Project Plan

Project Abstract

We will attempt to implement physically based rendering via ray tracing to generate an accurate render of an object. For the sake of time, we will focus on only a few properties of light and one key object in a scene (none or very few background elements). We will not be baking normals or supporting material properties that are typically seen when creating materials in a software such as substance designer or Blender. However we may make triangulated 3d meshes that are exported from Blender.

Application and Application Domain

The application from which this problem originates is ray tracing in video games. This belongs to graphics programming applications and we believe is similar in terms of high level principles to collision detection. This is significant for simulations or realistic modeling.

Algorithms

We plan to use a collision detection algorithm to determine whether and where a light ray hits an object. We either plan to use ray hit detection with a triangle's plane (if we use triangulated meshes) or a more hard-coded mathematical model based on the object geometry we are dealing with if we do not use triangulated meshes. Light rays will likely start from the camera and use their reflection to compute how intense the light is at that point.

Implementation Strategy

We predict that we may have to parse and read an .obj file and store its triangles in memory. We are anticipating that we will have to code some type of gaussian elimination due to needing to solve systems of equations. This can be easily parallelized. An alternative might be to use determinants via co-factor expansion for only small matrices (due to their n! computation time, we cannot use large matrices). We also will be heavily dealing with dot products of 3-dimensional vectors, and while this can be parallelized we anticipate that the computation speed will largely be impacted by the number of dot products we are performed rather than the complexity of each individual dot product. If we use an equation based object such as a parametric sphere, we anticipate using hard coded cross products of partial derivatives to get the surface normal equation. Furthermore, we will neeed to calculate the lights irradiance which will involve some form of integration either using sums which can be aided with a technique such as prefix sum or using a more complex method such as Monte Carlo integration which will still require some parallel reduction step.

Anticpated Obstacles

If we follow through with our plan to implement support for custom models or materials, we may encounter issues depending on the material type. We first are planning to mitigate this by not including support for such complexity until we have a working version of our project on a simple, non-imported asset (such as a sphere made with C code). Since there are several steps to the ray tracing process, we will have to decide which steps are most important and which ones we can afford to leave out.

Project Timeline

By the checkpoint, we expect to implement some form of backwards path traced light ray detection for hitting objects. We also plan to start with implementing ray tracing for solid (non-metallic) objects with standard normals. We may add metallic objects later, but not first as they require solid objects to judge how the ray tracing is working. We also plan to start with one fixed point-light source. After the light ray collision detection, we will have to parallelize the algorithm to compute the the intensity at the point. If time permits, we can add more features to the rendering process such as material properties such as glossy/metallic objects, microfacets (e.g. Cook-Torrance model for specular highlights), anti-aliasing, and Ambient Occlusion for better shadow effects.