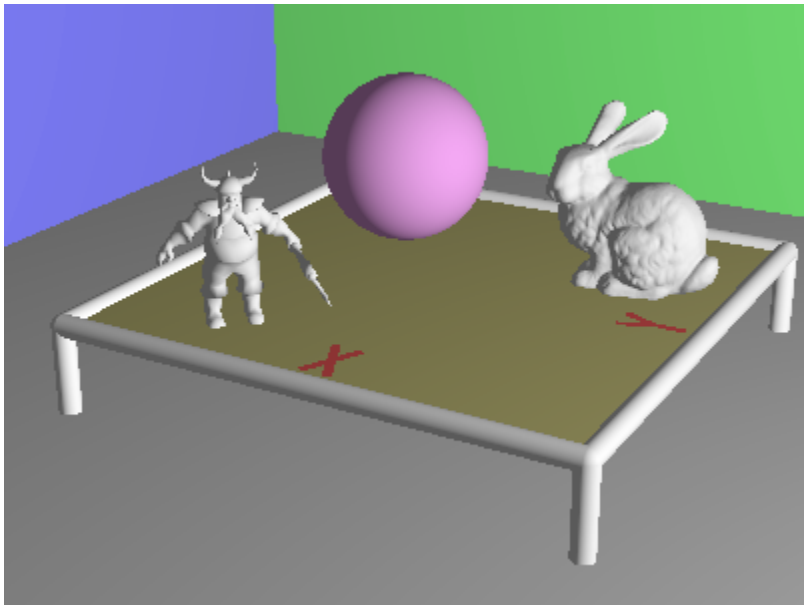
CS500
Project 1
Dylan Washburne
MS CS DigiPen

A different scene can be generated by adding an argument to the console while starting up the executable. If no path is provided, it will automatically search for “testscene.scn”.



The scene consists of 4 fundamental shapes: sphere, box, cylinder, and triangle.

While in standard graphics a scene is constructed entirely of triangles, that data format functions largely as a sort of shorthand that is friendly to the real-time nature of the process. Ray tracing does not have a need to take any such shortcuts for regular mathematical structures, and can render a perfect sphere or cylinder.

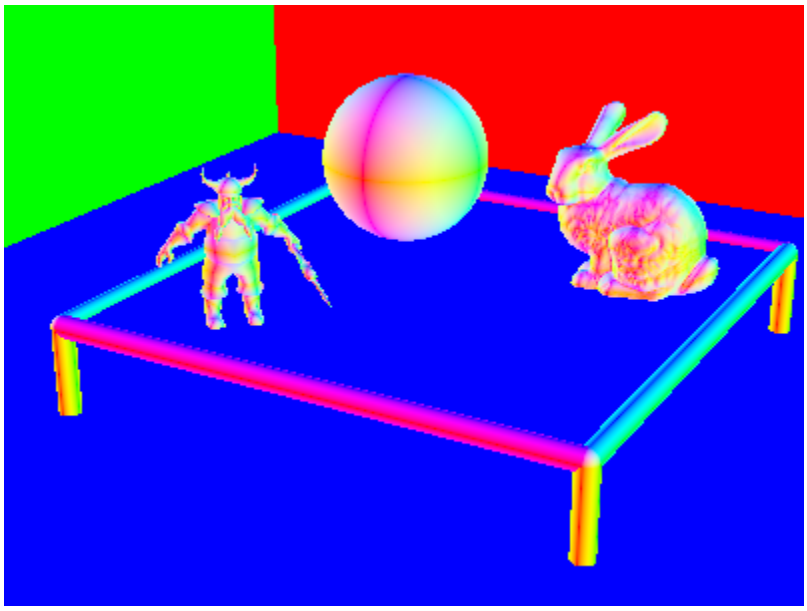
The algorithms required for each shape were each built with guidance from the provided “intersection” handout. Spheres check if the ray passes within a certain distance from the sphere’s center, triangles check the angle of their plane and whether the ray passes between the 3 edges, cubes check whether the ray passes the x y and z in a sane order such that it enters prior to exiting, and a cylinder combines the `interval` work of a cube with the circular work of a sphere.

All objects were tested and output with their depth, normals, and diffuse values to ensure that the scene was constructed pristinely. This was later combined to simulate low-effort Phong lighting (as seen above), more as a proof of concept than as a photorealistic evaluation of the scene.

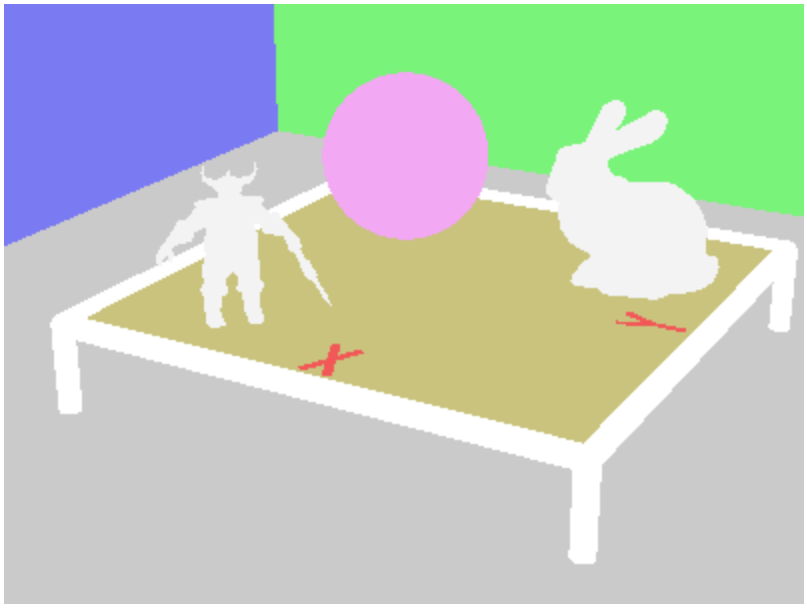
After the scene could be accurately rendered with all fundamental objects, an accelerator `bvh` was implemented. This library consists of many functions and algorithms which frankly I do not understand at the lowest level, but I understand enough at the high level to make it work within the confines of the scene. This library assigns every object in the scene a bounding volume, and performs a precomputation step which arranges all of these bounding boxes into a hierarchy which is friendly to rays attempting to pass through the scene.

Instead of every ray having to test itself against the 70,000+ objects in the scene, this accelerator limits the computation tests to only objects in the ray's immediate path, which may be as few as 1-3 objects.

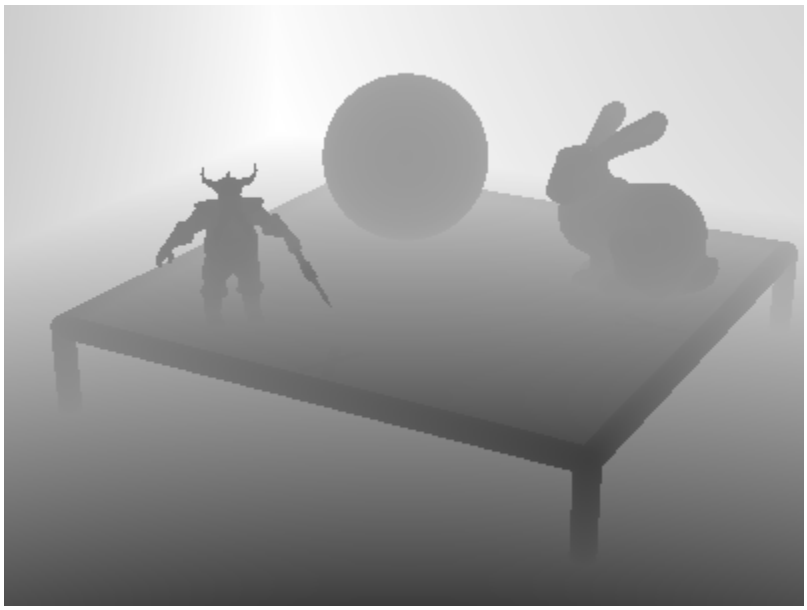
On my machine, the scene took just under 3 minutes to render. With bvh added, it added 12 seconds of precomputation time, after which the scene was rendered in under 1 second.



Surface normals (absolute values). Important so that in future projects, the ray can accurately calculate an angle of reflection.



Diffuse colors. Assigns color values to the rays to be used in lighting, and in future projects, can be added to allow light to bounce onto other objects around the room for realistic coloration for a physical environment.



Depth values. In this project it shows what object is the closest to the camera, but moving forward it will also be able to show which object is the closest to the reflection point and angle the ray is bouncing from. Additionally, can be used to extract world coordinates with a simple calculation using the ray's origin and angle.