A GUI-BASED OPEN-SOURCE PROGRAM FOR VIEWING AND ILLUMINATING ASTEROID SHAPE MODELS.

S. P. Levengood[1] and M. K. Shepard[1], [1]Department of Geography and Geosciences, Bloomsburg University, Bloomsburg, PA 17815, spl06880@huskies.bloomu.edu; mshepard@bloomu.edu.

Introduction: We created an interactive open-source program that can generate brightness maps and phase curves for a given asteroid shape (.SHP) model. Images of the model are rendered using raytracing via OpenGL.

Method: The program uses a raytracing algorithm to generate images of the models. The phase curves are generated by rendering multiple images at different phase angles, calculating the total brightness of each, and then comparing those normalized values.

Shape Models. The shape models that the program imports represent an asteroid as polyhedrons with triangular facets on the surface. A .SHP file contains a list of Cartesian coordinates of vertices followed by a list of facets. These models are assumed to have an arbitray unit scale [1]. The program reads this information and maintains a list of each triangular facet making up the model.

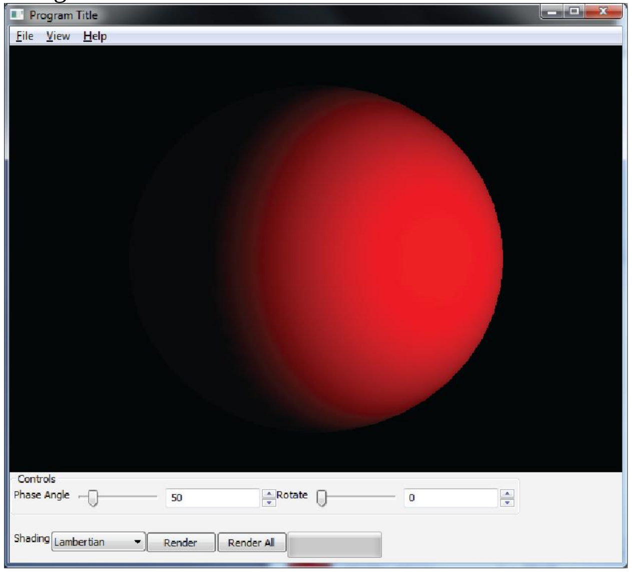
Raytracing Algorithm. The raytracing algorithm projects rays for each pixel from the viewing window toward the model until they intersect one of the triangular surface facets making up the model. If there is an intersection, the algorithm determines the point at which the collision occurs and calculates the surface normal vector of the model at that point as well as the incident angle. The algorithm then calculates a value for the pixel at that ray's origination point.

Ray-Triangle Intersection. To determine whether a given ray would intersect with one of the triangular facets making up the model, the method described in [2] was used.

Calculating Pixel Value. The program allows for different reflectance functions to be used when rendering images. This function is evaluated at every point a ray-triangle intersection occurs. By default the program assumes a Lambertian reflectance model, however the user may define any arbitrary illumination function from within the program.

Implementation: We chose to write this program using a combination of Python and C. The user interface and the majority of the raytracing logic is Python-based, however the main rendering loop and ray-triangle intersection method is written in C for performance reasons. The Numpy library is used to provide fast arrays which are used to store the vertices and surface facets, as well as vector operations required for raytracing. For the user interface, wxPython in combination with PyOpenGL (to provide a preview before rendering) was used. Matplotlib is also used to generate and display the phase curve graphs. Additionally, the Python Image Library (PIL) is also used to (optionally) save the rendered images.

Results: A screen-shot of the program can be seen in Figure 1 below.



Phase Angle 50
Rotate 0
Shading Lambertian

Figure 1 Default view upon starting the program

Before any shape file is imported, it displays a simple sphere object by default. There are options for changing the phase angle, rotating the model, changing the position of the light source, and changing the position of the viewing window. The render button is used to raycast an image with the current settings while the render all button will render the image at each phase angle and generate a phasecurve for that model. The results of this can be seen in Figure 2.

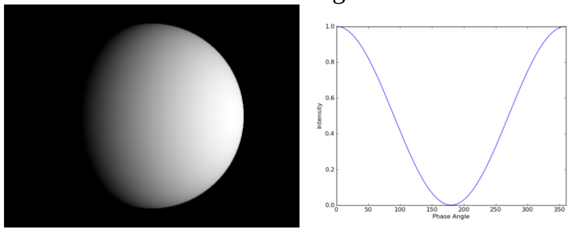
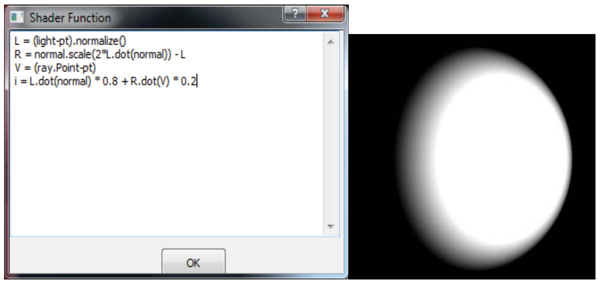


Figure 2 Rendered image from above settings as well as the phasecurve for that model.

As mentioned above, the program can render an image with any reflectance function input at runtime. An example is shown in Figure 3 below.



```
L = (light-pt).normalize()
R = normal.scale(2*L.dot(normal)) - L
V = (ray.Point-pt)
i = L.dot(normal) * 0.8 + R.dot(V) * 0.2
```

Figure 3 Defining a reflectance function at runtime. In this case, a Phong model is used.

Some examples of the program using shape models obtained from DAMIT [1] are shown in Figure 4.

Future Improvements: There are parts of the program which we still plan to improve. The user interface can be improved since not all of the options are easily accessible. The addition of a feature to automatically adjust the rotation for a specific date and time is also a goal. This would make it easy, for example, to predict the apparent plane-of-sky view for an asteroid ahead of a predicted occultation.

There are also performance improvements that can be made by taking advantage of multithreading or using bounding volumes to speed up the intersection code where most of the execution time is spent. Because of the highly parallel nature of raytracing, we believe that this program is also a good candidate for GPGPU acceleration using OpenCL or CUDA.

References: [1] J. Ďurech et al. (2010), *DAMIT: a database of asteroid models*, A&A, 513, A46. doi: 10.1051/0004-6361/200912693 [2] T. Möller and B. Trumbore. (2005) *Fast, minimum storage ray/triangle intersection*, ACM SIGGRAPH 2005 Courses doi: 10.1145/1198555.1198746

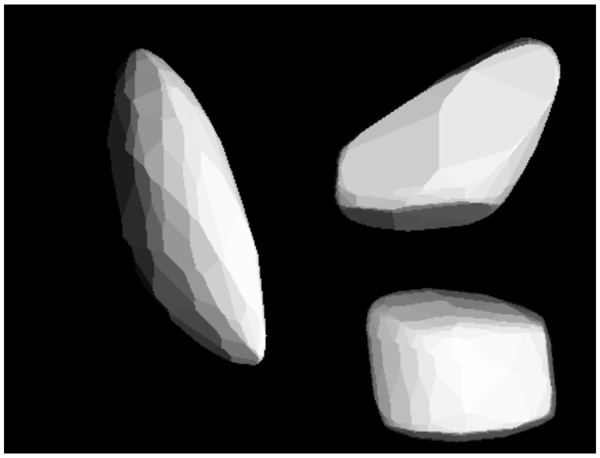


Figure 4 Various asteroid models (obtained from DAMIT [1]) rendered using the program. 1620 Geographos (left), 614 Pia (top-right), and 3908 Nyx (bottom-right).