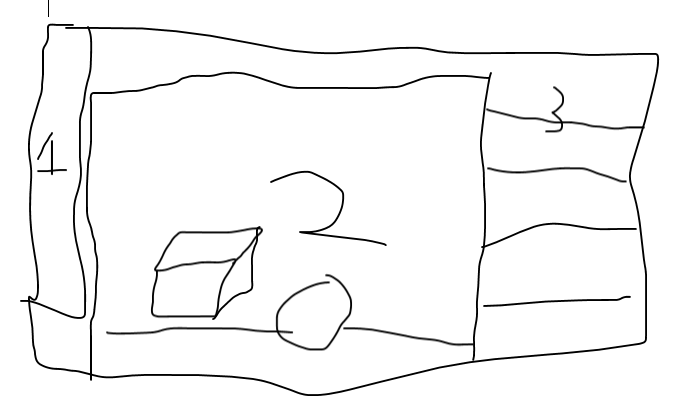
Raytracer with Rasterizing Editor

Overview:

 “This” will be a rendering engine used to create scenes and render them using basic raytracing technology. The basic user interface will look like a piece of software with a window holding a rasterizing rendering engine. Along the edge there will be a section holding tools and options for objects in the scene. The rasterizing engine is there for the user to be able to create scenes quickly with low quality so they can organize things how they want to look. The editor will create the model of the scene as defined by the user and then pass that data to the raytracing engine which will then render the image at a higher fidelity.

1: The tools window (move, rotate, scale etc.)

2: The editor window (allows for user manipulation of objects)

3: The options window (lets the user set options like material properties)

Technologies used:

* The editor will be written in C# using WPF (or UWP maybe). It will read in models from OBJ files and textures from various formats, primarily JPG and PNG.
* The editor will save the data in the scene out in JSON format to be read in by the raytracer.
* The editor will use a rasterizing engine written in C++ using SDL, OpenGL, and GLM to allow for quick edits to the scene.
* The raytracer renderer will be written in C++, using CUDA/OpenCL to render faster (using parallel processing on the GPU).
* Rapidjson? for reading and writing the JSON
* ASSIMP to load obj files into both engines
* Stb\_image to load images into both engines
* GLM for raytracer as well?

Biggest Problems to Solve

* Putting the rasterizing engine into the C# WPF container, since they are written in different languages, and will use different threads at the least.
* Implementing a raytracer
* Raytracing models
* Reading in models correctly for them to be raytraced
* Learning CUDA/OpenCL
* Learning WPF
* In the raytracer, CUDA/OpenCL require certain functions to be defined as host or device functions. Basically, will it be possible to load in data from a JSON file since that will be done on the CPU? I’m 90% yes on this, since data does not get marked as host or device, and in the example I have, the data is calculated at runtime using random functions… through a global function?? Does that mean it is on the GPU?
* The two applications/programs will probably communicate very little. The C# program will most likely just start the raytracer application with command-line arguments that point to the JSON file of the scene to render, and the output file format and location.