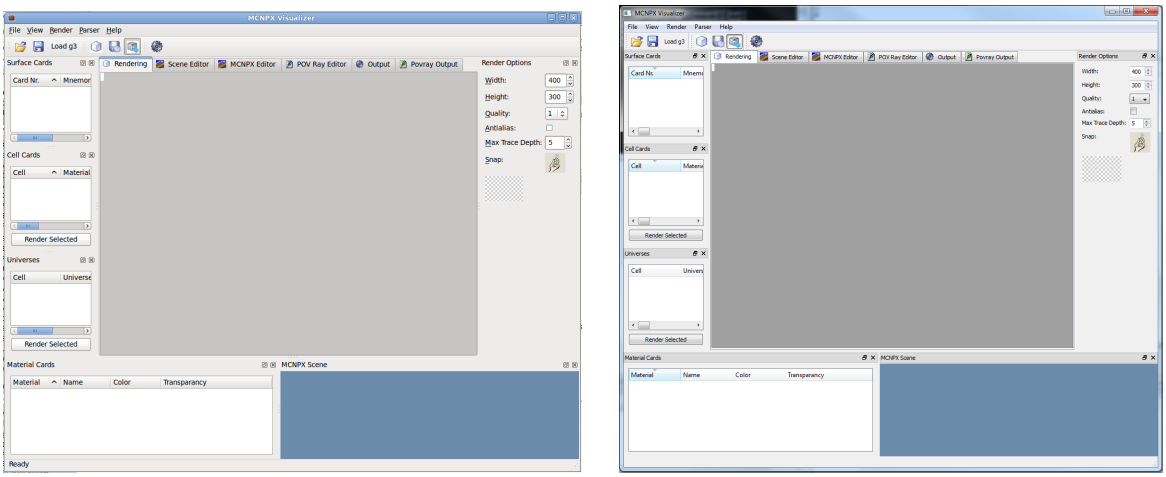
Download the Windows executable for **Sympy** from `http://code.google.com/p/sympy/` and install it. The package should be install in the `Lib\site-packages\` directory of the Python folder. If the package is not correctly installed, the Python parser will give an error in the output window of the application.

Configuring POV-Ray

The executable povray instance is released inside the package of the MCNPX Visualizer software. If the POV-Ray doesn't work, make sure that *options / keep single instance* is turned off and *options / script I/O restrictions* is set on *no restrictions*.

Installing the MCNPX Visualizer

Install the complete software package in a directory of your choice. After installation, you can start using the software and it should look like Figure 2.1b.



(a) Linux (b) Windows

Figure 2.1: Software when it is opened after installation

Chapter 3

Using the MCNPX Visualizer

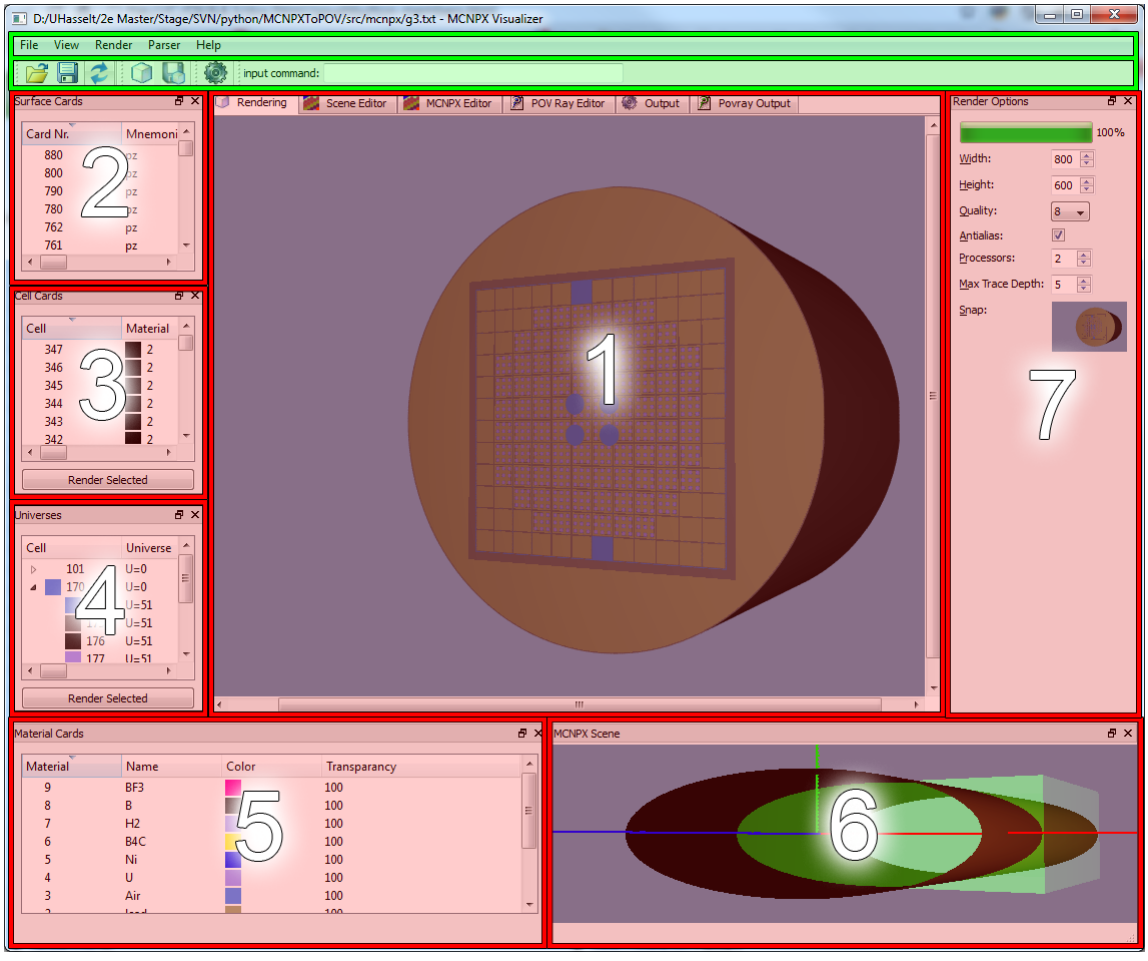


Figure 3.1: The MCNPX Visualizer consists of a set of dockwidgets. These dockwidgets can be used to set up the scene and rendering parameters.

The MCNPX Visualizer consists of a set of docked widgets. Figure 3.1 gives an overview of all the different dockwidgets. The green area describes the main menu and toobar of the sofware. The main central widget (1) contains multiple tabs. **Rendering** is the main tab. It is the most important widget because all the results are rendered to this tab. The other tabs are **Scene Editor**, **MCNPX Editor**, **POV-Ray Editor**, **Output** and **POV-Ray Output**.

Around the main central tab widget, there are some helpfull widgets that gives specific information for the MCNPX file and render options. Widget (2) contains all the **Surface Cards** together with its parameters. The **Cell Cards** are defined in widget (3). A tree stucture of the defined **universes** are given in widget (4). Widget (5) can be used to specify colors and materials for the different **MCNPX Material Cards**. To make the camera placement user friendly, the software contains an **OpenGL widget** discribing the basic structure of the MCNPX file. The camera can be placed very general around the MCNPX scene and rendered from that point of view. Finally in widget (7), the user can specify some basic **quality options** for the rendering of the scene (i.e. resolution or anialiasing).

This chapter will deal with each widget in more detail and will discuss all its functionality.

3.1 Menu Bar

The menu bar contains some basic actions for the software like: opening a MCNPX file, reloading the current file, saving the rendered scene, zoom-in/zoom-out/stretching of the rendered scene, parsing, ...

3.2 Toolbar

The toolbar is illustrated in Figure 3.2. As you can see in the Figure, there are four different toolbars. The first one contains all the basic file operations: opening, saving and reloading the file. The second one contains the render and save rendered scene actions. The third one contains a button for parsing the input file. The last toolbar is the most powerful one. It has the command line functionality. The commands that can be used are similar to the original MCNPX Visualizer. Here are a couple of commands that can be used:

origin v_x v_y v_z	Position of the plot so that the origin, which is in the middle of the plot, is at the point (v_x, v_y, v_z). The default values are 0 0 0.
extent e	Set the scale of the plot so that the horizontal distance from the origin to either side of the plot is e. The default value is 100.
reset	Reset the properties of the plot to the defaul values.
px v_x	Plot a cross section of the geometry in a plane perpendicular to the x-axis at a distance v_x from the origin.
py v_y	Plot a cross section of the geometry in a plane perpendicular to the y-axis at a distance v_y from the origin.
pz v_z	Plot a cross section of the geometry in a plane perpendicular to the z-axis at a distance v_z from the origin.

The keywords origin and extent can be combined with the px, py or pz commands. The commands are independent of the current scene settings of the GUI.

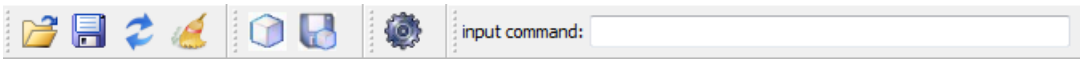


Figure 3.2: Toolbar of MCNPX Visualizer

3.2.1 Open

Open a new MCNPX file. It automaticly starts the preparser so all the widget information can be filled in from the file.

3.2.2 Save

Save the current MCNPX file.

3.2.3 Reload

Reload the current MCNPX file. If the user made changes in the MCNPX file, this action will reopens the file and restarts the preparser, so the widget information is updated with the new file information.

3.2.4 Delete Temporary Files

Delete all the temporary files of the currently opened MCNPX file. This will reset all the saves material settings.

3.2.5 Render

Start rendering the current scene. If the MCNPX file is parsed to a POV-Ray file and the camera is placed to the wanted position, this action will start to render the scene with the predefined quality parameters.

3.2.6 Save Rendered Scene

Write the rendered scene (visible in the central render widget) to an image file.

3.2.7 Parse

Parse the current MCNPX file. This is necessary to build a MCNPX file into an appropriate renderable scene. The user starts this action if he opened a new MCNPX file or if he changed the file or the cards of the file. After parsing it once, it is not necessary to reparse it over and over again. Changing rendering or camera paramaters have no effect on the parsed output, only when the materials or intput file are changed.

3.3 Surface Cards

The *Surface Cards* widget (2) contains a list of all the defined surfaces in the MCNPX file, together with its parameters. An example is given in Figure 3.3. It is only informational and can't be used to modify the surfaces. Only in the main text editor the content of the MCNPX file can be edited.

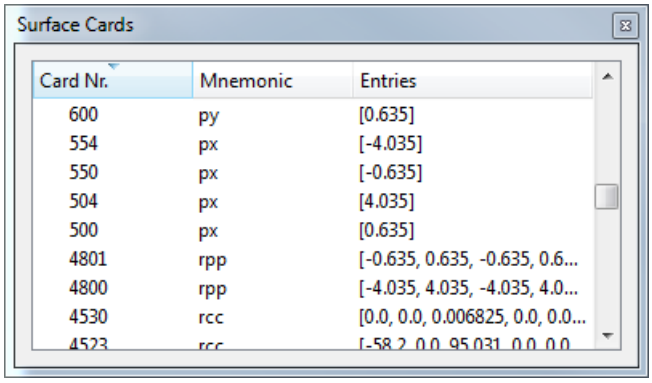


Figure 3.3: Surface Cars of a MCNPX file.

3.4 Cell Cards

The *Cell Cards* widget (3) contains a list of all the defined cells in the MCNPX file, together with its parameters. An example is given in Figure 3.4. It is only informational and can't be used to modify the cell cards. Only in the main text editor the content of the MCNPX file can be edited.

It's possible to only render the selected cells, by pressing the *Render Selected* pushbutton. This action will start to render the scene with the current camera and quality parameters. It corresponds to the main render action, but now only for the user selected cell cards.

ATTENTION: The translations and rotations of higer level cells will not be accounted for if they are not rendered. So, you need to be very carefull while placing the camera or defining cross sections. The toplevel geometry is not preserved.

3.5 Universes

The *Universes* widget (4) contains a tree structure for the cell hierarchy present in the MCNPX file. An example is given in Figure 3.5. It gives an overview of all the levels of detail in the MCNPX file. The toplevel cells are in the first level of the tree, then all subuniveres are recursively added to the tree. The material of the cell is also added and can be changed.

Like the cell cards, it's also possible to render only the selected subuniverse or cell, by pressing

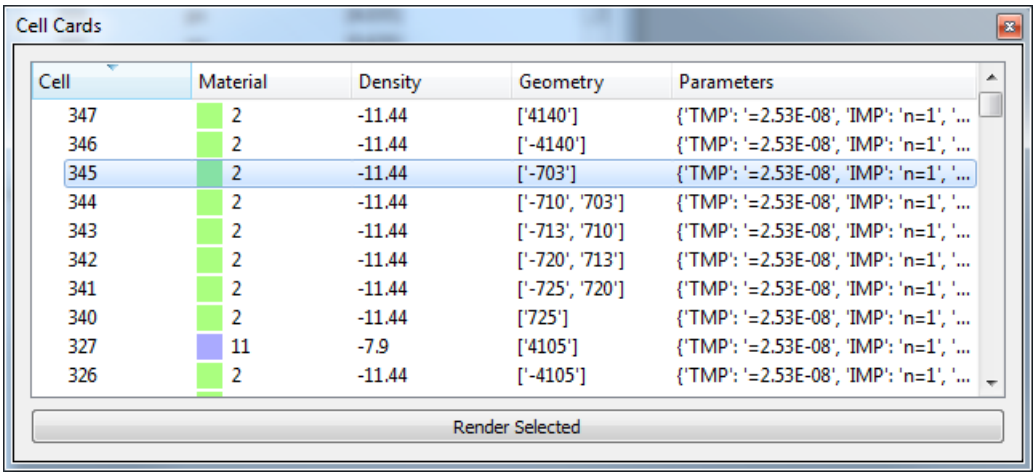


Figure 3.4: Cell Cars of a MCNPX file.

the *Render Selected* pushbutton. This will start to render the scene with the current camera and quality parameters. The action corresponds to the main render action, but now only for the user selected subuniverse or cell. For example, in Figure 3.5, cell 170 is selected and it contains subuniverse 51. So when the user pushes the *Render Selected* pushbutton, only subuniverse 51 will be rendered.

ATTENTION: The translations and rotations of higer level cells will not be accounted for if they are not rendered. So, you need to be very carefull while placing the camera or defining cross sections. The toplevel geometry is not preserved.

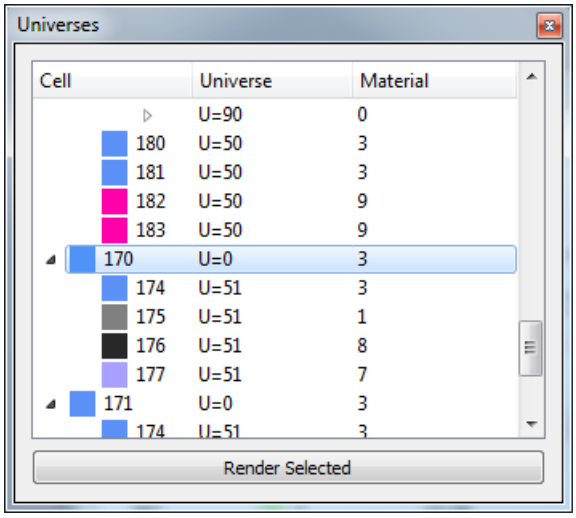


Figure 3.5: Universes hierarchy of a MCNPX file.

3.6 Material Cards

The *Material Cards* widgets (5) contains all the materials that are present in the MCNPX file. An example is given in Figure 3.6. The user can use this widget for changing the color and transparency of the materials appearance. It is required that the material is specified as a material card in the MCNPX file, otherwise he is not added to this widget and the parsing result is biased.

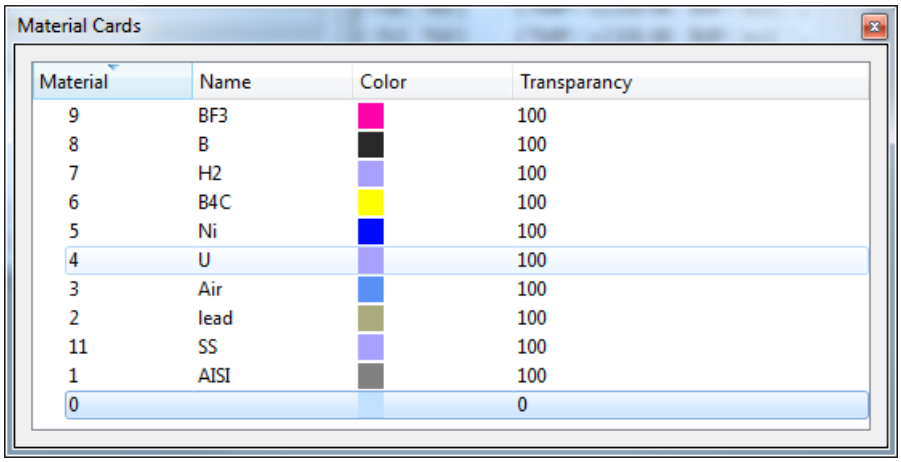


Figure 3.6: Material Cards of a MCNPX file.

The selection of the initial color is specified in various grades of detail. At first there is a specific mapping file for material names to colors. This file can be found in the data directory of the software. The file is called *materials.txt*. The writer of the MCNPX file can use a material name (defined in this file) in the comment line just above the material card. If it is done properly, the *MCNPX Visualizer* will perform a lookup for the materials color based on the given name. In the widget, the material name will appear. An example of a color map file is given in Figure 3.7. This file can be extended by the user if he wants to add new materials. It is only necessary to use the right syntax form:

```
Material_name => red green blue alpha
```

The material name can be any string without spaces. The red, green and blue values are numbers in the range 0 to 255 and the alpha is a number 0.0 to 1.0. The comment syntaxes are the same as for the MCNPX file. A material name must be specified in the MCNPX file like this:

```
c  m3=Air
m3   nlib=03c
```

Where **m3=Air** is put in a comment line just before the **m3** material card.
If the name is not specified in the material card or the material is not found in the materials file, a pseudo random color is chosen. After parsing the MCNPX file, the colors for the materials

are written to the temp directory of the software. This saved file will be used every time the MCNPX file is opened again. If there is no change found while reopening a MCNPX file, the saved materials are used. If you want to delete those temporary files of the current MCNPX input file, you need to click on the **delete temp files** in the menu.

Finally, it's possible to edit the colors and transparency levels of the materials in the widget by double clicking on the color or transparency and edit its value.

```
C MATERIAL COLORS
C CASE INSENSITIVE
C *Material name* => *red* *green* *blue* *alpha*


C -----------------------------------
C METALS
C -----------------------------------
SS              =>      169     159     255     1.0     $ (Stainless Steel)
lead            =>      170     170     127     1.0
aluminium       =>      182     197     190     1.0
U               =>      169     159     255     1.0
Ni              =>      0       10      255     1.0
B4C             =>      255     255     0       1.0
B               =>      40      40      40      1.0
titanium        =>      82      175     169     1.0

C -----------------------------------
C GAS
C -----------------------------------
Air             =>      91      144     246     1.0
H2              =>      169     159     255     1.0
BF3             =>      255     0       170     1.0     $ (Boron Trifluoride)

C -----------------------------------
C LIQUIDS
C -----------------------------------

C -----------------------------------
C OTHER
C -----------------------------------
soft_tissue     =>      219     151     149     1.0     $ (soft tissue)
skeleton        =>      205     205     205     1.0
lung            =>      170     68      68      1.0
Adipose         =>      242     248     50      1.0     $ (fat tissue)
Skin            =>      239     208     207     1.0
Muscle          =>      207     7       7       1.0
TLD             =>      78      78      78      1.0     $ (TLD Detector)
```

Figure 3.7: Color map for material names.

3.7 MCNPX Scene

This widget contains an OpenGL rendered view of the scene (see Figure 3.8) (6). The preparser will parse the cellcard that is defined as **imp:n=0**. The geometry of this cellcard is parsed and added to the OpenGL widget to let the user see where there is an importance of 1 for the particles. This will make it easier to place the camera around the scene and have a notion of the position of the scene.

ATTENTION: The geometry of the imp:n=0 cellcard needs to be simple enough to parse. It can consists of basis macro bodies or a combination of planes, but no complex boolean operators like intersections.

Different from giving a notion of size, the OpenGL widget has two main functionalities: **placing the camera**, and **viewing the cross sections**.

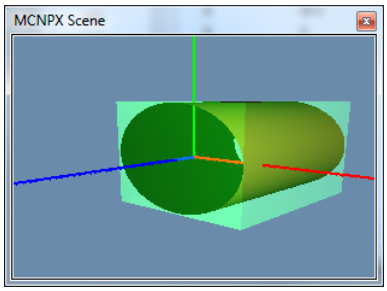


Figure 3.8: OpenGL rendered MCNPX scene of the imp=1.

Camera

The position of the camera is based on three values: **azimuth**, **elevation** and **distance**. Based on the origin of the coordinate system, the camera can be rotated around the object with an azimuth and a specific height, called the elevation. The distance is defined as the metric distance of the camera to the origin. To give the camera placement more freedom, there are three extra parameters: **strafe X**, **strafe Y** and **strafe Z**. These values specify a displacement of the origin.

The camera options can be edited in two ways. At first, you can use the input from mouse and keyboard. For azimuth and elevation you can left-click on the widget and rotate it with the mouse. The strafe X, Y and Z are respectively performed by left-click and holding *ctrl*, *alt* and *ctrl+alt* while moving from the left to the right on the widget (**attention: for strafing in the y-direction, you also need to move the mouse from left to right or the other way around**). Secondly, the camera options can be edited via the GUI in the **Scene Editor Tab** (described in Section 3.10).

Cross Sections

When defining cross section planes (see Section 3.10), the planes are also visible in the OpenGL MCNPX Scene widget (green planes). There are many ways to define cross sections. The main two cross sections displayed in this widget are **3D sections** (Figure 3.9a) and **pie slices** (Figure 3.9b).

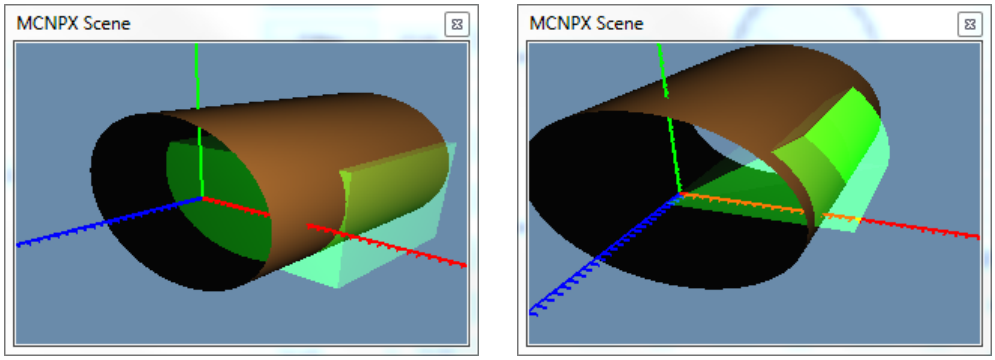
3.8 Render Options

In the Render Options widget (7), the user can specify some basic options for the quality of rendering. The resolution can be edited with the **width** and **height** parameters. The quality level can be set from 0 to 11 (POV-Ray quality parameter), where:

0-1 Just show quick colors. Use full ambient lighting only. Quick colors are used only at 5 or below.

2-3 Show specified diffuse and ambient light.

4 Render shadows, but no extended lights.



(a) 3D Section (b) Pie Piece

Figure 3.9: The basic cross section planes are illustrated in the MCNPX Scene widget.

5 Render shadows, including extended lights.

6-7 Compute texture patterns, compute photons

8 Compute reflected, refracted, and transmitted rays.

9-11 Compute media and radiosity

Antialiasing can be turned on or off. The antialiasing will take a great amount of rendering time, so it is better to only put this on when you want to finialize a rendered scene. The **number of processors** used to render the scene can be changed. Standard there are 2 processors used. It will startup two or more different POV-Ray processes that renders their part of the image. The **Max Trace Depth** must be set higher when you define transparent or semi transparent materials. If there are lots of refractions and the max trace depth is low, the pixel will result black and give artifacts in the final result.

Last but not least, there is a **snap shot** functionality. When you click on the snap shot area (right of the snap label), the renderer will start at a very low resolution and outputs its result in the snap shot area. The main advantage of this snap shot area is that you can rapidly render the scene and see very quickly how the placement of the camera is done and what area of the scene is visible on the rendered image. It allows the user to place the camera very fast at the desired postion without rendering intermediate high level resolution frames.

At the top of the widget, there is a progress bar for rendering the image. This bar gives only intermediate progress in a Linux environment.

3.9 Rendering

This is the main widget of the software. It is the most important widget of the central tab widget (1). It contains the rendered scene. The user can change the view of it. Normaly after

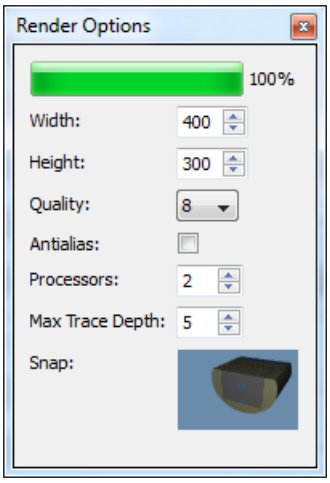


Figure 3.10: Render options. Quality options for the rendered scene.

rendering, the result is displayed in **Normal size**. In the menu **View**, the user can stretch the image over the whole widget by using **Fit to window**. It is also possible to **Zoom in** and **Zoom out**. You can save the rendered scene by clicking on the **Save Rendered Scene** action (or **ctrl+S**) in the menu or toolbar.

3.10 Scene Editor

The scene editor is a very big tab to controll the scene to be rendered (Figure 3.11). It is divided in two big areas. At the left, there are the main **camera properties** like described in Section 3.7: **azimuth**, **elevation**, **distance**, **strafe X**, **strafe Y** and **strafe Z**. It is also possible to change from a perspective camera view to an orthographic camera view. At the right area of the tab there is some functionality to create cross sections of the scene. It is possible to create standard **2D orthographic sections**, **3D cutout sections**, **3D pie pieces**,... This functionality will now be discussed.

The top 3 buttons **PX**, **PY** and **PZ** of the sections area are basic 2D sections aligned with an axis. For example, the pushbutton *PX* will create a cross section with a section plane on **x=base** and a specific camera **distance**. The X,Y and Z coordinates of the origin can be changed. This functionality is similar to the command line input described in Section 3.2 and is independent of the other scene parameters.

There are two other possibilities to create cross sections. The first one is a rectangular cutout. The scene has a bounding box. This boundig box acts as an initial 3D cutout. The planes of this box can be moved, so a 3D cutout area can be defined. When the section planes are placed on the desired positions, the rendering can be started by clicking on the main render action. The checkbox **Limited by bounding box** can be deactivated. This will make it possible to define the section planes more freely. This can be usefull when there are only a small amount

of cell cards rendered and the transformation is different from the total scene. In front of the section planes, there are some pushbuttons defined: **X min**, **X max**, **Y min**, **Y max**, **Z min** and **Z max**. These buttons are shortcuts for rendering 2D sections like PX, PY and PZ. They will start rendering a 2D orthographic section of the plane that is described after the button, centered at the origin of the plane.

The user can enable the **Pie Piece** checkbox. This will enable the Pie Piece properties and will disable the rectangular cutout properties. There are three different possibilities for defining pie pieces, all aligned by a specific axis plane. For example, **XY** pie piece, is a pie piece perpendicular to a z-plane. The pie piece is defined by 4 parameters and can be translated with the **strafe parameters**. **Angle Max** and **Angle Min** will define the angles of the piece. The **Radius** discribes how big the piece is. The **Height** will define the length of the 3D pie piece.

Figure 3.12 shows some examples of 3D sections.

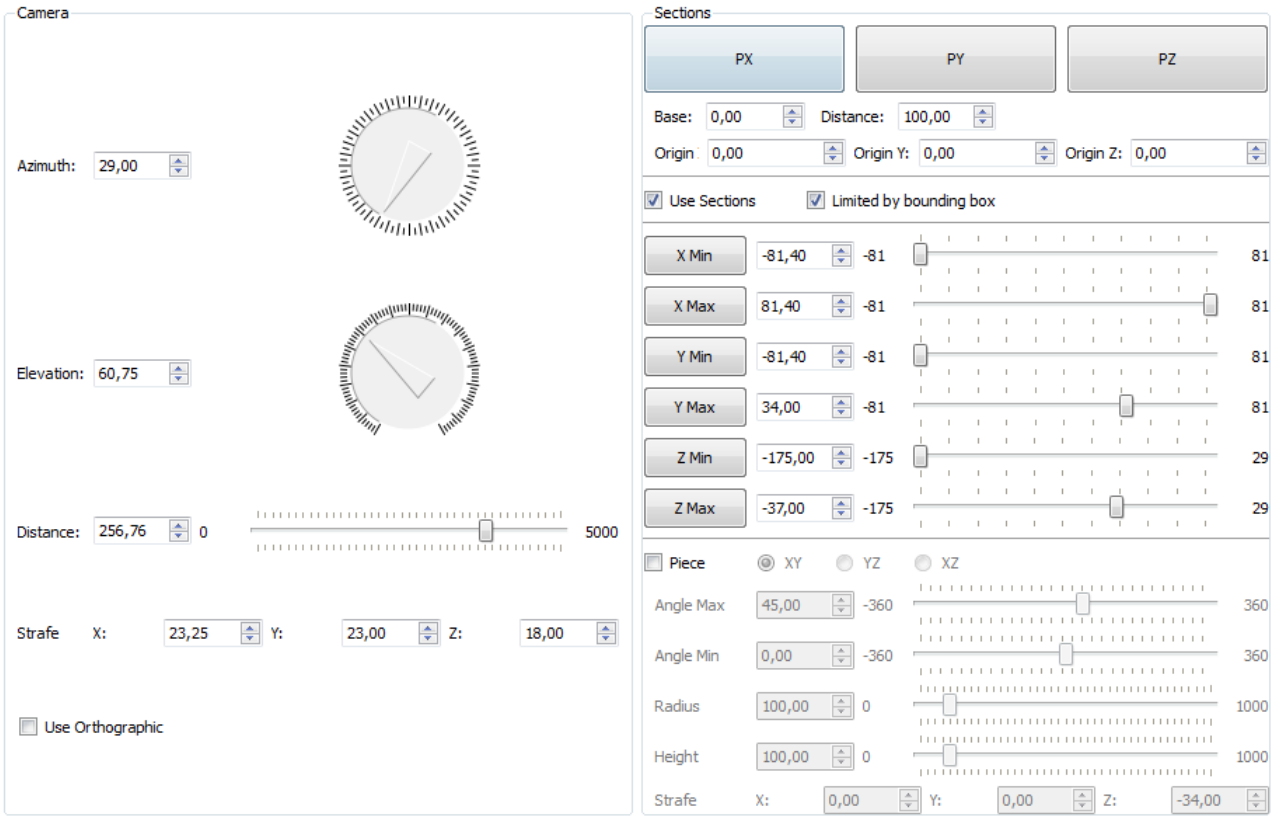
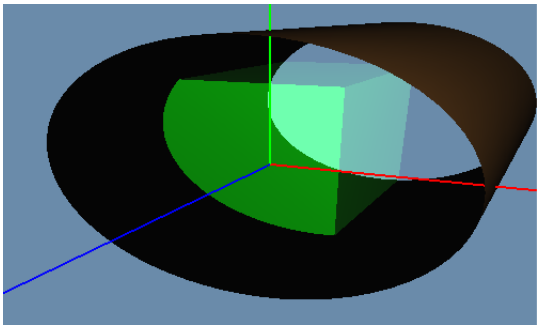
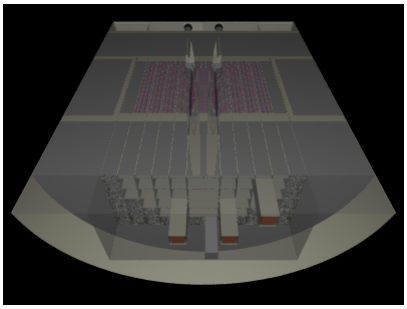


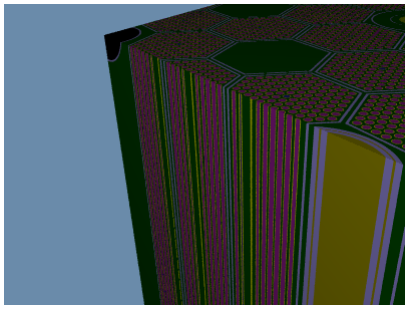
Figure 3.11: Scene Editor.



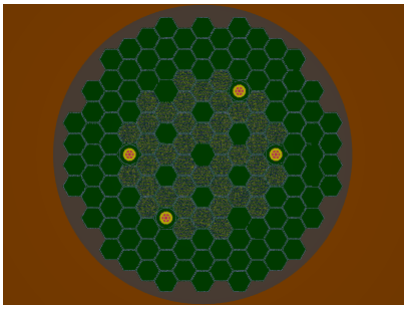
(a) Example 1



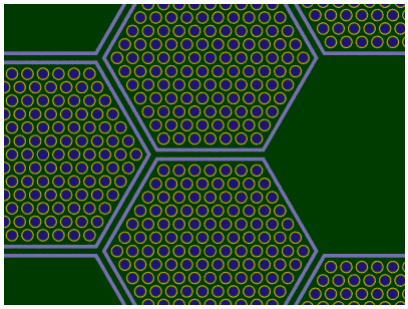
(b) Example 2



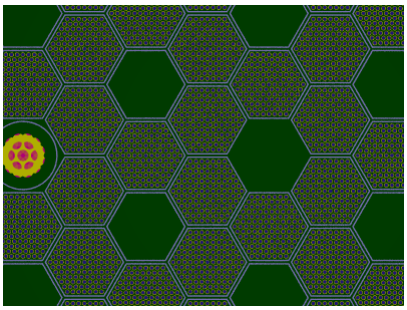
(c) Example 3



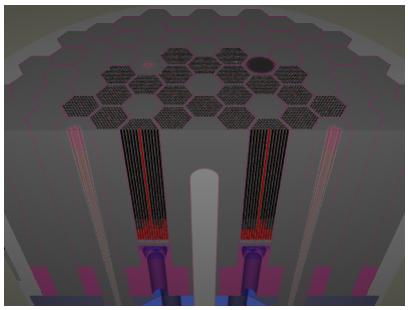
(d) Example 4



(e) Example 5



(f) Example 6



(g) Example 7

Figure 3.12: Examples of cross sections.

Chapter 4

Constraints

This chapter will discribe the basic restrictions of the input of the MCNPX file and the rendering system. The first section will sum up all the elements that should be accounted for when creating the MCNPX file. The second section contains some global rendering limitations of the software.

4.1 MCNPX Constraints

Global Layout

The global layout of the MCNPX file needs to be very strict. There can only be 3 or 4 paragraphs (with or without MESSAGE block), seperated by exatly one white space. At the end of the file, there can only be one white space. The basic syntax of the MCNPX file must be satisfied in order to parse it correctly. The software tries to give clear error feedback in the output window when the parsing failed. These errors are described in Appendix A.

Lattice

Lattices always need to be defined with a complete FILL-specification. This parameter defines how the lattice is filled in every direction. The parser always needs to know a full specification of the form:

```
fill=-8:8 -8:8 0:0
```

where all the three axis are defined.

To fill the lattice correctly and every universe is shifted correctly, the bounding box of the geometry of the toplevel lattice element needs to be calculated. This is only possible if the geometry of the element is rather simple. For hexagonal lattices of type 2, the element needs to be defined as a RHP or HEX macrobody. For rectangular lattices of type 1 there are multiple possibilities. At first, the BOX, RPP or REC macrobodies can be used. Secondly the geometry can be a combination of 6 planes. At all times, this bounding box should be calculated. If it's not possible, the output returns an error and the MCNPX file is not parsed.

4.2 Rendering Constraints

imp:n:0

In order to render the inside and outside geometry in the OpenGL widget of the GUI, the user should declare the imp:n:0 as a cellcard parameter. Just as for the lattice elements, the geometry should be simple to interpret and to calculate a bounding box from. The geometry can be a combination of simple aligned planes and macrobodies like BOX, RPP, REC and RCC. The result should always define a rectangular or cylindrical shape. If the parser failed to interpret the geometry, it informs the user with a warning, but the result is still usable. The only side-effect is that the OpenGL widget and defining the section planes in the GUI is not that straighthforward anymore.
An example of a cylindrical geometry is:

```
c Cell Cards
100  0 (3801:-880:712)

c Surface Cards
3801 cz 80.3999
880  pz -174
712  pz 28.7
```

Another example is:

```
c Cell Cards
100 0 #(-8 -301 200)

c Surface Cards
8   cz  376.5
301 pz  201
200 pz  -250
```

Bounding Boxes

There more the parser can calculate bounding boxes, the faster the scene will render. This is a main advantage of the POV-Ray **bounded_by** object. For intersections and differences it can give a speedup of 20%.

Infinite colors

The user should be aware that infinite colors are not always rendered. If, for example, the user defines a cell with an outside surface, the rendered object will still looks like the object itself and the entire space is not filled. Only when there is a cross section plane defined. The colors will appear. In this case, the user should be very carefull while interpeting 3D projections of the scene.

Transparency

Defining materials with a transparency level can give the same problems as described in the section above (infinite colors), even if cross sections are used. The semi-transparant material will introduce a new projection of 3D objects.

To avoid black pixels in the resulting image, the **max_trace_depth** should be set hight enough. This is only necesarry if there are lots of object defined with a transparency level.

Size scene vs Speed

The size of the scene is limited by the memory of the system. POV-Ray is a very powerful tool but needs lots of processing power. The speed of the rendering is manageble for smaller scenes. But as soon as the scene gets bigger the rendering times rises very fastly.

Bibliography

[1] Persistence of Vision Raytracer. Pov-ray. World Wide Web, 2008. http://www.povray.org.

[2] SCK•CEN. Sck•cen - belgian nuclear research centre. World Wide Web, 2008-2011. http://www.sckcen.be/en.

Appendix A

Errors/Warnings

Cell Parse Errors

No fully specified fill found in cell x	The lattice of cell x needs a full specification of the fill parameter (i.e. fill=-8:8 -8:8 0:0).
Problem reading fill boundary parameters, too little args in cell x	The lattice of cell x needs a full specification of the fill parameter (i.e. fill=-8:8 -8:8 0:0).
Cell x contains a LAT with unknown type y	The lattice of cell x needs to be of type 1 or 2, but not y.
Cell x contains LAT, but no FILL	Cell x has a LAT parameter, but no specification how it should be filled (forgot FILL parameter?).

Cell Build Errors

Reached a maximum recursion depth	The software exceeded the maximum recursion depth of universes. Maybe there is a loop in the MCNPX file.
Bounding box of lattice card must be known in order to fill the lattice correctly	The bounding box of the geometry of a lattice element must be defined properly. It can only be a HEX, RHP, BOX, or RPP macrobody or a rectangular combination of planes. Be sure there is no infinit direction. The specification of the FILL can be 0:0, but the item itself must be limited. Probably the range for the 0:0 fill couldn't be calculated.
HexOffset could not be calculated	see *Bounding box of lattice card must be known in order to fill the lattice correctly.*
Unable to find bounding box for subsurface	Geometry consisting of subsurfaces (brackets) can't be used as bounding box of lattice element.
Material m of cell x not known	Material m of cell x is not found and is not defined as a material card.

Universe Build Errors

Universe x not found	Try to build universe x, but universe not defined in any cellcard.

Surface Card Build Errors

Surface x not known	Surface x not defined as a surface card.
Surface x with type y has not enough or too much arguments	The arguments of the type y are not correctly defined for surface x. Beware that the software needs fully specified information.
Write surface x of type y to file not succeeded. Not enough arguments.	The software tried to write a surface x of type y to file, but did not succeeded because it has not enough information to do it.

Parsing

Bad density input for cell x solved to y	Format of the density was not correct
Not enough/too many newlines in input file expected 2 or 3 newlines got	There is no clean subdivision of the different cards. Make sure there are no extra newlines at the end of the MCNPX file (max 1 empty line) and all the other blocks are seperated properly with one new line. It is recommended to also use the MESSAGE block.

Warnings

Hexagon is not regular and is not been drawn	A hexagon needs to be aligned with a basic axis so it can be regular. The hexagon is ignored and the builded result is probably wrong.
Type of surface x of surface card y not defined	Type x of the surface used in surface card y is not a valid surface type. Beware that the surface is not rendered, so the result can be wrong.
Too much cellcards with imp:n=0	There are more cellcards defined with imp:n=0. The preparser couldn't find the right outercase to draw in the OpenGL widget.
Celcard with imp:n=0 not found	Cellcard with imp:n=0 is not defined. The preparser couldn't find the right outercase to draw in the OpenGL widget.
Get bounding box for surface x of type y failed.	Failed to calculate the bounding box for surface x of type y. Make sure the properties of the surface are defined correctly.

Appendix B

Class Diagram

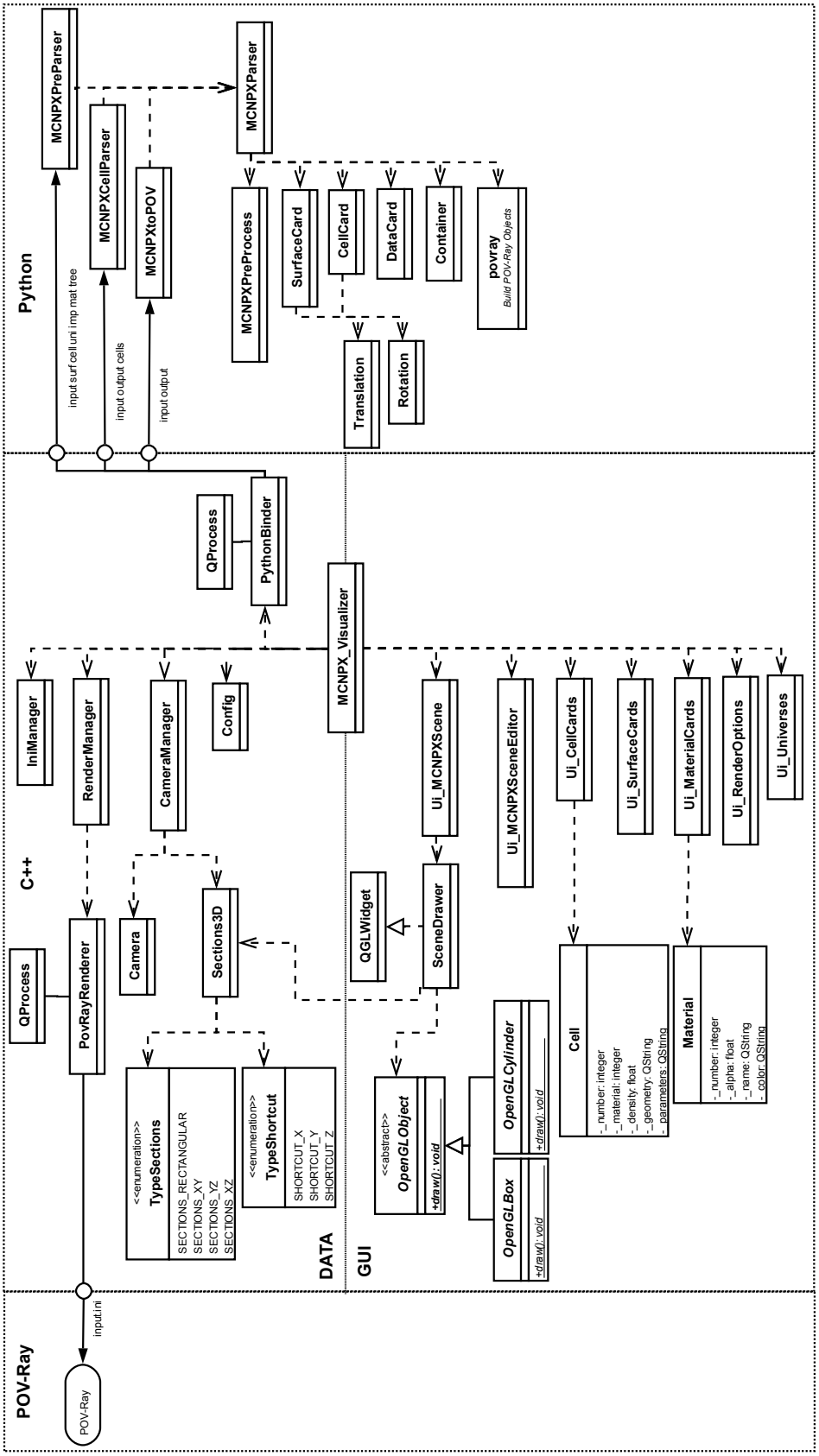


Figure B.1: UML Diagram of the MCNPX Visualizer