**Summary**

We have created a parallelized ray tracer in CUDA through the GPU. This ray tracer takes in a scene and outputs an rendered image. Because ray tracing must consider pixel calculations, the algorithm for ray tracing is time consuming. For this reason, the multithreaded parallelism offered by the GPU speeds up this process.

**Introduction and Background**

When we approached this ray tracing program, we first needed to consider several data structures that would be used in the algorithm. The first data structure we used was the “Point” data structure, where each “Point” represented a particular pixel on the screen. This “Point” structure considers the 3-dimensional screen, so as such, each point contains an x, y, and z coordinate. This is necessary for the ray tracing computations, such as rotation, scaling, and translation. (TODO)

After the “Point” data structure, we also have a “Ray” structure that represents a ray of light with that encounters an object. Once the light encounters an object, the distance is recorded and the color of the surface is also stored into the ray. Thus, the “Ray” structure contains a distance member,

and three integer members that represent the intensity of the colors Red, Blue, and Green.

Another data structure that was commonly used in this program was the “Superquadric” structure. This structure contains the a position matrix, orientation matrix, and a scaling matrix, as well as two eccentricity values. The matrices are there to specify size, location, and orientation of the shape on the screen, while the eccentricity values indicate what shape the superquadric is. The superquadric shape is an extension of Piet Hein's Lame curves and it contains two shape parameters, known as the east-west and north-south shape parameters (the eccentricity parameters cover these). With these parameters, the superquadric ellipsoid can mimic the surfaces of several three-dimensional surfaces, such as spheres, cylinders, and cubes. For the ray tracer, the function that will be used to render the image of a superquadric will be the following:

The function “isq” represents an “inside-outside” function for the ray tracer. If the output of the function is greater than zero, then it is outside of the object. If the output is zero, then it is on the surface of the object. Lastly, if the output is less than zero, then it is inside the object. The x, y, and z represent the pixel coordinates that are found through the “Point” data structure. The inputs “e” and “n” represent the eccentricity values of the surface. The “e” represents an “east-west” eccentricity parameter, while the “n” represents a “north-south” eccentricity parameter. The class contains methods such as isq(Point \*) and contains(Point \*) that checks the position of a point with respect to the superquadric.

**Approaches**

TODO:

Languages: C++, CUDA

Machines: UNIX machines, GPUs

Due to a previous graphics laboratory class, code for a ray tracer program already exists, but we are unable to use that code due to the fact that CUDA does not support several of the data structures that the CPU supports. An example of this is STL, or the Standard Template Library. Any data structures that are in this library are not CUDA supported, so a challenge was to work around this and to use other data structures that are CUDA supported for parallelization to occur.

Another challenge that we had to overcome was the fact that CUDA does not support virtual classes and pointer references as well. When memcpying from the host to the device, the pointer reference may be lost. Therefore, we had to work around the original code by refactoring it to not include virtual functions and any pointer references in any of the classes. A consequence is that the optimization of the ray tracing decreases.