EXPERIMENTALLY EXPLORING ALGORITHMIC DESCRIPTIONS OF THREE-DIMENSIONAL GEOMETRY

Elizabeth Skiba

Problem and Motivation

Describing three-dimensional geometry is important in a number of application areas including scientific modeling and engineering. Describing three-dimensional geometry becomes increasingly difficult as the modeled geometry becomes more complex. One technique used to reduce this complexity is CSG (Constructive Solid Geometry) in which simple, easily describable shapes, or primitives, are combined using Boolean operators to construct more complex shapes. Our research group at SUNY Geneseo uses this technique to model the geometry of particle physics simulations in our software IViPP [2]. IViPP displays the geometry of a simulation as well as the resulting particle trajectories.

The geometry descriptions that IViPP displays are based on the particle physics simulator MCNP [5]. We have developed a tool called pyMGeo that automatically generates geometry descriptions for this simulator. pyMGeo is categorized as an algorithmic description tool because it allows users to write algorithms to describe the geometry, for example, through the use of classes, loops, and recursion. pyMGeo seems to be an effective method for generating geometry descriptions for MCNP. In order to test how well pyMGeo performed, I compared it with existing tools and methods for generating MCNP geometry descriptions.

Background and Related Work

The most widely used tools for describing geometry are graphical-user-interface based and algorithm-based tools. GUI-based tools allow users to graphically position pieces of geometry to generate descriptions. Blender is an example of a popular GUI-based tool [3].

Algorithmic tools are text-based tools that allow users some computation in describing geometry. PostScript, a printer control language, is algorithmic because users have computational power including the ability to create functions and use control structures [1]. Algorithmic tools range in level of abstraction. Graphics APIs or graphics programming languages are algorithmic but often require users to work at a low level of abstraction. For example, users are required to define each vertex in the geometry description. pyMGeo is an algorithmic tool that provides a higher level of abstraction, allowing users to define primitive shapes and their transformations.

MCNP accepts textual descriptions of geometry, but the notation is unique in not being GUI-based but also not providing users with the ability to perform computations in descriptions.

Although algorithmic and GUI-based tools are most widely used for generating geometry descriptions there are other methods being researched. For example, three-dimensional geometry descriptions can be generated from user created sketches or sketches on existing images [4, 6].

Approach and Uniqueness

The three tools and methods I compare in my experiment are: pyMGeo, an algorithmic description tool; VisEd, a GUI-based tool, and hand-written MCNP; Because all these tools share a common target, MCNP, differences between the tools can be measured independently of differences in the descriptions they produce.

As an example of each tool in this experiment, consider a simple model in which a beam of neutrons is shot from the origin towards a carbon disk. Figure 1 shows the pyMGeo source code for generating a geometric description of this model. Within the definition of a `CarbonModel` class are variables storing dimensional and transformational information about the carbon disk such as

`carbonRadius` and `carbonTransformation`. At the end of the source code there is a main program that generates the model and writes the description to a file.

```
10  class CarbonModel(MCNPObject):
11      carbonRadius = 3.81
12      carbonThickness = 10
13      carbonTransformation = Translate(y = 100.0)
14      simulationAreaRadius = 5
15      simulationAreaThickness = 120
16      carbon_12 = Material("Carbon 12", Elements.carbon_12(100))
17
18      def __init__(self, model):
19          MCNPObject.__init__(self, model)
20
21          graphiteDisk = Cylinder("carbon 12 disk", self.carbonThickness, self.carbonRadius, self.carbonTransformation)
22
23          self.cells.append( Cell("graphite disk", self.carbon_12, 2.26, graphiteDisk, ("N", 1)))
24
25          simulationArea = Cylinder("simulation area", self.simulationAreaThickness, self.simulationAreaRadius)
26
27          self.cells.append( Void("simulation area", Difference(simulationArea, graphiteDisk), ("N",1)))
28          self.cells.append( Void("outside simulation area", Compliment(simulationArea), ("N", 0)))
29          self.addToModel()
30
31  if __name__ == '__main__':
32      model = MCNPModel("Graphite Demo -- Generated by pyMGeo --")
33      CarbonModel(model)
34
35      file = model.write("C:/Users/eks6/Desktop/carb")
36      file = open('C:/Users/eks6/Desktop/carb.txt', 'a')
37      file.write('MODE N\n')
38      file.close()
```

Figure 1 – Example of pyMGeo, an algorithmic description tool

Figure 2 shows a hand written description of the same model. Lines 2 – 8 are CSG descriptions composed of the primitive shapes in lines 10 – 14. Most MCNP users write these descriptions by hand. pyMGeo generates output in the form shown in Figure 2.

```
1   Graphite Demo -- Generated by pyMGeo --
2   C Cell definitions
3   C graphite disk
4   1 1 -2.26 -1 IMP:N=1
5   C simulation area
6   2 0 ( -2 1 ) IMP:N=1
7   C outside simulation area
8   3 0 2 IMP:N=0.0
9
10  C Surface definitions
11  C carbon 12 disk
12  1 RCC 0.0 100.0 0.0 0.0 10.0 0.0 3.81
13  C simulation area
14  2 RCC 0.0 0.0 0.0 0.0 120.0 0.0 5
15
16  C Material definitions
17  C Carbon 12
18  M1 6012 1.0
19  MODE N
```

Figure 2 – Example of hand written MCNP

Figure 3 shows a GUI-based tool, VisEd. User interaction is centered on a pair of two-dimensional projections of a geometric description as it is created.

Simulators allow experimental results to be gathered quickly on a succession of alternative physical configurations. Therefore, it is important for a tool to allow changes to be made to a geometric

description with very little effort. I measured the effort to make changes with each of the three tools to four existing geometry descriptions, or models. To ensure the experiment would be unbiased both the models and the changes made to each model were contributed by collaborating physicists.

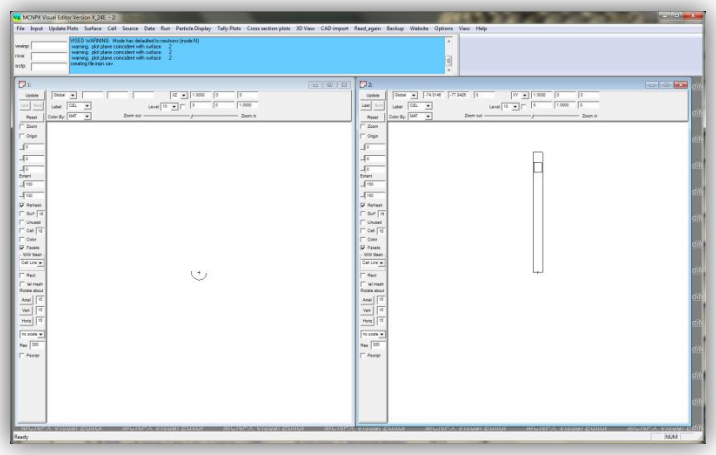

Figure 3 – Example of VisEd, a GUI-based tool

Figure 4 shows typical changes from one model in my experiment. In this model there is a radiation source (red) contained in a holder (blue) sandwiched between two carbon disks (purple), sandwiched between two detectors (green). This assemblage is contained in an air volume (yellow).

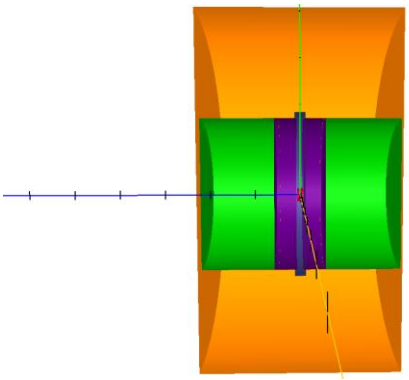

Figure 4a - Gamma Ray Detector Original

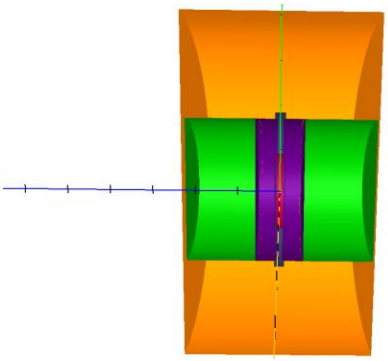

Figure 4b - Enlarge Sodium Disk

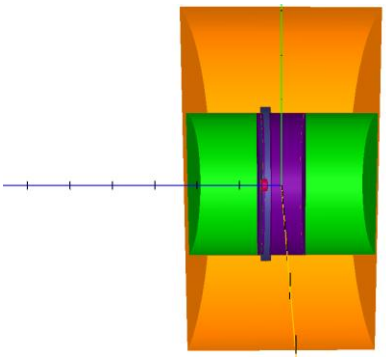

Figure 4c - Source Closer to Disk

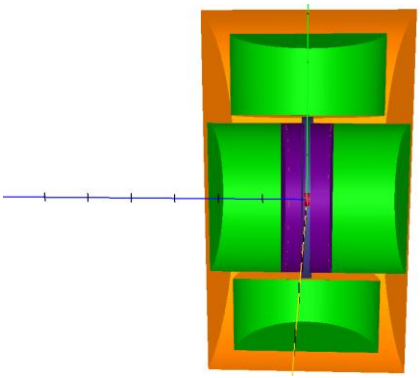

Figure 4d - Add Two More Detector

In the first change to this model, the radiation source was enlarged, as shown in Figure 4b. The second change moved the radiation source and the holder closer to the leftmost detector, as shown in Figure 4c. The last change, shown in Figure 4d, added two additional detectors, one above the radiation source and one below it.

I measured effort to create each geometric description and effort to make changes. For the text-based tools this means counting the number of lines needed to create the original file and number of lines modified or added to make the change. For the GUI- based tool "actions" are measured. An action is a single user input, for example, a mouse drag, opening a pull down menu, or entering a value in a field. The ratio of effort needed to make a change to effort needed to create an original description provides a measure of how easy it is to make a change. For example, there were 110 lines in the pyMGeo source code used to generate the model in Figure 4a and 3 lines were modified or added to add two additional detectors. The resulting effort is measured as follows:

$$\frac{\text{3 lines to make change}}{\text{110 lines in original}} = \text{2.73\% of original effort to make change}$$

Results and Contributions

Table 1 shows the results of my experiment. The second column shows the number of actions or lines needed to create the original description. The fourth, fifth, and sixth columns show the percent of effort required to make each change with each of the three different tools and methods.

Model Name	Number of Actions/Lines in Original	Change to Model	Percent Change in pyMGeo (lines)	Percent Change in VisEd (actions)	Percent Change in MCNP input file
Carbon Activation	pyMGeo: 38 VisEd: 38 MCNP: 10	Carbon Disk Closer to Source	2.63%	15.79%	10.00%
		Vacuum Includes Source	2.63%	15.79%	10.00%
		Enlarge Bounding Cylinder	2.63%	15.79%	10.00%
Gamma Ray Detector	pyMGeo: 110 VisEd: 134 MCNP: 22	Enlarge Sodium Disk	0.91%	4.48%	4.55%
		Source Closer to Detector	1.82%	8.21%	9.09%
		Add Two More Detectors	2.73%	29.10%	22.73%
Vacuum Chamber	pyMGeo: 166 VisEd:136 MCNP: 34	Move Silicon Detectors	1.20%	8.09%	5.88%
		Move Plate Between Flanges	7.23%	22.79%	17.65%
Omega	pyMGeo: 121 VisEd:490 MCNP: 136	Move Entire Chamber	5.79%	65.51%	46.32%
		Remove 58 Ports	47.93%	43.47%	88.24%

Table 1 – Experimental Results

Generally I found that a higher initial investment was needed for algorithmic description, but subsequent changes took less effort. For example, in the vacuum chamber model, the number of lines needed to create the original model was 166 - much higher than the number of lines needed for hand-written MCNP, and number of actions needed in VisEd. However, both changes to the model resulted in a smaller percentage of effort overall in pyMGeo. In absolute terms, adding two additional detectors required modification of 3 lines in pyMGeo, an additional 39 actions in VisEd, and modification of 5 lines in the hand-written MCNP. The three modified lines in pyMGeo instantiated two additional detectors with transformations as well as modifying the surrounding air volume.

Not all models require a higher initial investment with algorithmic description. Figure 6 shows the Omega model, essentially a sphere punctured by sixty port holes. In this model, only 121 lines were needed in the initial pyMGeo description but 490 actions were required in VisEd. This is because pyMGeo allows the use of a loop to iterate over a list of port positions, creating a port description for each. In VisEd each port in the sphere must be entered manually. The second change made to this model involved removing fifty-eight ports from the sphere. This change was implemented by deleting entries from a list of port locations. Changing the bounds the loop iterates over would require even less effort.

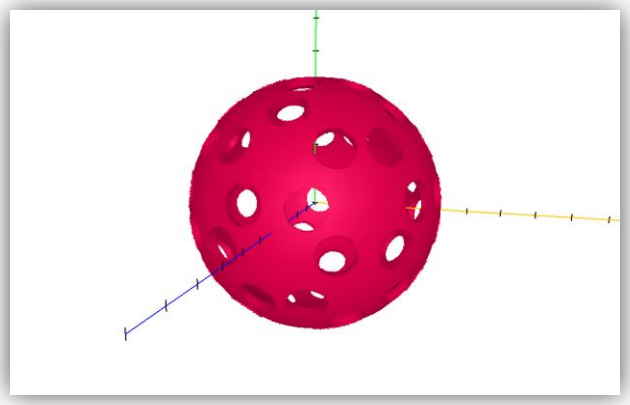

Figure 6 – Omega Model Original

One advantage of algorithmic description of geometry is data abstraction. Storing dimensions and transformations in variables is advantageous in all models. This is not possible with the other as the dimensions must be entered explicitly each time they are used. Furthermore, pyMGeo provides abstractions of common transformations such as rotations and translations, making them easy to create and apply. MCNP and VisEd, in contrast, often require users to apply transformations manually and either enter the resulting coordinates or store the transformation in matrix form. Additional advantages include encapsulation and repetition. Models that include multiple substructures can modularize those substructures and use a loop to generate multiple instances easily.

However, there are cases where the design of a pyMGeo program may not be able to easily accommodate a change to the geometry. As with any piece of software the ease of making a change depends on the original design. This is not true of the other two tools since every change to the geometric description requires a considerable amount of effort.

In the future, it would be interesting to compare algorithmic description tools to more robust GUI-based tools such as Blender. Blender has features that are not included in VisEd, such as the ability to copy and paste pieces of geometry. I suspect that algorithmic description would still be beneficial since subsequent changes made to one piece in Blender may not propagate through all of its copies.

In this experiment I compared algorithmic description of geometry to other popular methods for generating geometry descriptions. The tools and methods I compared shared a common target, MCNP. Using percent of effort needed to make a change to an existing geometric description as a metric I found that algorithmic description had advantages over GUI-based and hand written methods. Results demonstrate that often a high initial investment is required with algorithmic description but subsequent changes to a model require very little effort.

Acknowledgements

Funded in part by the U.S. Department of Energy through the Laboratory for Laser Energetics at the University of Rochester. Special thanks to my advisor, Dr. Doug Baldwin and Dr. Stephen Padalino for the support.

References

1. Adobe Systems Inc.. *PostScript Language Reference*. Third Edition. Reading, Mass: Addison-Wesley, 1999.

2. Baldwin, Doug, *IViPP*, http://cs.geneseo.edu/~baldwin/ivipp/index.html, April, 2012.

3. Blender Foundation, *Blender*, http://www.blender.org/, April 2012.

4. Eitz, Mathias, Ronald Richter, Kristian Hildebrand, Tamy Boubekeur, and Marc Alexa, "Photosketcher: Interactive Sketch-Based Image Synthesis", *IEEE Computer Graphics and Applications*, 31 (6), November/December 2011, 56 – 66.

5. Los Alamos National Laboratory, *MCNP*, https://laws.lanl.gov/vhosts/mcnp.lanl.gov/, April, 2012.

6. Olsen, Luke, Faramarz Samavati and Joaquim Jorge, "NaturaSketch: Modeling from Images and Natural Sketches", *IEEE Computer Graphics and Applications*, 31 (6), November/December 2011, 24 – 34.