Final Report: Global and Volumetric rendering in A2

Katie LaRue and Sean Brzoska
https://github.com/csci480-22f/fp-brzosks_laruek2_fp

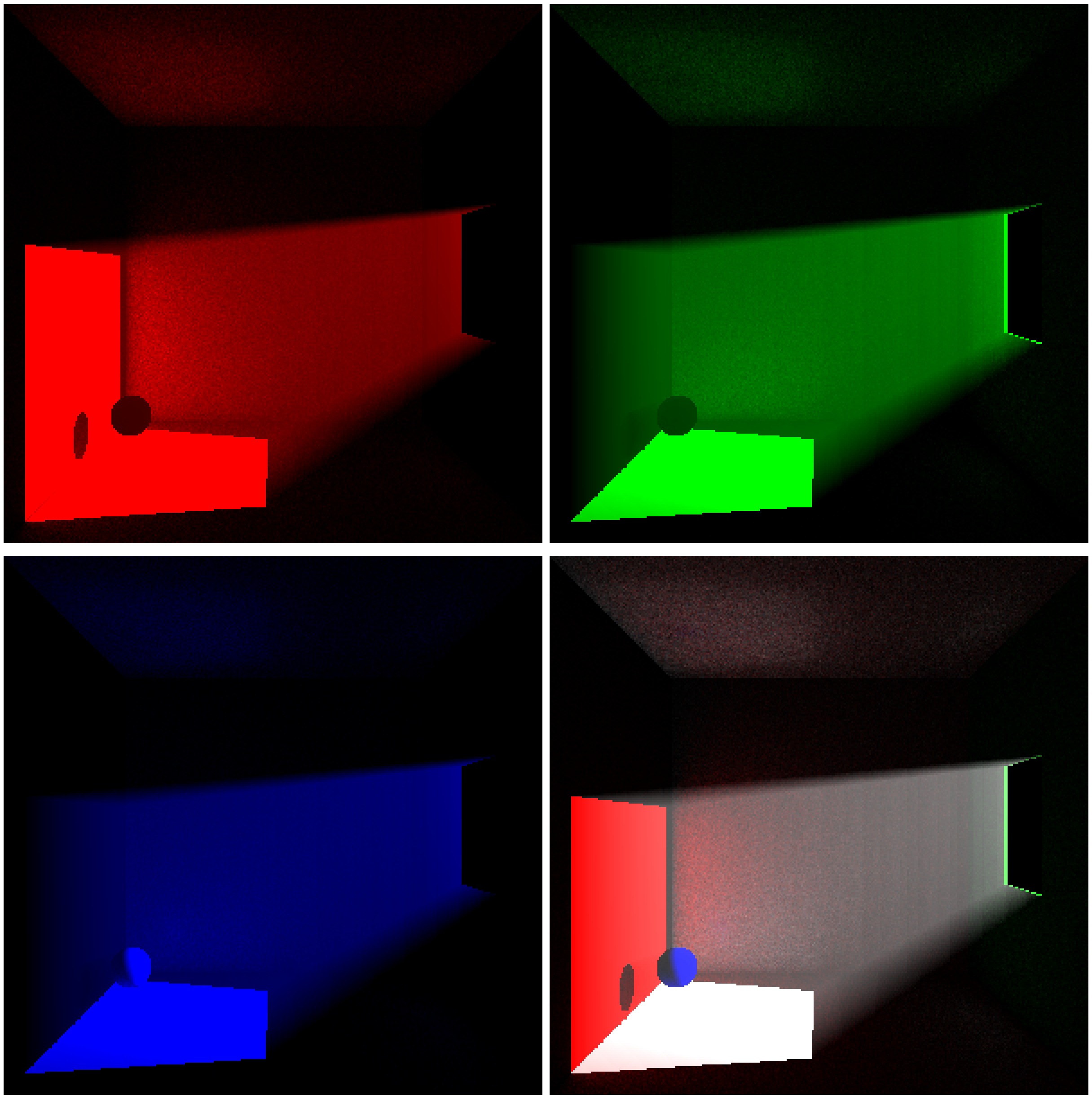
Our implementation of global illumination and volumetric lighting gives good results when looking at the images produced. We set out to implement basic methods for both global illumination and volumetric lighting, which we have accomplished. However, our plan to optimize these methods fell off once we optimized our code as much as we could and realized that optimized methods use mapping and other non trivial 'cheat' methods. So we added colored lighting, special surface support for global illumination, and localized volumetric lighting.

Global illumination gives color accurate results, and supports diffuse and specular surfaces (with the glossy coefficient added to Materials). We realized our preferred sampling method was intended for specular surfaces so global illumination was changed to support diffuse surfaces with random sampling and specular surfaces with glossy sampling. This was a slight adjustment on the initial plan but gives good results.

Volumetric lighting gives good results and supports both global fog and localized spherical fog. It seems most methods that calculate intensity falloff in volumetric lighting use mappings. We couldn't implement this and calculated the falloff based on the distance from the light. However, the light falloff was too strong in most cases so we had to scale it accordingly.

Results:

Colored Lights with global and volumetric:



From top left to bottom right:

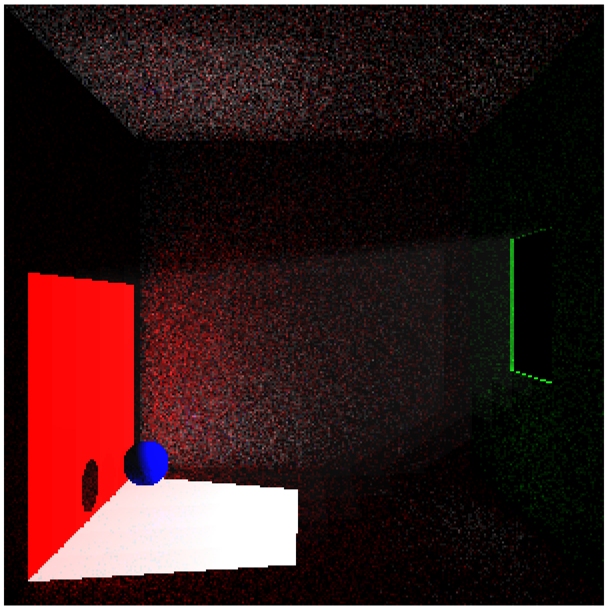
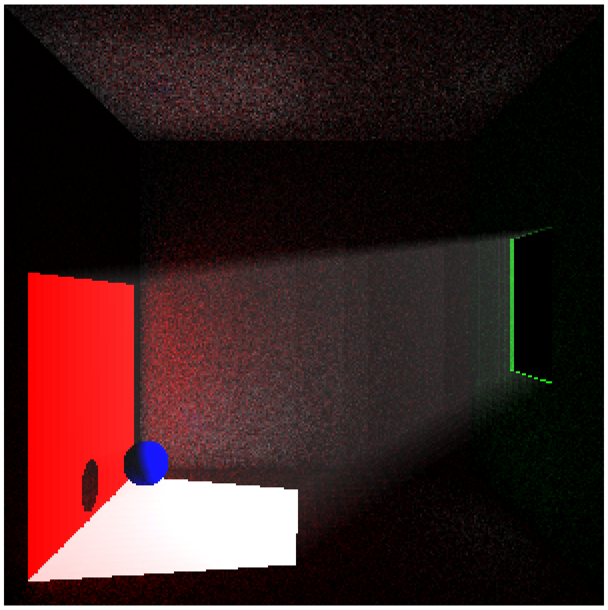
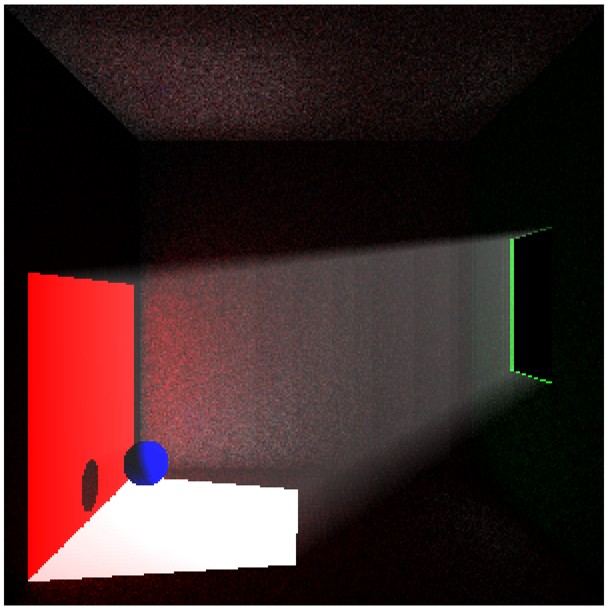
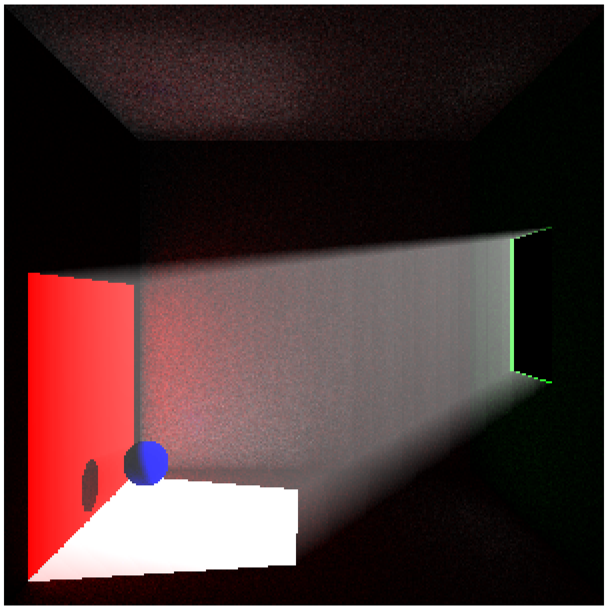
WWURay.main(13, 1, 300, 300, "r128-128-0.png", true, true, false, 128, true, 128, 0)

WWURay.main(14, 1, 300, 300, "g128-128-0.png", true, true, false, 128, true, 128, 0)

WWURay.main(15, 1, 300, 300, "b128-128-0.png", true, true, false, 128, true, 128, 0)

WWURay.main(12, 1, 300, 300, "w128-128-0.png", true, true, false, 128, true, 128, 0)

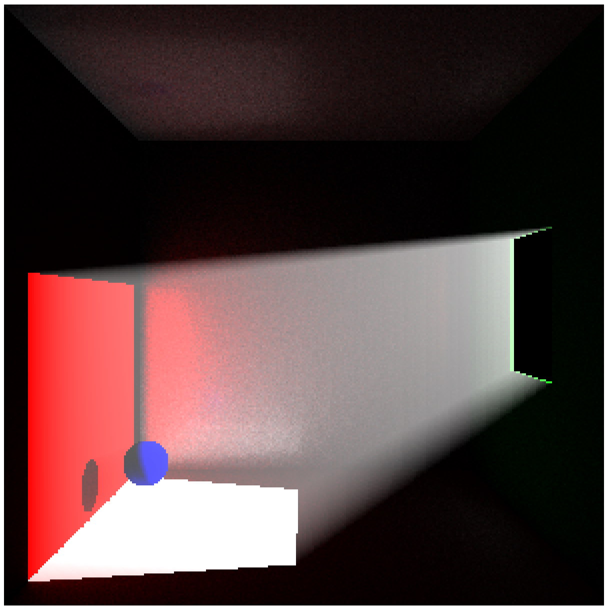
Low to high sample rates:

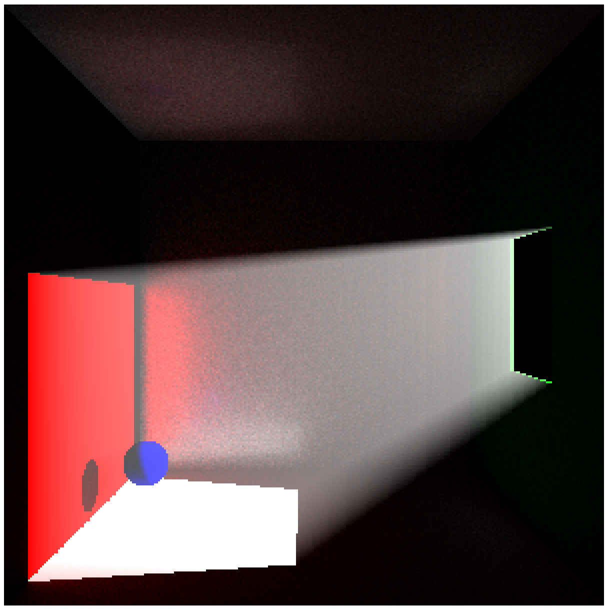





From top left to bottom right:

```
WWURay.main(12, 1, 300, 300, "w12-12-0.png", true, true, false, 12, true, 12, 0)

WWURay.main(12, 1, 300, 300, "w36-36-0.png", true, true, false, 36, true, 36, 0)

WWURay.main(12, 1, 300, 300, "w64-64-0.png", true, true, false, 64, true, 64, 0)

WWURay.main(12, 1, 300, 300, "w128-128-0.png", true, true, false, 128, true, 128, 0)
```

Specular lighting on back wall:



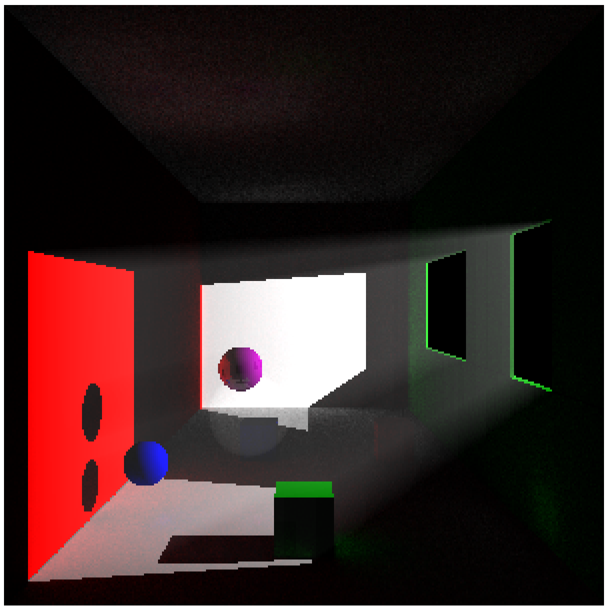


Left to Right:

WWURay.main(17, 1, 300, 300, "glw256-512-0.png", true, true, false, 256, true, 512, 0)

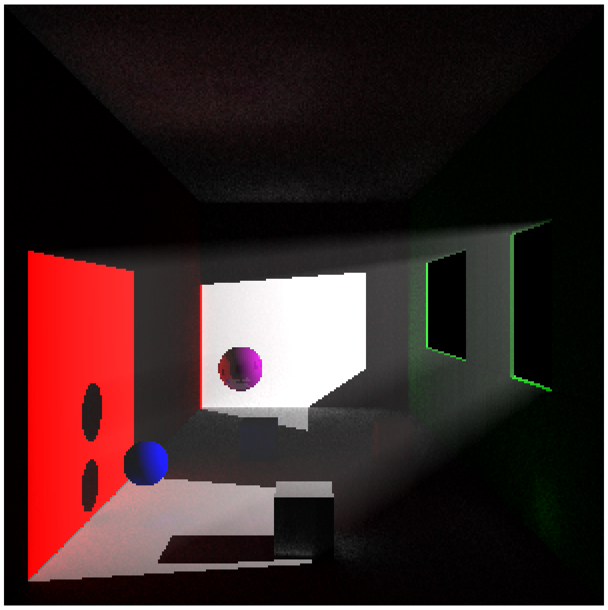
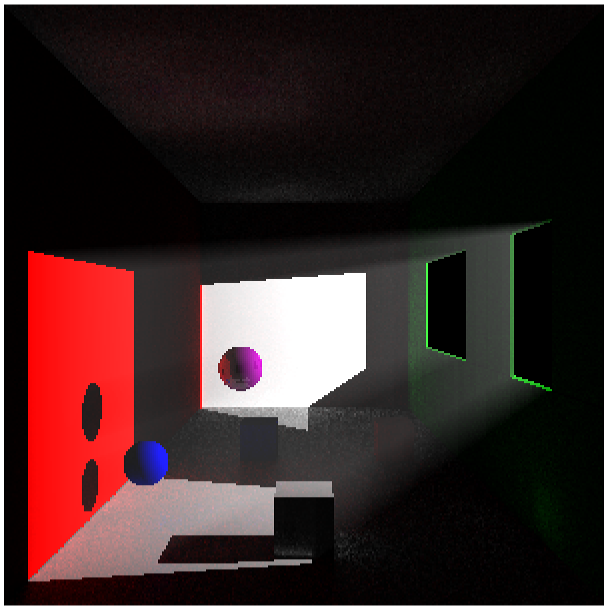
WWURay.main(19, 1, 300, 300, "glw256-512-0.png", true, true, false, 256, true, 512, 0)

More complex scene:



WWURay.main(16, 1, 300, 300, "results/scene16.png", true, true, true, 128, true, 128, 0)

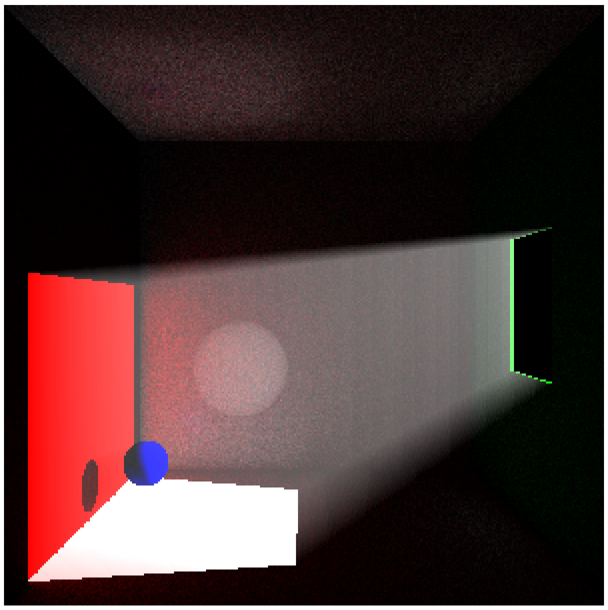
Left to Right:

WWURay.main(18, 1, 300, 300, "results/scene18_T1.png", true, true, false, 128, true, 128, 0)

WWURay.main(20, 1, 300, 300, "results/scene18_T2.png", true, true, false, 128, true, 128, 0)

Local Fog:



WWURay.main(12, 1, 300, 300, "results/lw128-128.png", true, true, true, 128, true, 128, 0)

Instructions:

Our code can be run with these parameters WWURay.main(scene_num, camera, height, width, outfile, volumetric, global_fog, local_fog, vCount, illum, gCount, gDepth)

New parameters:

1. volumetric, global_fog, local_fog, and illum are boolean values that say weather or not volumetric lighting, a scenic global fog, predetermined localized fog areas, and global illumination are on (T) or off (F)
2. vCount is the number of steps that we are considering (how many segments we break the ray into) when it comes to volumetric lighting
3. gCount - how many rays we generate for global illumination for each intersection
4. gDepth - how many bounces we go through (for each intersection that we generate rays for, how many of those intersections do we generate more rays for)

Specific commands for images are listed below the images they rendered.

Code Guide:

Our code is very similar to A2 in terms of layout and structure. Almost all changes we made were in the WWURay/traceray method and adding some extra helper methods.

The few changes outside of this method, include manipulating the pointlight and directional light struct's in Lights.jl and the associated shade_light methods in WWURay.jl to allow for colored lights, a new Cloud object in Scene.jl to represent our localized fog location and associated density, and making our Scene struct hold a air array to keep track of any localized fog areas since the scene needs to know where the intangible sections of the scene are. We also added a glossy_coeff to our Material struct in Materials.jl so that we can have more complicated objects.

The bulk of our new additions are in the traceray method and in the helper methods in WWURay.jl

The new helper methods are:

1. direct_light(lights::Light, scene::Scene, point::Vec3) - returns a hitrecord if point is not in the direct path of a given light
2. find_density(scene::Scene, pos::Vec3, global_fog::Bool) - finds the density of the room at a given 3D point based on sections of localized fog/mist/smoke
3. glossy_sample(scene::Scene, hitrec::HitRecord, view::Vec3, gloss::Float64, depth::Int, maxDepth::Int) - collects random directions in unit circle around intersection with more probability around mirror direction for glossy surfaces
4. diffuse_sample(scene::Scene, hitrec::HitRecord, view::Vec3, depth::Int, maxDepth::Int) - samples random directions based on unit circle around intersection for diffuse color
5. find_point_hemi(t, l) - given an angle t in rads, and length l, find the coordinates of the end of the vector on the unit hemisphere.

Traceray now allows the user to implement volumetric lighting by walking along a generated light ray and sampling the density of the air at that location. If we are in the direct path of the light, we calculate how much light should be visible at that point and add that to our local color.

We next consider global illumination. If that is desired, we check if we have a diffuse or glossy material and perform the appropriate sampling method. Diffuse sampling gets random directions around our intersection to collect colors, and then divides those color contributions by the amount of rays we just sent out. Glossy sampling does the same thing but with a weighted amount being closer in the direction of the mirrored ray. The rays that are closer to the mirrored ray get scaled down so they do not over contribute to the other randomly generated rays.

These additions were added to traceray because the math behind volumetric lighting and global illumination have to do with how rays are generated, how we collect ray information, locations and associated scene colors.