Project Report

Title: 4D Space Raymarched Pathtracer

Name: Arnav Kumar

1 Project Overview

The program generates a render of a 4D scene consisting of multiple frames, each representing a different camera position rendering the same scene. We split the scene up into 3D hyperplane slices and have a camera image each slice using pathtracing.

The renderer is a raymarching pathtracer which means that intersections are calculated with raymarching, and we use Monte Carlo integration to approximate the shading equation at our hit spots. Scenes are specified using code with a 4D object manipulation system allowing custom specification of infinite tilings, objects with multiple materials, and rounding and onioning arbitrary geometries. There is also support for the GGX microfacet model, ideal diffuse surfaces, specular mirrors, and glass (with Fresnel refraction). Lastly, there is support for volumetric scattering in mediums like glass and air.

Provided are 3 example scenes. There is interesting insight into 4 dimensional geometry, and we can visualize 4D oddities like hyperboxes and the densest packing of equal hyperspheres.

2 Technical Outline

2.1 Camera and 4D coordinate system setup with rendering to GIF

To begin, let the camera starting and ending positions be represented by $p_i, p_f \in \mathbb{R}^4$, and let the 4 orthonormal unit vectors $\hat{x}, \hat{y}, \hat{z}, \hat{w} \in \mathbb{R}^4$ form a basis for the camera space.

When rendering frame f of F, we consider a 3D affine subspace of the 4D space given by the basis vectors $\hat{x}, \hat{y}, \hat{z}$ which contains the point $p = \frac{f}{F}p_i + \frac{F-f}{F}p_f$. We then generate rays from our camera to colour pixels in our frame as if we were in a 3D space.

So for pixel (i, j) we can consider a virtual screen in the $\hat{z}$ direction from p. The right side of the screen is in the $\hat{x}$ direction, and the top of the screen is in the $\hat{y}$ direction. We can calculate the location of the virtual pixel q, as a linear combination of $\hat{x}, \hat{y}, \hat{z}$ added to p, which means that the virtual pixel lies in the aforementioned 3D affine subspace. In particular, $q \in p + \operatorname{span}\{\hat{x}, \hat{y}, \hat{z}\}$.

Then, our ray to trace for the pixel (i, j) in frame f starts at p and has direction $q - p$.

2.2 Hypersphere geometry and raymarching

A hypersphere geometry with radius r centered at the origin can be specified with the SDF given by $f(x) = \|x\|_2 - r$. A halfspace geometry with normal n can be specified with $f(x) = x \cdot n$.

Now, for the raymarching, we can get the step size based on the minimum value of the SDFs of all the objects in the scene with respect to the current position. We check to see if we are within a "marching epsilon" of an object by checking if our step size is sufficiently small, and if so, we handle the hit. If not, we march the ray forward by the calculated step size and repeat.

How do we handle the hit? We have to have information about the hit position, material information, and the surface normal, which we can calculate as the normalized version of $\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial w}\right)$ [Won16]. We can estimate

$$\frac{\partial f}{\partial x}(p) \approx \frac{f(p + \varepsilon\hat{i}) - f(p - \varepsilon\hat{i})}{2\varepsilon}$$

where $\hat{i}$ is the standard basis vector in the x direction, and similarly for the other components.

When glass is implemented, there is the additional condition that we have to check if we are inside a medium and use the absolute value of the SDFs as well. We have to consider whether the interface is glass-to-air or glass-to-opaque.

2.3 Material specification system

There is an intention to represent infinite geometries in one object. To this end, an object should be able to take multiple materials based on position.

Every object has a material map: a map that given a position and a surface normal, returns a material. Materials themselves can be one of a light, glass, air, mirror, shiny (GGX microfacet), or diffuse (lambertian with optional emission).

Materials are used in hit shading. Given a material, refractive indices of mediums, hit position, surface normal at intersection, and the direction of the ray, we can produce information about emission (from this vertex of the path), tint (for the rest of the vertices in the path), and a new ray giving the new direction to follow.

2.4 Microfacet BRDFs, Lambertian BRDF, and Fresnel refraction

Implementing Fresnel refraction can be done by sampling the outgoing direction of the reflected ray between either reflection or refraction with probability of reflection approximated as $c + (1-c)(1-(l \cdot n))^5$ where $c = \left(\frac{\eta_1 - \eta_2}{\eta_1 + \eta_2}\right)^2$, and η_1, η_2 are the refractive indices inside and outside the current medium [Hof12].

Implementing the Lambertian BRDF in 4 dimensions is not so straightforward. If our surface has albedo ρ then the BRDF is of the form $\frac{\rho}{k}$, but what is the value of k? We want a k such that when $\rho = 1$, we exactly conserve the amount of energy in the system.

This means computing a 4D integral to determine the energy of the system, but luckily we can get around this. Consider sampling a direction according to a cosine weighted distribution, and then multiplying the contribution of all of the subsequent terms by the albedo. This is equivalent and much easier to complete in 4D.

First, sample from a ball, then project that sample up onto a hypersphere to get a cosine weighted distribution. How do we then transform the sampled distribution to the surface normal? We multiply the sampled vector by the Householder reflection matrix for reflecting the surface normal to the center of the distribution. Knowledge of the householder reflection matrices is something taught in CS475. Specifically, if we want to find the matrix F such that $Fx = \|x\|_2 \hat{i}$ then we simply let $v = \text{sign}(x_1) * \|x\|_2 \hat{i} + x$ and then take (where $\otimes$ is the outer product)

$$F = I - 2\frac{v \otimes v}{v \cdot v}$$

The next consideration is for microfacet BRDFs. There are multiple implementations but they all follow the following formulation consisting of a Fresnel reflection component, a distribution function, and a shadowing function / geometry term [Cao15; PJH18]

$$f(v,l) = \frac{F(v,h)D(h)G(l,v)}{4(n \cdot l)(n \cdot v)}$$

where v is the direction the ray came from, l, the direction it will go in, h a vector between l and v, and n the surface normal.

The most popular microfacet model seems to be GGX, which is what I choose to employ.

As with the implementation of the Lambertian BRDF, I will rely entirely on sampling. This is to avoid the calculation of the BRDF so we can easily ensure that there isn't an addition of energy into the scene. To this end, I implement the sampling of a mircofacet normal h according to the GGX distribution around the surface normal n. We keep resampling according to a probability given by the geometric term of the generated microfacet normal. The probability that we resample is given by the formula for the geometry term using the Schlick-GGX approximation. Then, reflection is conducted with a tint given by the Fresnel term.

We find the sampling strategies for $D(h)$ via GGX [Cao15] to satisfy (for $\epsilon \sim \text{Uniform}(0,1)$)

$$\theta = \arctan\left(\alpha\sqrt{\frac{\epsilon}{1-\epsilon}}\right)$$

Thus, we sample from a sphere, and then sample θ as above, combining the two to get the new $\mathbb{R}^4$ vector representing direction.

Furthermore, we can find the geometric term for the GGX model with the Schlick-GGX approximation that gives us $G(l, v) = G_1(l)G_1(v)$ where

$$G_1(x) = \frac{n \cdot x}{(n \cdot x)(1-k)+k}$$

Here, $k = \frac{\alpha}{2}$, and α is the square of the roughness of the material.

The last part is the Fresnel term which we know to be $F(v, h) = c + (1-c)(1-(l \cdot h))^5$ from the Schlick approximation from earlier [Hof12].

2.5 Volumetric scattering

Given a ray, I assume it is in homogeneous medium. Then, I can sample a distance d from an exponential distribution by letting $\epsilon \sim \text{Uniform}(0, 1)$ and then taking $d = \frac{-\log \epsilon}{\sigma_t}$ where σ_t is the extinction coefficient. As we march through the medium, we keep track of how far we've come, and if we march further than d without an intersection, we scatter the ray according to a phase function (I have chosen to scatter in a random direction with equal probability) and multiply the result by $\exp(-d\sigma_t)$. If we intersect with an object first, then we simply proceed as normally, but after the hit, we sample a new d based on the medium.

2.6 Pathtracing with parallelization

This section is simply about sending multiple rays for each pixel, and then averaging them. Parallelization done with openMP. I completed this step and had Monte Carlo integration before working on diffuse or microfacet BRDFs.

When we receive information about what to do about a hit, we get information about the emitted light from the surface, the tint for the following vertices, and the direction of the next ray.

In specific, we are solving

$$L_o(x, \omega_o) = \int_\Omega f(x, \omega_o, \omega_i) L_i(x, \omega_i) \cos(\theta_i) d\omega_i$$

using Monte Carlo integration. When we sample from a distribution, we will still get a coefficient related to the BRDF, which is what translated to the "tint".

So essentially, along with each ray, we keep track of two values: accumulated colour (based on running sum of emissions times the tint of the ray when accumulating the considered vertex of the path) and accumulated tint (product of all of the tints at all the vertices of the path so far).

2.7 SDF infinite tiling, rounding, onioning, and transformations

For a given SDF f, the onioned version of f with thickness d is given by $f'(p) = \|f(p)\|_2 - \frac{d}{2}$ [Qui]. The rounded version with radius r is given by $f'(p) = f(p) - r$ [Qui]. A translation by t is given by $f'(p) = f(p-t)$. A rotation whose inverse rotation matrix is R is given by $f'(p) = f(Rp)$.

For infinite tiling, we can think of a tiling as splitting space into cells. Then, a tiling can be defined by an object to tile, and a function which returns the center of the cell containing a given position. If such a function is g then the tiling SDF is given by $f'(p) = f(p - g(p))$.

Reference the code or the devlog for details on what the function g looks like for the D4 tiling.

2.8 Denoising

The denoising algorithm is based off of Median filtering. Simply put, if we render everything into a 3D gird of pixels, we can iterate over each pixel and get the RGB values of all of its neighbours (neighbouring frames and pixels in the same frame). Then, form a new set of frames where each pixel's value has RGB components which are the medians of the channel values we found for its neighbours [Wik25].

4D Geometry

[Sny] Bailey Snyder. *Interactive 4D handbook.* URL: https://baileysnyder.com/interactive-4d/.

Raymarching and SDFs

[dem20] demofox. *Ray Marching Fog with blue noise.* May 2020. URL: https://blog.demofox.org/2020/05/10/ray-marching-fog-with-blue-noise/.

[Qui] Inigo Quilez. *Distance Functions.* URL: https://iquilezles.org/articles/distfunctions/.

[Won16] Jamie Wong. *Ray Marching and Signed Distance Functions.* July 2016. URL: https://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/.

Pathtracing

[CJ16] Per H. Christensen and Wojciech Jarosz. "The Path to Path-Traced Movies". In: *Found. Trends. Comput. Graph. Vis.* 10.2 (Oct. 2016), pp. 107–113. ISSN: 1572-2740. DOI: 10.1561/0600000073. URL: https://doi.org/10.1561/0600000073.

BRDFs

[Cao15] Jiayin Cao. *Sampling microfacet BRDF.* Nov. 2015. URL: https://agraphicsguynotes.com/posts/sample_microfacet_brdf/.

[CT82] R. L. Cook and K. E. Torrance. "A Reflectance Model for Computer Graphics". In: *ACM Trans. Graph.* 1.1 (Jan. 1982), pp. 11–13. ISSN: 0730-0301. DOI: 10.1145/357290.357293. URL: https://doi.org/10.1145/357290.357293.

[GSN19] GSNComposer. *Microfacet BRDF: Theory and Implementation of Basic PBR Materials.* July 2019. URL: https://www.youtube.com/watch?v=gya7x9H3mV0.

[Hof12] Naty Hoffman. "Background: Physics and Math of Shading". In: *SIGGRAPH 2012 Course: Practical Physically Based Shading in Film and Game Production.* Self Shadow, 2012, pp. 9–18. URL: https://blog.selfshadow.com/publications/s2012-shading-course/hoffman/s2012_pbs_physics_math_notes.pdf.

[HVS13] Christophe Hery, Ryusuke Villemin, and Pixar Animation Studios. "Physically based lighting at pixar". In: *Physically Based Shading SIGGRAPH Course* (2013), pp. 10–12. URL: https://graphics.pixar.com/library/PhysicallyBasedLighting/paper.pdf.

[PJH18] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation.* The MIT Press, 2018. Chap. 8.4 Microfacet Models. URL: https://www.pbr-book.org/3ed-2018/Reflection_Models/Microfacet_Models.

[Wal+07] Bruce Walter et al. "Microfacet Models for Refraction through Rough Surfaces." In: *Rendering techniques* 2007 (2007), 18th.

Miscellaneous

[Sim15] Cory Simon. *Generating uniformly distributed numbers on a sphere.* Feb. 2015. URL: https://corysimon.github.io/articles/uniformdistn-on-sphere/.

[Wik25] Wikipedia. *Median filter.* July 2025. URL: https://en.wikipedia.org/wiki/Median_filter.