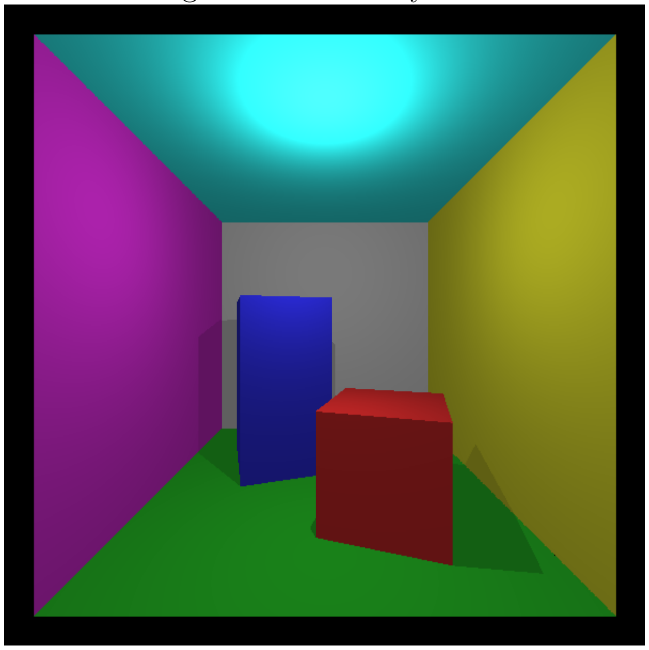
Raytracer

Iman Malik (im13557) and James Sewart (js13830)

April 29, 2016

All the given raytracer features were implemented and exist in the basicRaytracer folder. Camera movement was implemented simply by multiplying a speed constant by the elapsed number of time and incrementing the position of the camera in the desired direction. Camera rotation is implemented similarly by incrementing a pitch and yaw variable and generating a rotation matrix from these to apply to rays when casting.

Figure 1: Basic Raytracer



The ray intersection inverse operation was expanded to calculate distance first, reducing the amount of computation needed.

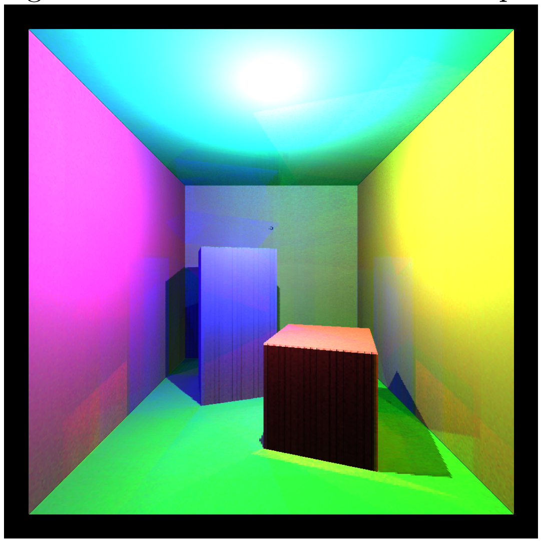
Two methods global illumination were attempted.

The first was to discretise the model into buckets representing 3d areas. A set of buckets was created to save the results from multiple bounces. Rays were cast from the light source and would increment the bucket that was the closest to the position that it intersected with the scene. A single reflected ray would then be calculated and an intersection would be calculated again, this next ray would increment the bucket corresponding to the second bounce.

By separating the bounces a nice reflective effect is achieved. To render to the screen, a ray tracer was run from the camera, intersecting with the world to get a position that is then used to lookup the light values in the closest bucket for each bounce. These values are summed to produce a final colour value which is drawn on the screen. This method potentially allows real-time camera control as the view rays are only looking up the value that has been generated.

Future work may be decoupling the bucket filling and querying to allow the renderer to both increase in quality over time as well as being able to view the progress in real-time. The downside of this approach is the need to discretise the space, this means that only so much quality is possible. This approach only calculated diffuse lighting, as the view angle is not known when filling. This implementation is located in folder globalIllumination1.

Figure 2: Global Illumination Attempt 1



The second approach was to only shoot rays from the camera to the light source, this permits a much higher resolution of image. First rays are send from the camera position into the scene to get a 3d point for which to calculate the lighting for. From this position, a hemisphere of rays are generated. Each ray is bounced around, accumulating colour and losing colour magnitude with each bounce. Once the ray intersects a light source, the accumulated colour is multiplied by its intensity to get a value for this ray. A diffuse and specular value is then calculated for the ray based on the surface normal and camera ray. These values are then added and averaged to get the final shading for the pixel.

With this method, the more rays generated for each point, the higher the quality of the render. As a trace is going from camera to light source, it is necessary to increase the size of the light source as a point source is never going to be hit. By adding another couple triangles in the scene with an intensity value, we can then check when we intersect with something whether we can finish bouncing. Because the light source now has a 2d structure, we get penumbras around obstacles. As each ray is reflected around the surface normal for each bounce, second and third bounce effects are as if the surface is quite reflective.

When a ray hits a light source, the resulting brightness is determined by the cosine of the angle between the ray and normal of the light to the 5th power to add a cool downward facing light effect. When shooting rays from the camera, a scene that isn't enclosed is used. However all subsequent rays that are casted are done so in an enclosed scene as this means few rays will disappear into a void. This also means the points closer to the camera become a bit more interesting as they are light up by rays bouncing off the wall that we are now looking through.

Figure 3: Global Illumination Attempt 2 with random rays for each pixel

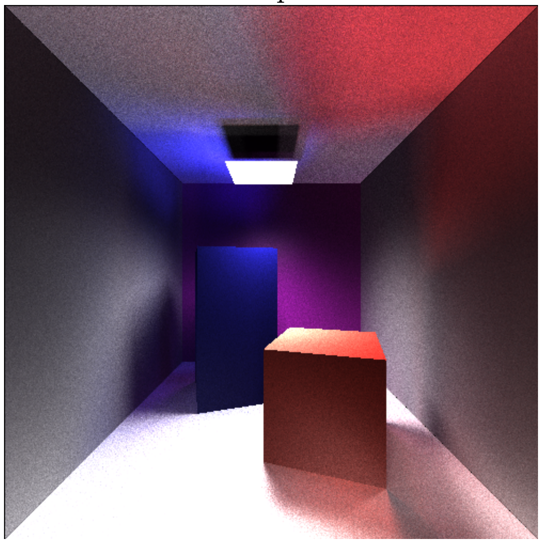


Figure 4: Global Illumination Attempt 2 with fixed rays for each pixel