Raytracing!

A Math 424 Final Project by Brennan Seymour

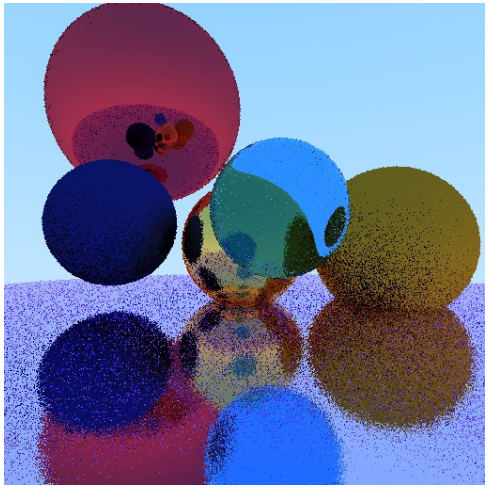
How does it work?

- Raytracing is a rendering technique – that is – it’s goal is to generate a 2d image of a 3d scene.
- There are other ways of doing this, but a raytracer simulates natural light transport by tracing rays of light.
- This is very computationally expensive, but looks *really good*, since its process is built off of real physics.

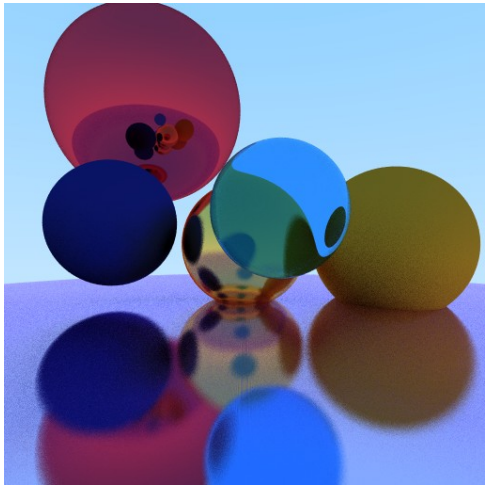
The Process

- For every pixel, a corresponding ray is sent out of the camera.
- To determine where the ray collides, *every object* in the scene is checked to see whether this ray collides with it. The closest collision is chosen as the ray's actual destination.
- Then, we figure out a color and reflection from this collision, based on the material of the object we hit.
- This is repeated until we intersect with a light source – in this case the only light source is the background.
- All the colors we've intersected with are attenuated together to build the final color of our pixel.

Graininess!?!?!



One ray per pixel

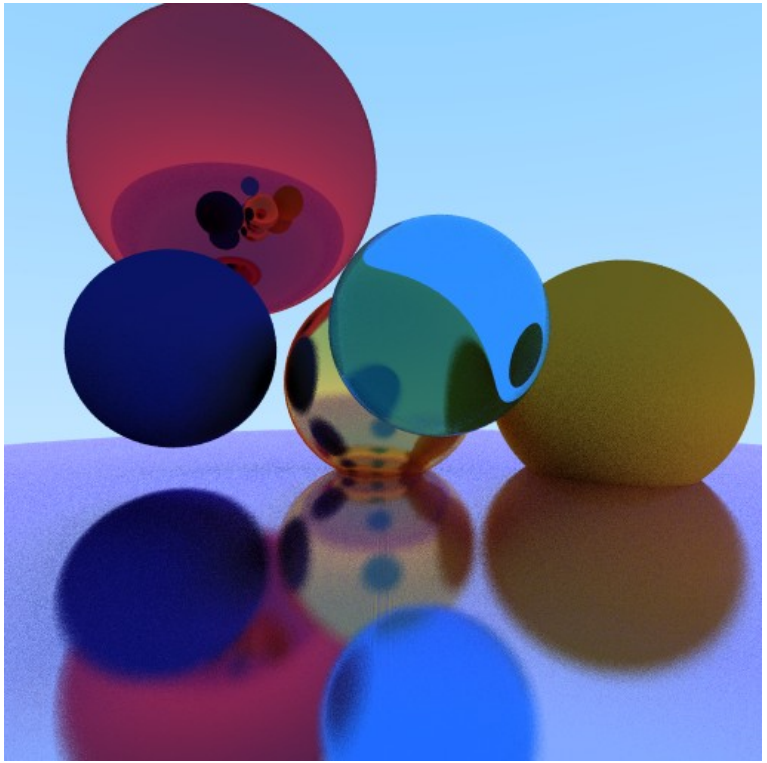


64 rays per pixel

- As you can see, randomness introduced in our reflections causes an awful graininess.
- This can be remedied by casting many slightly-altered rays per pixel and averaging the results.

Materials

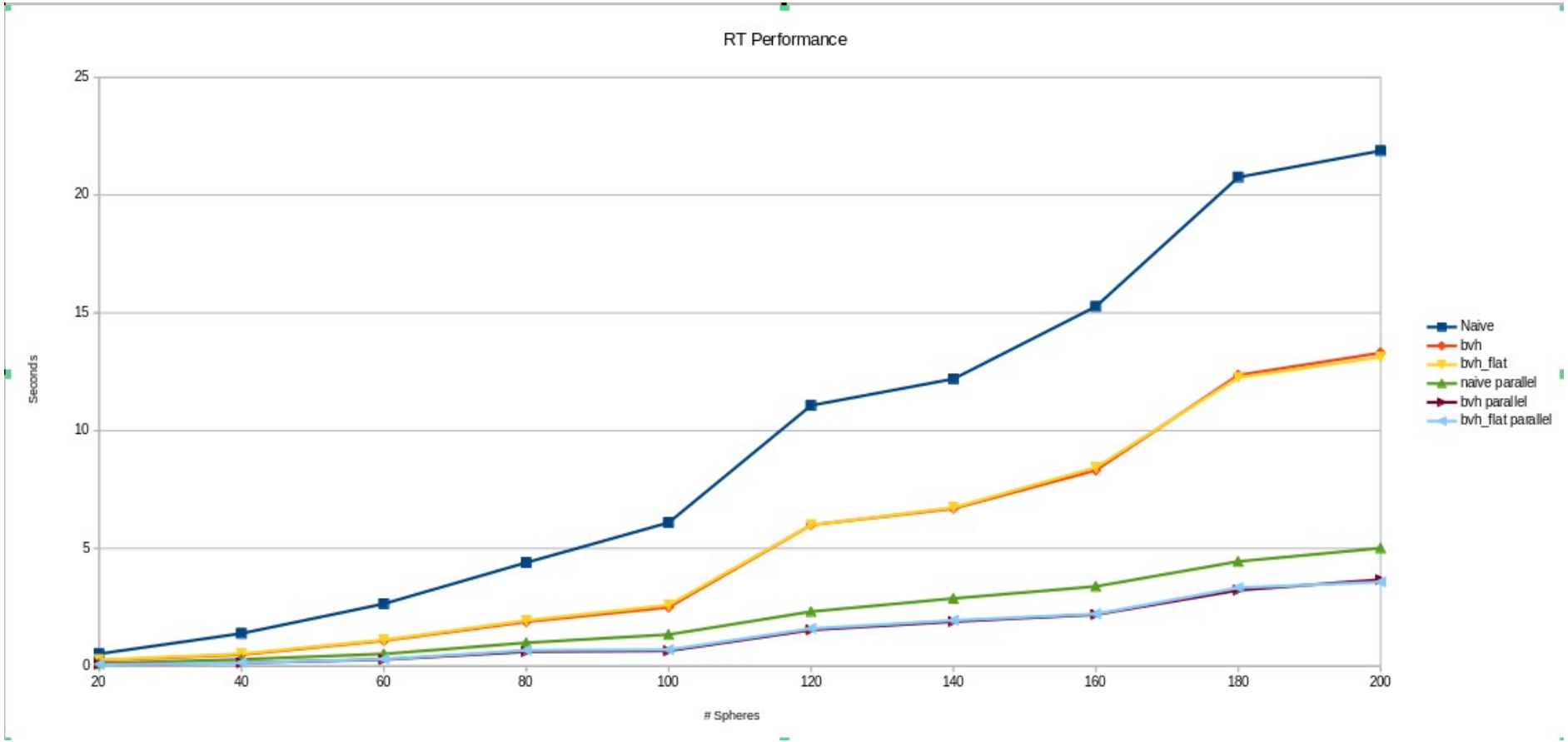
- Diffuse – reflect light in fairly random directions. Results in a matte surface.
- Specular – reflect light directionally. Add a bit of perturbance for a metallic sheen.
- Dielectric – reflect or refract based on a Schlick approximation. Refracts (bends) light going through it.



Bounded Volume Hierarchy

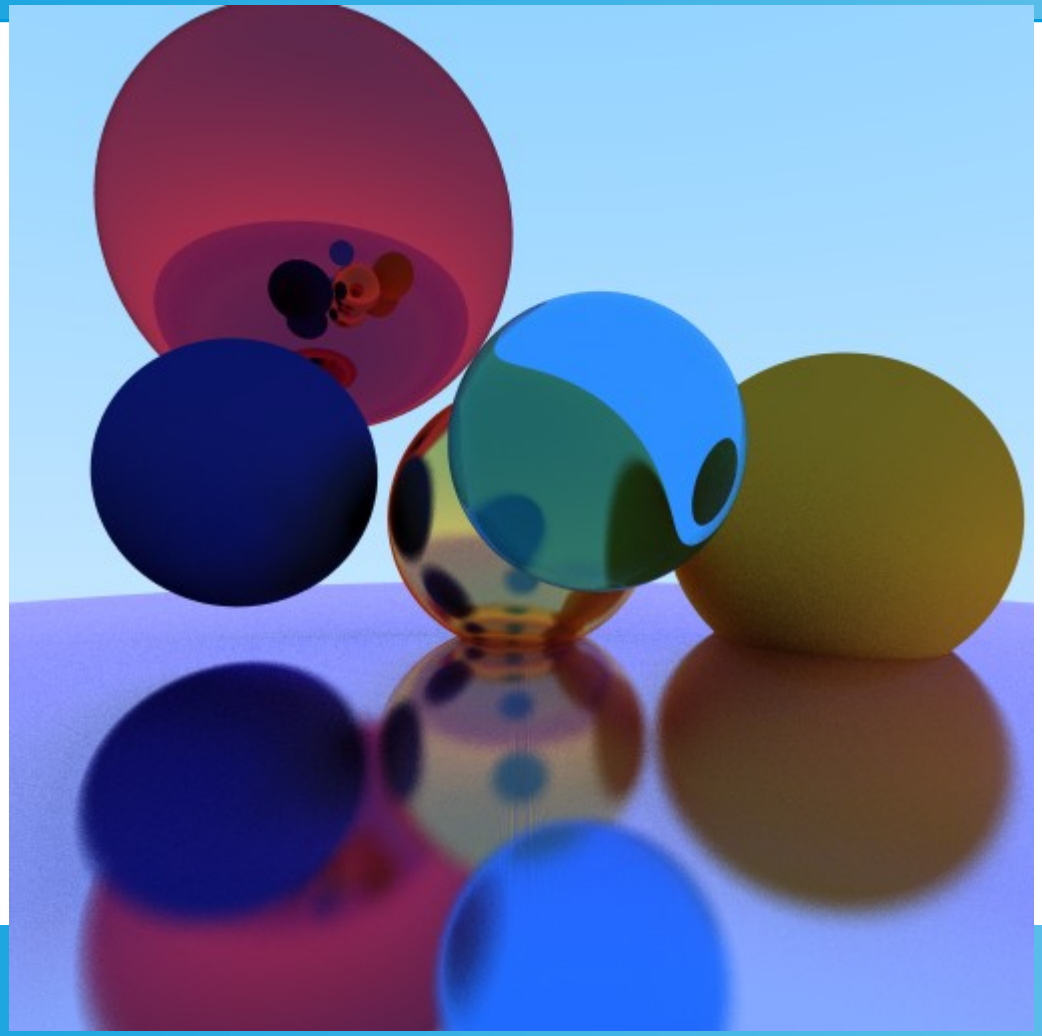
- Full disclosure: I've written a raytracer in the past.
- This time around, I've added a novel feature to speed up the engine with high object counts – Bounded Volume Hierarchy!
- Instead of checking each ray's collision with *every* object in the scene, we can partition the scene recursively into bounding boxes.
- Then if we don't collide with a given bounding box, we can safely ignore it and all its contents.
- If I have time at the end, I'll do some whiteboard drawing about this.

Performance

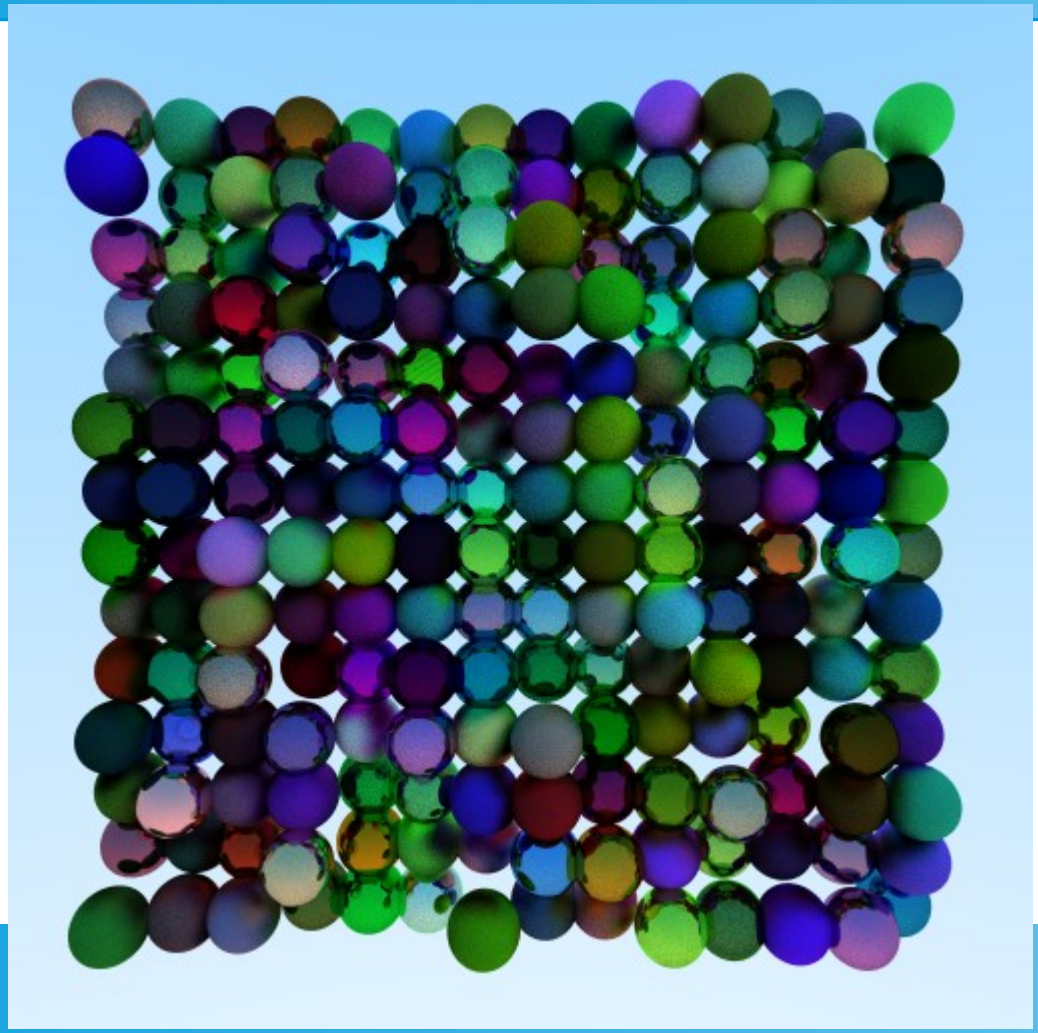


Tests were run five times each on an 8-core machine, rendering a 256x256 image with 16x supersampling. Each test had a varying number of random spheres.

Pics!



Pics!



Pics!