

Project Proposal: Raytracer

The primary goal of my project will be to program a 3D raytracer.

1. To begin, we import the `JuicyPixels` package to enable rasterization and rendering of basic images and scenes.
2. Then, I want to define some algebraic data-types to provide my scene geometry, like spheres, triangles (ie. 3-simplices), and some other basic geometric shapes.
3. I'll have to define something analogous to the ray class in the `C++` setting, but, since we're working in Haskell, I could probably just get away with defining it as a function $P(t) = A + Bt$ where this is an affine transformation. Here, A is the origin of the ray, B parameterizes the direction with respect to time, and t is, y'know, time.
4. I'll introduce a function for calculating the ray-sphere intersection.
5. Probably the trickiest part here is being able to get shading on our objects by computing surface normals and find a function for determining when which side gets the shadow by differentiating the outside and local normals.
6. Implement a function to manipulate the ray vector in a small bounded region where the ray hits a surface to simulate diffusion of light. To do so, we generate a random vector inside the unit sphere, normalize it, and then invert the normalized vector about an origin (ie. the point where the ray intersects the surface).
7. Set up something like a camera object to make the scene viewable from different angles (I'm not entirely sure how to do this in haskell).
8. We'll have the scene render and rasterize via the GPU, and so this we'll (really me here) parallelize this program via the `Accelerate` library for Haskell.

I'll also consider adding some bells and whistles if this project is too simple. Like, for instance, allowing different diffusion processes for different material types, or the simulation of different fluids as an aside once we have the ray-tracer working.