CSC2504: Computer Graphics Assignment 3

Jacob Ritchie (999771983/ritchi30) and Oliver Straszynski (999464379/straszy2)

December 9, 2017

1 Structure of Submission

We added files `octree.cpp` and `octree.h`, which contain the code for implementing the octree data structure and the associated hash table helper data structure. The remainder of the code has been added to the existing files. The makefile has been modified to include the new files, and compiles and runs on the CDF machines.

All code is contained within the directory `a3/raytracer`. For each of the features below, we have indicated the location of all the code used in implementing that feature.

Our image files can be found in the directory `a3/raytracer/images`. The six images for part A (`sig1, sig2, diffuse1, diffuse2, phong1, phong2`) are located directly in the a3/raytracer/images directory The images are separated into folders based on the features they display and are named accordingly.

When the raytracer is run (AFTER REPLACING THE FILE MAIN.CPP WITH MAIN_MODIFIED.CPP), it produces images showcasing the test scene and each of the implemented features in the directory `a3/raytracer/output_images`. These images are also included in the report (report images were generated at high resolution, except the figure for comparison of anti-aliasing vs. no anti-aliasing).

2 Implemented Features

All of the features for part A (scene signature, ambient and diffuse components of Phong illumination, complete Phong illumination) as well as recursive raytracing (shadows and reflections) are shown in Figure 1.

2.1 Shadows

For each ray's point of intersection, we create a new ray from the intersection point to the position of each light source. If this ray intersects an object before reaching the light source, we consider the ray to be in shadow and add only the ambient component of the Phong illumination model to the color of that ray.

Files: `raytracer.cpp (Raytracer::compute_shading()), light_source.cpp (PointLight::shade())`

External resources used: Textbook [1]

2.2 Reflections

Rays which strike a surface are "reflected", with the direction of reflection calculated as given in the textbook. By default the maximum recursion depth is set to 10 reflections, but this can be changed using the Raytracer's `reflection_depth` class member. The specular component of the material was used as the coefficient of reflection, (as detailed in the texbook, pg. 212 [1]). An example of multiple recursive reflections is shown in Figure 3.

Files: `raytracer.cpp (Raytracer::shade_ray()), raytracer.h, util.h (Ray3D)`

External resources used: Textbook [1]

2.3 Antialiasing

To generate an anti-aliased image, multiple rays are traced for each pixel and the colour of each ray is then averaged to determine the final pixel colour. We used the jittering scheme described in the textbook[1], where the origin of each ray is selected from a grid of size `N x N` centered at the position of the pixel in the image plane, but the position of each sample in the grid is jittered by a small random quanitity. By default N is set to 5, resulting in 25 samples. Antialiasing can be turned on by setting `raytracer.antialias = true`. A comparison of the default scene with and without anti-aliasing is shown in Figure 2.

Files: `raytracer.cpp (Raytracer::render(), Raytracer::traceRay())`

External resources used: Textbook [1]

2.4 Area Light Sources and Soft Shadows

We allow the user to define parallelogram-shaped area light sources from a corner point and two edge vectors, resulting in soft shadows. We use the same jittering scheme that was used for anti-aliasing to generate soft shadows, sampling from the skewed grid defined by the shape of the area light source. By default we average 100 samples per light source. As detailed in the textbook, we generate two independent sets of sample points - one to compute illumination and a second to compute shadow rays [1]. A comparison of the default scene with and without soft shadows is shown in Figure 2.

Files: `raytracer.cpp (Raytracer::compute_shading()), light_source.cpp (AreaLight::get_samples, AreaLight::shade, shuffle_array), light_source.h, util.h (Ray3D)`

External resources used: Textbook [1]

2.5 Environment Mapping

Environment Mapping was implemented using cube maps. If a ray does not interesect with an object, we calculate the cube map face and UV coordinates associated with that ray's direction. The formula for implementing cube maps was taken from the textbook and Wikipedia [1,2]. For each face, we store an associated a .bmp image. Cube Maps can be turned on by setting `raytracer.use_cubemap = true`. Two sources of images [3],[7] were used to generate .bmp cube maps. A scene using the images from [3] is shown in Figure 4. (Note: because of the high-resolution of image used, at low output resolutions these images are very noisy unless anti-aliasing is used).

Files: `raytracer.h/raytracer.cpp (CubeMap class and associated functions, Texture class and associated functions)`

External resources used: Textbook[1], Wikipedia [2], Cube Map Images [3], [7]

2.6 Texture Mapping

UV texture mapping was used to map textures to the surface of spheres and planes. A simple procedural checkerboard texture was implemented during development. Textures can be loaded from .bmp files using the code in `bmp_io.cpp`. Figures 5 and 6 show examples of textures applied to different images.

Files: `raytracer.h/raytracer.cpp (Texture class and associated functions, SceneDagNode, Raytracer::addTextureInfo()), scene_object.cpp, util.h (Ray3D, Intersection)`

External resources used: Textbook [1], Wikipedia [4], Textures [5], [8], [9]

2.7 Octree

We followed the octree implementation detailed by Glassner in his paper "Space subdivision for fast ray tracing" [6]. The octree was generated by recursively subdividing non-empty regions of the scene into 8 voxels of equal size until each voxel contained only a single object. To determine whether an object is inside a voxel we use the object-aligned bounding box.

A hash table was implemented and each voxel given a unique id according to Glassner's numbering scheme. The hash table entry was a linked list of the objects contained in that voxel. We used less efficient but simpler method than Glassner to advance the ray to the next voxel.

We tested the octree implementation using a large scene containing 28 spheres and a single point light source. Maximum depth for recursive reflection was set at 10 reflections. The scene took 4.4 seconds to render using the normal traversal algorithm, and 1.9 seconds using the octree (at a resolution of 500x500 pixels). The resultant images are identical and are shown in Figure 7.

The octree can be turned on by setting `raytracer.octreeUsed = true`. Note: the octree was not used when implementing the rest of the advanced raytracing features, and may not work in conjunction with them.

Files: `octree.cpp, octree.h (Entire file), raytracer.cpp, raytracer.h (Raytracer::octreeShadeRay(), Raytracer::traverseTree(), Raytracer::extendRay(), Raytracer::findStartNode(), Raytracer::getQuad()), scene_object.cpp, scene_object.h (UnitSquare::octree_intersect(), UnitSphere::octree_intersect())`

External resources used: Glassner's paper [6]

3 Contributions

We began collaborating after both group members had individually completed part A.

Jacob wrote the code for features 2.1-2.6.

Oliver wrote the code for the octree and aided in debugging reflections, shadows and anti aliasing.

4 External Resources

[1] P. Shirley and M. Ashikhmin, Fundamentals of computer graphics, second edition. Wellesley, Mass.: A K Peters, 2005.
[2] "Cube mapping", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Cube_mapping. [Accessed: 04- Dec- 2017].
[3] "Environment Mapping", Mcs.csueastbay.edu, 2017. [Online]. Available: http://www.mcs.csueastbay.edu/ tebo/classes/4840/DiscreteTechniques/Page7.html. [Accessed: 04- Dec- 2017].
[4] "Texture mapping", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Texture_mapping. [Accessed: 04- Dec- 2017].
[5] "Planet Earth Texture Maps", Planetpixelemporium.com, 2017. [Online]. Available: http://planetpixelemporium.com/earth.html. [Accessed: 04- Dec- 2017].
[6] A. S. Glassner, "Space subdivision for fast ray tracing," in IEEE Computer Graphics and Applications, vol. 4, no. 10, pp. 15-24, Oct. 1984. [Online]. Available: http://ieeexplore.ieee.org/document/6429331/. [Accessed: 04- Dec- 2017]
[7] Paul Bourke. "Converting to/from cubemaps", paulbourke.com, 2017. [Online]. Available: http://paulbourke.net/miscellaneous/cubemaps/ [Accessed: 04- Dec- 2017].
[8] Sonic More Music. "The National released a new track called Sunshine On My Back" [Online]. sonicmoremusic.wordpress.com, 2015. [Online] Available: https://sonicmoremusic.wordpress.com/2015/04/02/the-national-released-a-new-track-called-sunshine-on-my-back/ [Accessed: 08- Dec- 2017]

[9] Marc Hogan. "The National Tout New Album ‘Trouble Will Find Me’ With Trompe L’oeil Cover Art" [Online]. SPIN, 2013. [Online] https://www.spin.com/2013/03/the-national-trouble-will-find-me-title-cover-art-track-list/ [Accessed: 08- Dec- 2017]

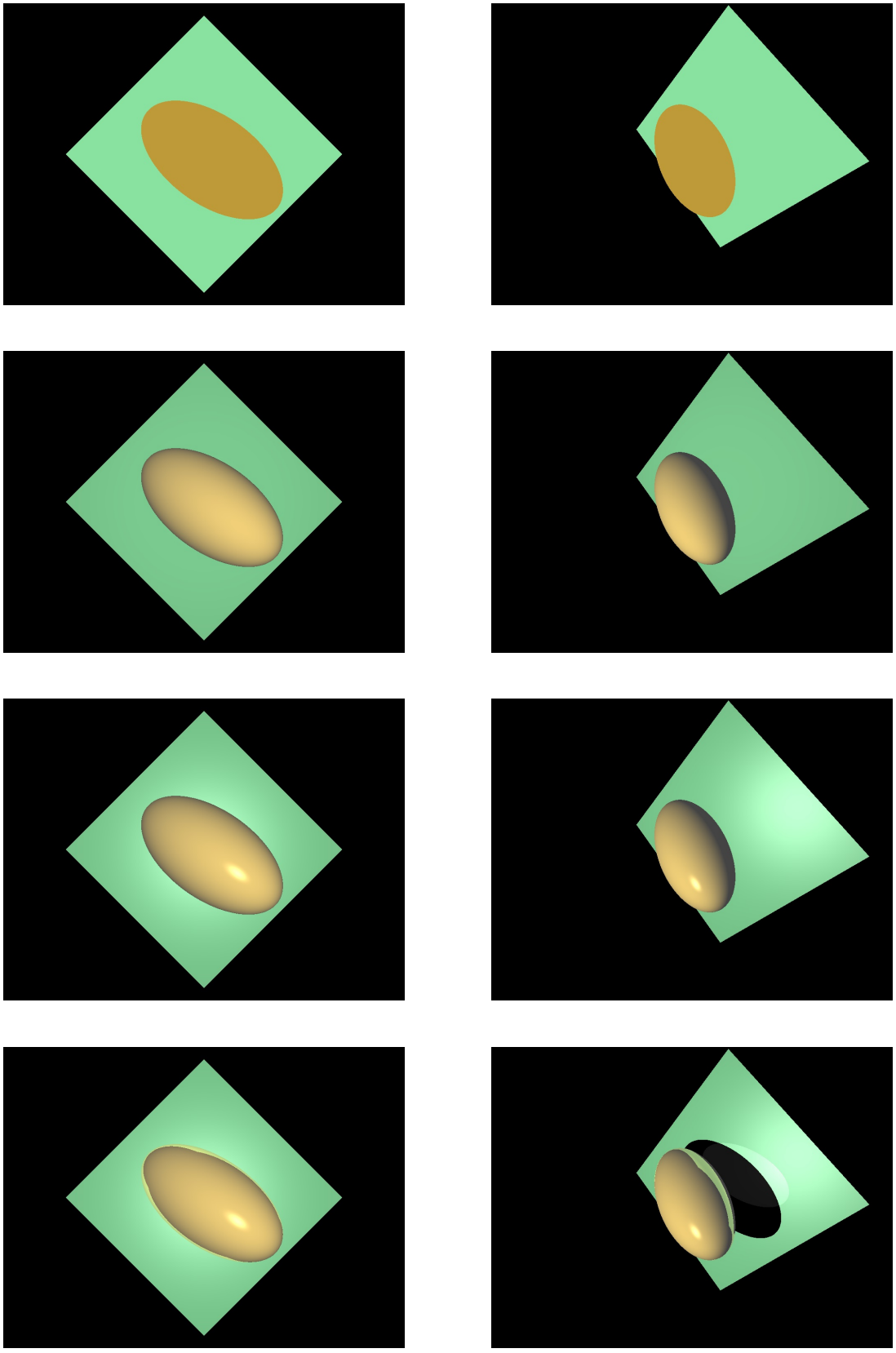

Figure 1: From top to bottom: (a) Scene Signature (b) Ambient and Diffuse Phong Illumination (c) Complete Phong Illumination (d) Diffuse and Specular Illumination with Shadows and Reflections (Recursive Raytracing)

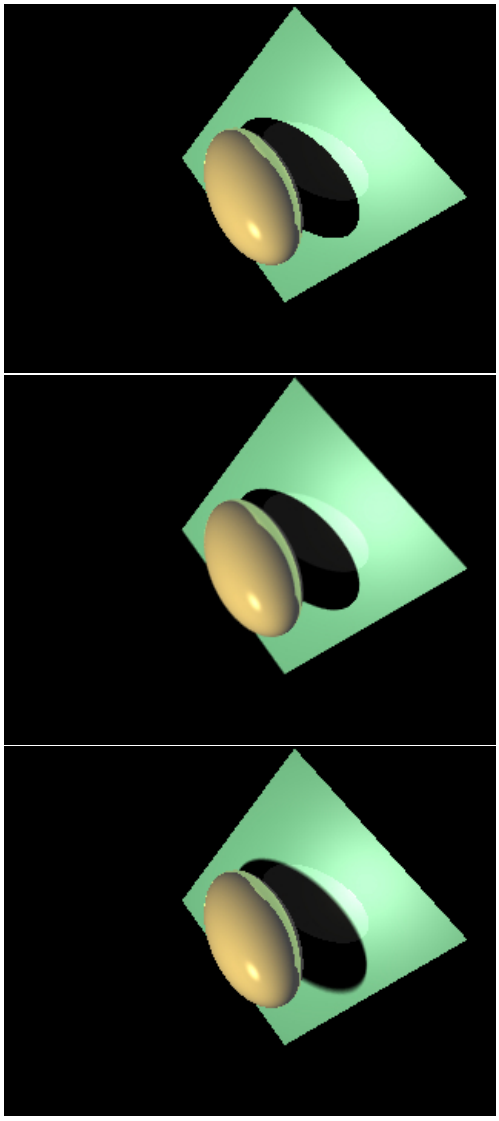

Figure 2: From top to bottom: (a) Recursive Ray Tracing (b) Anti-Aliasing (c) Soft Shadows

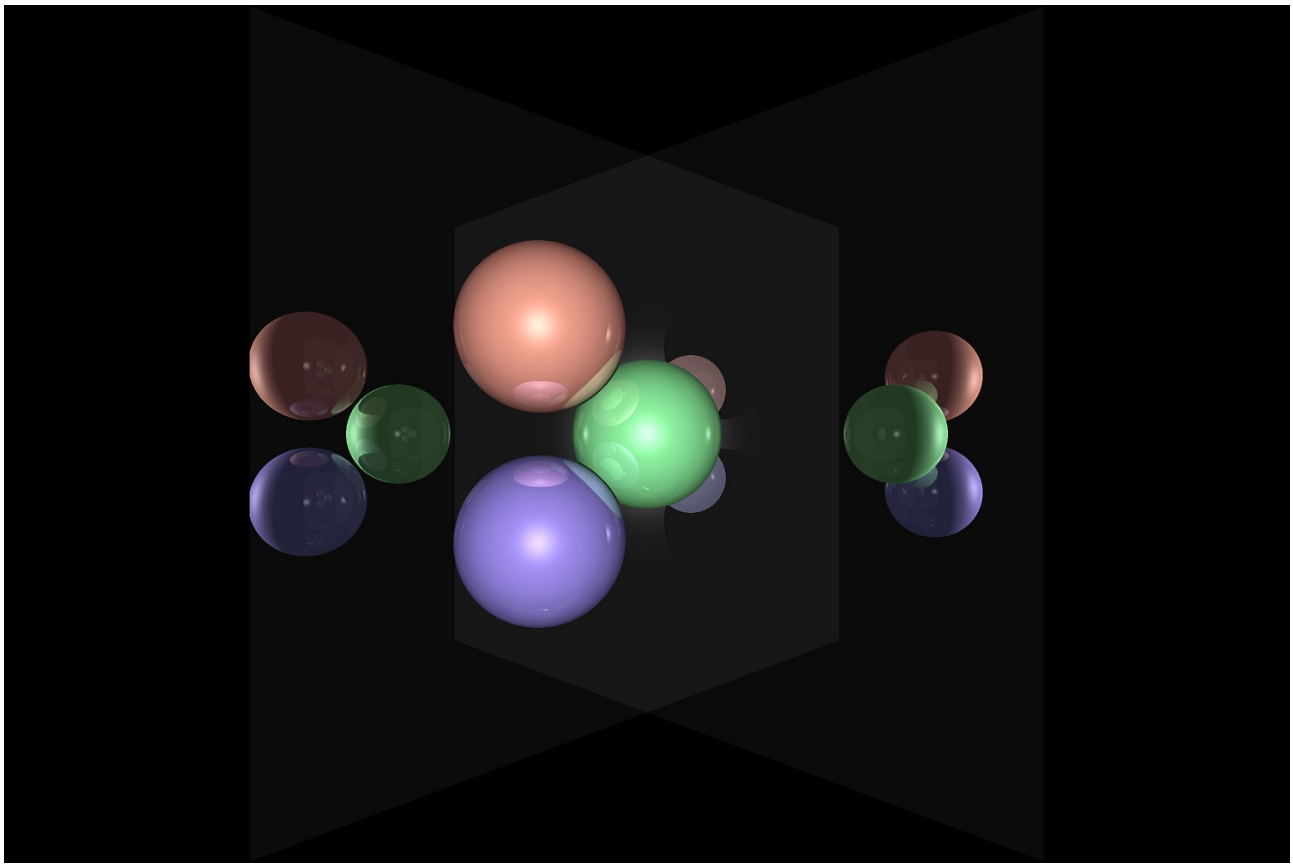

Figure 3: A complicated image showing multiple recursive reflections

Figure 4: Cube maps

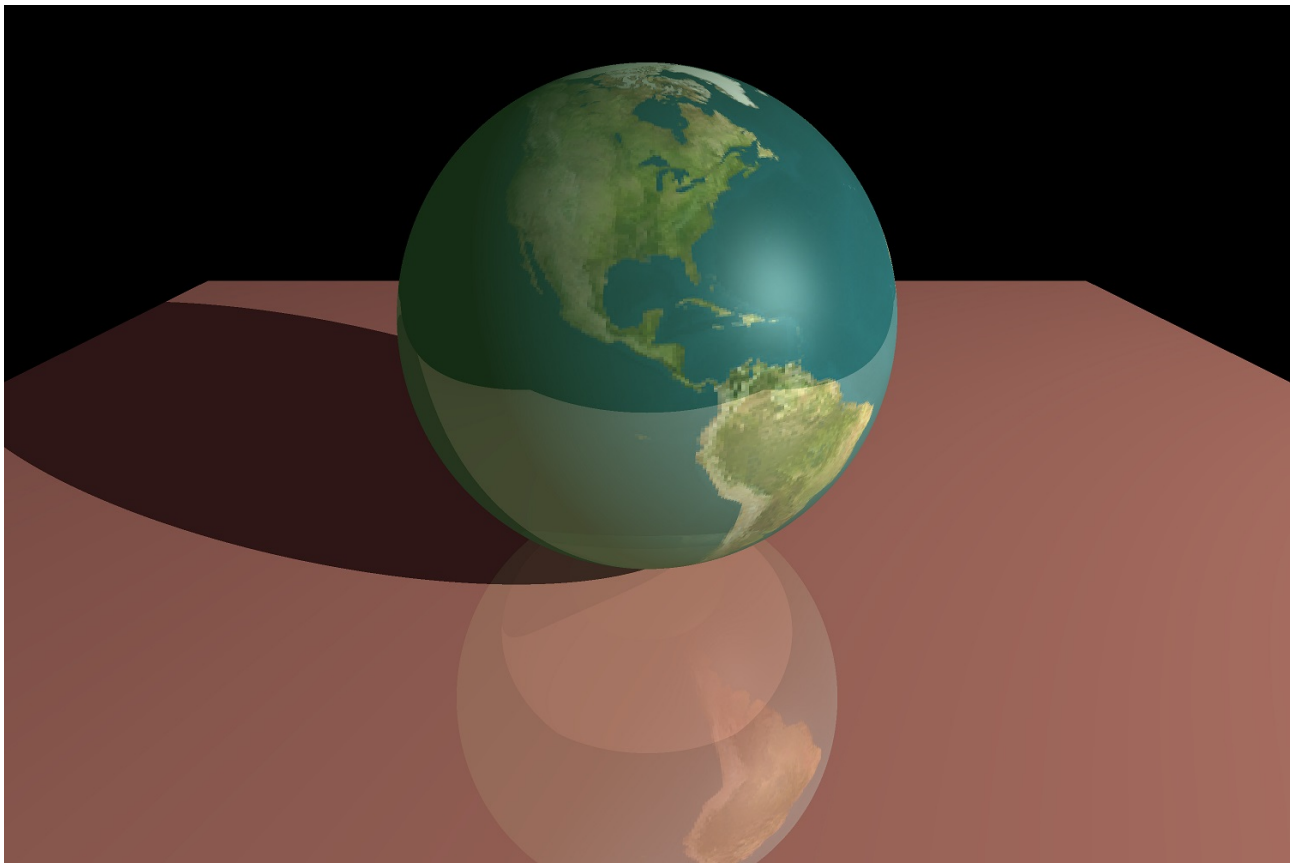

Figure 5: Image showing textured sphere

Figure 6: An image showing two textured planes of different dimensions

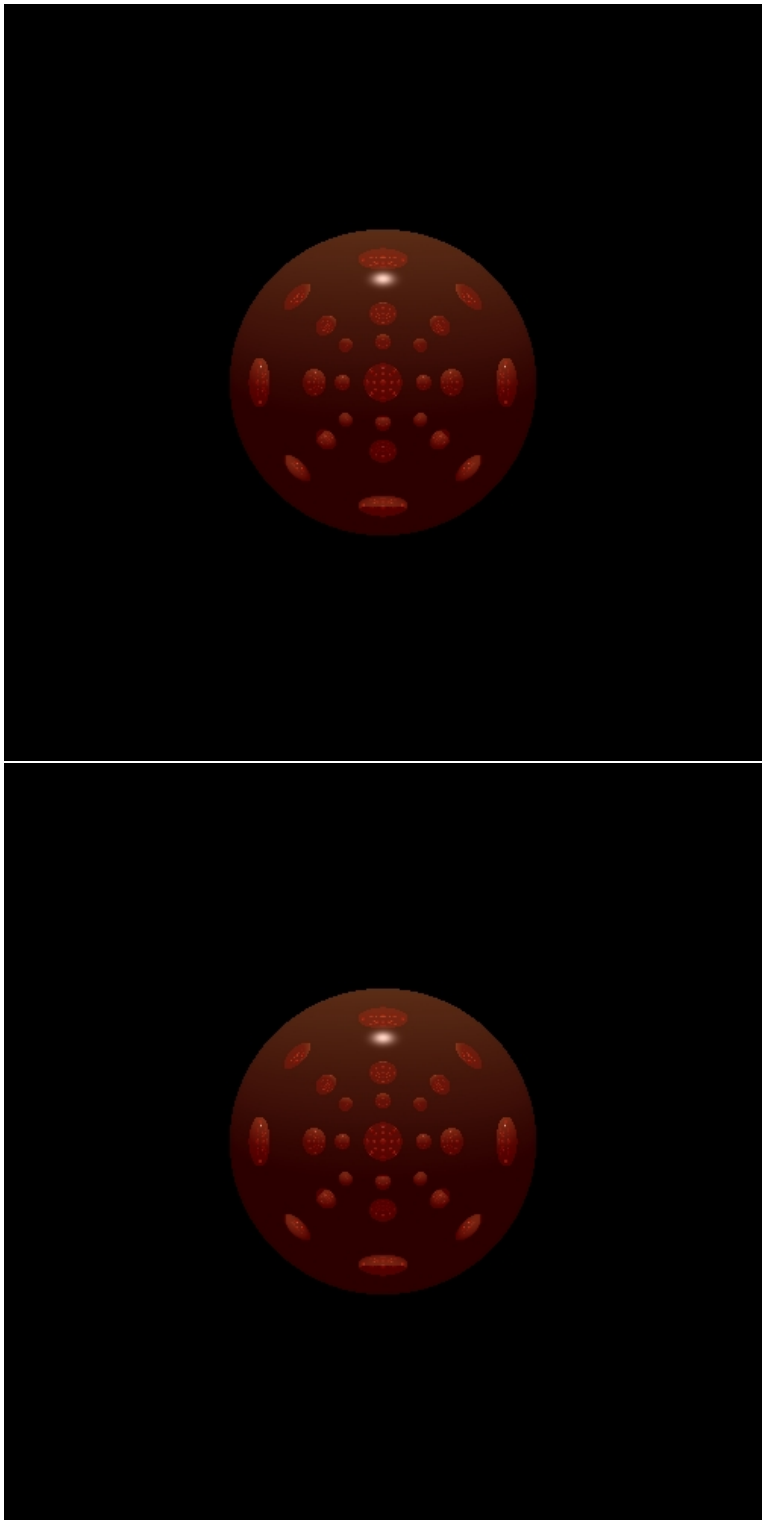

Figure 7: A large scene (28 spheres) generated with an octree (above) and with no octree (below). With the octree, the scene was generated 2.3x faster.