CSE168 assignment 1

Alisha Lawson, Hallgeir Lien

1 Images

We included 3 images in the zip file.

1.1 reflection.ppm

This image includes 5 objects; the teapot is fully reflective, the plane and one of the spheres are texture mapped with the bump mapped stone texture, a partially reflective with a yellowish diffuse color, and finally a refractive blue sphere where we used a refractive component of (0, 0, 1).

1.2 refraction.ppm

This image shows nine fully refractive spheres with increasing refractive indices. Starting at 1.0, which appears to be invisible because there's no refraction, it increases in steps of .05 from left to right, ending at 1.4 in the top right corner.

1.3 bunniez.ppm

Here we have three bunnies facing in different directions. We built a transformation matrix to achieve the positioning and rotation. The one to the far left is fully refractive, the middle one is fully reflective and the right one is diffuse.

2 Textures

We created a shading class that inherits from the phong shader that looks up the diffuse color in a texture instead of a constant color. That shading class looks up that diffuse color from a Texture object that it receives when it's constructed. There are two lookup functions - one for 2D and one for 3D textures.

3 Hacker points

3.1 SSE

We implemented the triangle intersection in SSE. We store the vertices in an array in the TriangleMesh class so that we don't need to do load operations for every test. Performance wise, we saw about 10% improvement in rendering speed when rendering one bunny.

We ran the rendering 5 times with SSE and 5 times without on 16 threads and took the fastest time from both tests. The fastest with SSE was 5 minutes, 38.34 seconds and the fastest without was 6 minutes, 14.72 seconds, which gives a speedup of 1.11.

3.2 Environment mapping

We use a sphere mapped HDRI envionment map. We created a global tone mapping function, where c is the input intensity from the HDRI image and I is the maximum intensity in the image, $f(c) = min\{1, 4 \cdot (c/I)^{0.3}\}$. The number 0.3 and 4 were found by experimentation to work well with the image.

4 Extras

In addition to this, we implemented bump mapping. Tangents are calculated by taking the largest component from the normal, inverting it, shifting it one position and zeroing the other components. The cross product is then taken to get tangent 1, and the cross product between the normal and tangent 1 is taken to find tangent 2.

Second, we implemented multithreading using OpenMP. To run with multithreading, compile it with the compile time define OPENMP defined (in GCC, compile with -DOPENMP).

We also implemented normal averaging. The averaging is done as the model loads, by building a neighbor list containing the normals for each vertex and finally averaging them once everything has been loaded.