CS 488: Final Project Report

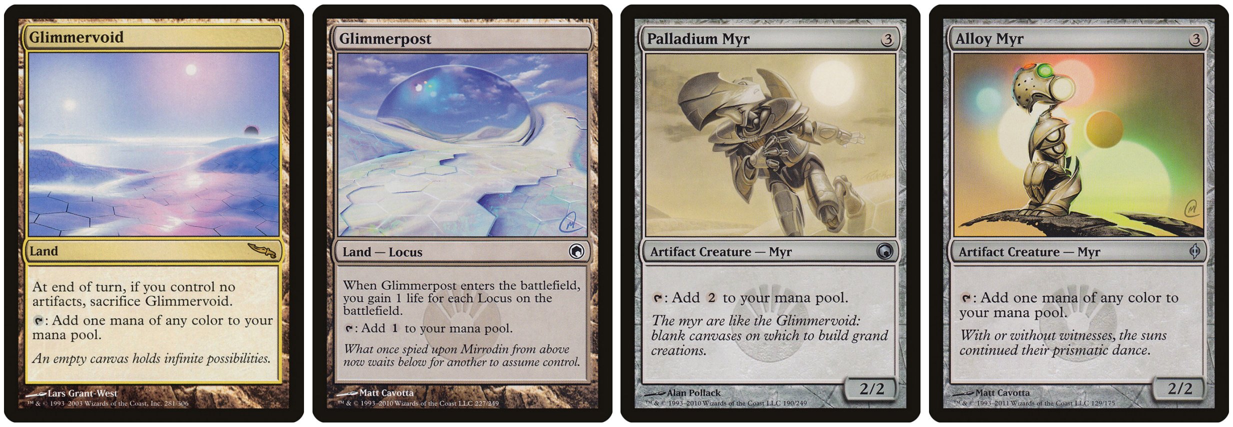
Benjamin Colussi 20469173 *Spring 2025*

1 Introduction

Welcome to Mirrodin, a planet made entirely of metal. Five moons orbit the planet, each emitting a different wavelength of light corresponding to the five colours of mana, red, green, white, blue, and black.

Glimmervoid is Mirrodin's equivalent of a desert. It is a landscape of rolling hills made of hexagonal metal plates stretching infinitely into the horizon.

This project is inspired by original artwork from the trading card game, Magic: The Gathering. The world of Mirrodin makes for an interesting simulation of light transport due to the combination of reflective metal surfaces, multiple coloured light sources, and atmospheric scattering effects. Most of the colour in the scene will ultimately correspond to the light scattering within the atmosphere from the coloured light sources, and not the reflective metallic surfaces.


Glimmervoid
Land
At end of turn, if you control no artifacts, sacrifice Glimmervoid.
: Add one mana of any color to your mana pool.
An empty canvas holds infinite possibilities.
Lars Grant-West
Glimmerpost
Land — Locus
When Glimmerpost enters the battlefield, you gain 1 life for each Locus on the battlefield.
: Add 1 to your mana pool.
What once spied upon Mirrodin from above now waits below for another to assume control.
Matt Cavotta
Palladium Myr
3
Artifact Creature — Myr
: Add 2 to your mana pool.
The myr are like the Glimmervoid: blank canvases on which to build grand creations.
2/2
Alan Pollack
Alloy Myr
3
Artifact Creature — Myr
: Add one mana of any color to your mana pool.
With or without witnesses, the suns continued their prismatic dance.
2/2
Matt Cavotta

Reference art depicting Mirrodin, with its coloured moons and hazy atmosphere.

2 Objectives

2.1 Refactor existing renderer

I built my final project on top of the CS 488 base code. The main sections of code in cs488.h are:

1. Global constants and variables
2. Image class (stripped down from base code)
3. Material class (where I implemented BRDF models)
4. Triangles (slightly stripped down and modified from base code)
5. BVH implementation (same as base code - includes my SAH-BVH implementation)
6. Sphere and Scene classes and shaders (where the path tracing algorithm is implemented)

The main.cpp file handles command line arguments, which specify the .obj file to load as a triangle mesh, and the number of samples to use in the Monte Carlo estimation. `Sphere` objects are added to the scene in the main function, such as light sources.

I took some inspiration from smallpt.cpp to implement parallelization with OpenMP, as well as the simple gamma correction and writing to a .ppm file. The compiler command "#pragma omp parallel for schedule(dynamic, 1) private(shade)" specifies to parallelize the following for loop on a dynamic schedule, because each pixel value can be computed independently of the others, but each will require an amount of work that is unknown at compile time. The "private" clause specifies that each thread has its own copy of the variable "shade", so that pixel values do not interfere with each other.

2.2 Path tracing

I have implemented a unidirectional path tracing algorithm that traces rays from the camera (backwards) into the scene in order to construct paths as samples for Monte Carlo estimation of the light transport equation.

I have implemented BRDF sampling for path continuation (more on this in the materials section). Sampling methods, along with BRDF evaluation and PDF computation are used in the estimator of the integral equation.

I have implemented Russian roulette in order to decrease runtime of my program without introducing bias. Russian roulette, in the very best case, will not increase variance, and in the worst case, increase variance, but the computational savings make up for this by allowing us to estimate using more samples, which reduces variance more than Russian roulette increases it.

I have implemented next event estimation in order to sample paths that contribute more radiance to the estimator, which in turn reduces variance and helps our estimator to converge faster.

I have implemented multiple importance sampling of the BRDF and NEE sampling strategies. The balance heuristic is used in order to choose optimal weighting of the PDFs for each of the two sampling strategies. MIS with the balance heuristic reduces variance by weighting samples highly if they have higher overlapping values in the PDFs of all the sampling strategies.

2.3 Light sources and spheres

I have implemented the `Sphere` class to create analytic sphere objects that can be placed within a scene. The `Sphere` class implements ray-sphere intersection detection. You can also specify a material for each sphere object, as with triangle meshes. By specifying the `LIGHT` material type, you can create spherical area lights which is required for my scene. You can also get coloured lighting by changing the RGB emission values of the material.

2.4 Materials

I first implemented specular reflection, which was relatively simple. By calculating the reflected direction and continuing the path in that direction, we are perfectly importance sampling the Dirac delta distribution and we simply continue the path without next event estimation.

I then implemented Fresnel reflection/refraction for glass using Schlick's approximation for performance. We importance sample the double Dirac delta distribution by choosing whether to reflect or refract based on the calculated Fresnel reflectance term which is always between 0 and 1. We can sample a uniformly distributed random variable and continue our path by calculating the reflected or refracted direction.

Lastly, I implemented a BRDF model for rough metallic surfaces based on the Cook-Torrance microfacet model. Instead of modelling a rough material manually, Cook-Torrance models the tiny microfacets of the rough surfaces stochastically. It models the surfaces as having tiny, flat, pyramid-shaped bumps that reflect specularly and are stochastically distributed according to a normal distribution function. In order to ensure this normal distribution function is actually a valid PDF, we include a geometric term, or masking/shadowing function, which further accounts for the possible orientations of the microfacets of the microsurface. Such situations might be that a microfacet is masked by another, so our ray would have intersected with another microfacet, or that the sampled microfacet is facing away from the incoming ray.

We also include a Fresnel term to account for the possibility of being reflected or refracted. In my program, I implemented the model to combine lambertian and specular reflections, to get a realistic effect for metallic surfaces. I compute the Fresnel term to properly weight the possibility of being specularly reflected about the sampled micronormal.

The algorithm begins by sampling a microsurface normal from the unit hemisphere in such a way so that the probability matches our normal distribution function, in my implementation I use the GGX NDF. We then centre this about the macrosurface normal, and reflect the incoming ray about this properly-oriented microsurface normal. We now have all of the inputs required to evaluate the BRDF and PDF, incoming direction, macrosurface normal, microsurface normal and outgoing direction. Evaluating the BRDF requires evaluating the Fresnel term, the NDF, and the geometry term. In my implementation I use the Smith approximation to the geometry term, which models the geometry of the incoming direction and outgoing direction independently, so it factors into two separate terms, one for each direction. These terms must be derived specifically for the NDF that we use.

The BRDF is given by the following, where $\mathbf{v}$ is the outgoing light direction, $\mathbf{l}$ is the incident light direction, $\mathbf{n}$ is the macrosurface normal, and $\mathbf{h}$ is the sampled microfacet normal. The first term is the diffuse part, which is the Lambertian model BRDF, the base colour divided by π to normalize. The second term is the specular part:

$$\mathrm{f}_r(\mathbf{v},\mathbf{l}) = \frac{\rho_d}{\pi} + \frac{\mathrm{F}(\mathbf{v},\mathbf{h})\mathrm{D}(\mathbf{h})\mathrm{G}(\mathbf{l},\mathbf{v})}{4\langle\mathbf{n}\cdot\mathbf{l}\rangle\langle\mathbf{n}\cdot\mathbf{v}\rangle}.$$

For the Fresnel factor, we use Schlick's approximation. For the normal distribution function, we use the GGX model. For the geometry function, we use Smith's approximation with Schlick-GGX factors. r_p is the roughness parameter, which makes surfaces appear rougher as it increases:

$$\mathrm{F}_{\text{Schlick}}(\mathbf{v},\mathbf{h}) = \mathrm{F}_0 + (1.0 - \mathrm{F}_0)(1.0 - \langle\mathbf{v}\cdot\mathbf{h}\rangle)^5$$

$$\mathrm{D}_{\text{GGX}}(\mathbf{h}) = \frac{\alpha^2}{\pi\left(\langle\mathbf{n}\cdot\mathbf{h}\rangle^2(\alpha^2-1)+1\right)^2} \quad , \quad \alpha = r_p^2 \quad , \quad r_p \in [0,1]$$

$$\mathrm{G}_{\text{Smith}}(\mathbf{l},\mathbf{v}) = \mathrm{G}_1(\mathbf{l})\mathrm{G}_1(\mathbf{v}) \quad , \quad \mathrm{G}_{1\ \text{Schlick-GGX}}(\mathbf{v}) = \frac{\langle\mathbf{n}\cdot\mathbf{v}\rangle}{\langle\mathbf{n}\cdot\mathbf{v}\rangle(1-k)+k} \quad , \quad k = \frac{\alpha}{2}$$

2.5 Volumetric scattering

I initially intended to implement a ray marching algorithm to simulate single scattering, but I instead chose to implement free-flight sampling on top of my unidirectional path tracing algorithm. This method is much more robust and much less biased, because it estimates the actual volume rendering equation by sampling distance as well as solid angle. This allows us to estimate multiple scattering and not just single scattering, by allowing light to travel in all directions and eventually end up along a ray towards the camera.

We first sample a direction, and then sample free-flight distance along this direction. We then check for a ray-surface interaction, and check if the sampled distance is less than the distance of the surface interaction. If the sampled distance is less, we have a volume interaction. If the sampled distance is greater, we have a surface interaction. In both cases we perform next event estimation, and then sample solid angle to continue the path. In the case of a volume interaction, we sample the phase function and compute the accumulated radiance based on the volume rendering equation. In the case of a surface interaction, we accumulate radiance in the usual unidirectional path tracing manner.

We also implement multiple importance sampling of the sampling methods used in volume scattering. The only thing that changes from the usual path tracing MIS is that we are now sampling distance at each step to organically continue the path. This means that our BRDF/phase function sampling method must now be multiplied by the probability of sampling the distance at each step. We weight the estimate of the accumulated radiance using the balance heuristic as before, but now our two sampling strategies are BRDF/phase function multiplied by distance sampling, and sampling solid angle towards a light source during NEE.

This method allows us to Monte Carlo estimate the double integral of the volume rendering equation, which involves an integral over distance of an integral over solid angle. When we have a volume interaction, we multiply by the probability of reaching the sampled distance. When we have a surface interaction, we multiply by the probability of reaching or exceeding the distance to the surface, because that is the probability of sampling a distance at or beyond the surface, if the surface wasn't there.

The volume rendering equation and its Monte Carlo estimator are given by the following:

$$L(x,\omega_i) = \int_0^t T(x, x+s\omega_i)\,\sigma_s(x+s\omega_i)\,L_s(x+s\omega_i,\omega_i)\,ds$$

$$L_s(x+s\omega_i,\omega_i) = \int_S p(\omega_i,\omega)\,L(x+s\omega_i,\omega)\,d\omega$$

$$L(x,\omega_i) \approx \frac{T(x,x+s\omega_i)\,\sigma_s(x+s\omega_i)\,p(\omega_i,\omega)\,L(x+s\omega_i,\omega)}{pdf(s)pdf(\omega)}$$

2.6 Atmospheric scattering

I initially intended to try implementing various atmospheric scattering models but I decided this was not fully necessary for the scene I was trying to create, considering it is set in a fictional world with coloured light sources, so the atmospheric effects produced by Rayleigh or Mie scattering wouldn't be very meaningful.

In the end, I implemented the Beer-Lambert law to simulate the attenuation of light through a homogeneous atmosphere, in order to achieve a hazy atmospheric effect, and create a glow around the coloured light sources in my scene.

The Beer-Lambert law states that the radiance of light decays exponentially as it travels through the medium according to the absorption and scattering coefficients of the medium. The absorption coefficient, σ_a, is the probability density that the light is absorbed per unit distance travelled in the medium. Similarly, the scattering coefficient, $sigma_s$, is the probability density that there is a scattering event per unit distance travelled in the medium. The attenuation coefficient is the sum of these, $\sigma_t = \sigma_a + \sigma_s$.

Under the Beer-Lambert law, the integral defining the transmittance function can be solved analytically to get the exponential function:

$$T(t) = \exp\left(-\int_0^t \sigma_t ds\right) = \exp\left(-t\sigma_t\right)$$

We must sample from the exponential distribution in order to importance sample the transmittance term that arises from the Beer-Lambert law. I have implemented this sampling by using the inversion method of the CDF to transform our uniform random numbers into exponential distribution samples.

I am using the simplest phase function $\frac{1}{4\pi}$, so I sample a direction on the unit sphere, which can also be accomplished by sampling spherical coordinates using the inversion method of the CDF. This allows us to importance sample the phase function exactly.

There is a lot of cancellation in the volume rendering equation when using the Beer-Lambert law for volume scattering in homogeneous media. When volume scattering, the probability of sampling a free-flight distance cancels with the transmittance term that arises from the Beer-Lambert law, leaving only $\frac{1}{\sigma_t}$. We also importance sample the phase function exactly by sampling the unit sphere, so this cancels perfectly. All that is left is the scattering coefficient from the in-scattering term. So we end up only multiplying by the scattering coefficient over the attenuation coefficient, $\frac{\sigma_s}{\sigma_t}$.

In the case of surface scattering, the probability of reaching the surface or exceeding it is given by the tail of the exponential distribution, which is 1 minus the CDF at the distance of the surface interaction. This is equivalent to the transmittance of the previous vertex to the surface, so it cancels perfectly.

2.7 Model the scene

I used Blender to create a simple hexagonal shape with bevelled edges to create a rocky effect. I placed mutated copies of this object in a circular pattern, leaving room to place a `Sphere` object, which is defined in `main.cpp`.

References