Abstract

This project involves leveraging photon mapping techniques for non-physically based effects. I implemented two separate photon mapping algorithms as standalone volume integrators for Pharr and Humphrey’s physically based ray-tracer. The first performed Jarosz’s beam radiance estimate, and the second approximated radiance through Jarosz’s Beam Query x Beam Data, 1D Blur estimate. By modifying the default, physically accurate input parameters of the latter, I create non-photorealistic, artistic effects in line with the strategies first presented by Nowrouzezahrai in his research for the cgi Disney film Tangled.

1 Photon Mapping Overview

Photon mapping is a particle-tracing algorithm for solving the light transport equation. Photon mapping approximates incoming light as discretized packets of energy, or ”photons,” which are stored throughout the scene at specified interaction points. The initial photons originate from light sources, and are ”bounced” throughout the scene given the reflective properties of each surface, and the photon’s associated ”power” is diminished at each interaction step.

\[ L_r(x, \omega) = \int_{\Omega} f_r(x, \omega', \omega) L_i(x, \omega) \cos \theta_i d\omega \]

Every photon stores, minimally, an associated weight, incoming direction, and point in space. The PBRT integrator interpolates among these photons to approximate the rendering equation (eq. 1). Given point \( x \) and direction \( \omega \), the rendering equation integrates over the hemisphere \( \Omega \) above \( x \), accounting for incoming luminance in all directions multiplied by the bidirection reflectance distribution function. The rendering equation ultimately returns the outgoing radiance at \( x \) in direction \( \omega \).

Photon mapping can be easily extended to handle volume regions as well. Although PBRT is shipped with a photon integrator, it does not have a photon based volume integrator. I implement two such algorithms, the Beam Radiance Estimate, and the Beam Query x Beam Data Estimate, with 1D Blur, both of which use photon-mapping techniques to render volumes.

2 The Beam Radiance Estimate

This is the algorithm first presented in Jarosz’s paper The Beam Radiance Estimate for Volumetric Photon Mapping.

2.1 Radiative Transfer Equation

The radiance equation assumes constant radiance along rays. However, this is rarely the case, since even air scatters light. Participating media are particles which affect the behavior of light.
through regions of space. In lieu of rendering equation, light transport in participating media is instead modeled by the radiative transfer equation [2].

\[
L(x, \omega) = T_r(x \leftrightarrow x_s) L(x_s, \omega) + \int_0^L T_r(x \leftrightarrow x_t) \sigma_s(x_t) L_i(x_t, \omega) \, dt \tag{2}
\]

Transmittance \( T_r \) is a value between 0 and 1 giving the fraction of radiance transmitted between points \( x \) and \( x_s \). \( L(x, \omega) \), Equation 1, gives the outgoing radiance at the closest surface. The integral \( \int_0^L \) sums the accumulated in-scattered radiance along the entire length \( S \) of the ray. Each radiance measure (given by \( L_i \)) at current point \( x_t \) is then attenuated by the aforementioned transmittance \( T_r \), and the scattering coefficient of the participating media, \( \sigma_s \).

Traditional volume photon mapping estimates \( L_i(x_t, \omega) \) through ray-marching. A random step-size is generated, and the integrator marches down the ray in these increments. At each point, a spherical sample is taken and all encompassed photons are used to compute the in-scattered radiance. The continuous integral became [2],

\[
\sum_{i=0}^{S-1} T_r(x \leftrightarrow x_t) \omega_s(x_t) L_i(x_t, \omega) \Delta t \tag{3}
\]

### 2.2 Beam Radiance Estimate

The Beam Radiance Estimate reformulates Equation 2, so in lieu of ray-marching down the line of sight and performing separate calculations at each step, the entire sum of in-scattered radiance is calculated directly from the sum total of photons within a certain distance from the ray. The reformulated algorithm calls assigns a radius to the ray being followed, and considering all photons whose radii intersect the ray-cylinder. Equation 2 becomes [2]:

\[
\frac{1}{N} \sum_{i=1}^{N} K_i(x, \omega, s, x_i, r_i) T_r(x \leftrightarrow x'_i) \sigma_s(x'_i) p(x_i, \omega, \omega'_i) \alpha_i \tag{4}
\]

The summation now eliminates the computationally intensive radiance estimate \( L_i \) at various points throughout the ray. Instead, the in-scattered radiance is directly computed as an average over all photons \( p(x_i, \omega, \omega'_i) \), whose radii intersect the ray-cylinder.

### 2.3 Implementation

I began with starter code which contained a PBRT implementation of photon volume mapping (with ray marching), coded by Chia-Kai Liang and Chihyuan Chung of the National Taiwan University.

1. Reformulate the existing PBRT KD-Tree implementation to handle the photon data structures.
2. Implement an “n-closest” search in the Kd-Tree, a lookup function which returns a radius for the input photon based on its proximity to the \( n \) closest photons. Thus, the photons have a scalable adaptive radius, with control parameter \( n \) controlling the blur.
3. Reformulate the PBRT bounding box implementation to handle the photon data structures. Add a bounding box constructor taking a volume photon KD-Tree as input.
4. During rendering time, the ray is intersected with the volume photon BBH, which returns a vector of all photons whose bounding boxes intersect with the ray.
5. Calculate Equation 4 and return the value as the accumulated radiance due to participating media (the surface integrator handles direct lighting).
2.4 Results

I was able to get the integrator up and running with a pre-existing PBRT spotlight and fog scene. However, there were certainly many limitations in rendering that speak to large room for future improvements. The circular sampling space was quite evident, as Figure 1 amply evinces. Fog is clumped in circles throughout the space. Furthermore, all scattering and step-size parameters (for photon shooting) have to be calculated by hand. In Figure 1, I set the step distance to one that does not quite reach the surface too often, as it has a high probability of extinction before reaching the ground.

Figure 1: Fog rendered with Beam Radiance Estimate (my implementation)

However, the result was mostly in line with the example images provided in the paper. The following two images were both taken from the original paper. Having implemented adaptive radii, I managed to avoid the jarring artifacts of Figure 2. However, my integrator leaves much to be desired in the way of blurring the conic sampling areas, and in that respect the integrator falls short of Figure 3’s photorealistic renderer.

Figure 2: Fixed Radius Beam Radiance Estimate (Disney Research)
The source code is on my github. The volume photon map integrator is located in volumephotonmap.cpp and volumephotonmap.h.

3 Beam Query x Beam Data, 1D Blur

This section draws from Jarosz’s paper on Photon Beams, A Comprehensive Theory of Volumetric Radiance Estimation using Photon Points and Beams, and Nowrouzezahrai’s work on Artistic Volumetric Lighting in A Programmable System for Artistic Volumetric Lighting

3.1 Photon Beams

Photon beams are analogous to ray marching from the lights instead of the eye. This makes possible a different aesthetic from the point-based approach, in which the effect of a beam of light is only noticed at the random points at which a photon-point is deposited. Photon beams are essentially cones which, initially propagated from the light, then extend the whole stretch of the medium. At randomly sample points throughout this beam, further beams are constructed to simulate multiple scattering. In the Photon Beam approach, Equation 3 is manipulated into [1]:

\[
\frac{\sigma_s}{\mu_R(r)} \sum_{b \in \bar{R}_b} \frac{f(\theta_b)\Phi_b e^{-\sigma_t^b} e^{-\sigma_t^c} e^{-\sigma_t^c \sin \theta_b}}{\sin \theta_b}
\]  

(5)

The paper on Artistic Volumetric Lighting logically re-arranges the components of this equation into parameters describing each components physical contribution to the final radiance estimate. Once arranged as such, it becomes a simple matter of modifying the physically-based defaults for non-physically based parameters, allowing for artistic control over the shading of the beams [3].

1. \( f_1(\Phi_b, r_b) \Rightarrow \) applies bi-weight blurring kernel to the beam width
2. \( f_f(\sigma_s, \rho, \theta_b) \Rightarrow \) color change due to viewing angle
3. \( f_c(\sigma_t, t^c_b) \Rightarrow \) attenuation along line of sight
4. \( f_b(\sigma_t, t^b) \Rightarrow \) attenuation along beam path until intersect point

3.2 Implementation

1. To model beams, I began with the source code for the cylinder primitive as a base. Cylinders were propagated from the light source to the end of the volume. I then coded a recursive split function to separate the cylinder into sub-cylinders.
2. The sub-beams are stored in a reformulated Bounding Box Hierarchy.

3. The cylinders were then changed to cones, allowing for an adaptive beam width. The ending beam width is generated with the pdf of the light from which the cone is propagated.

4. Multiple-scattered beams remain cylinders, with their widths being the width of that point in the original cone from which the scattered beam originated. Each sub-beam has an id indicating to which larger beam it belongs.

5. At rendering time, duplicate beam-ids are removed, keeping only the beam closest to the line of sight. Plugging the values into Equation 5 returns the photon beam radiance estimate.

3.3 Results

Figure 4 shows a sphere with a spot light directly above, and a homogenous volume region encompassing the entire scene. My photon beam implementation has several marked improvements over my code for the beam-radiance estimate. The only control parameter is the number of the beams, and values such as step-size and the width of beams are automatically scaled to fit the volume region and the light’s probability distribution function. For that reason, the incompletely filled volume region of Figure 1 is not possible with this implementation. Although this program is also noticeably faster than my previous code, the scene in Figure 4 nonetheless took about an hour and half to render, which indicates ample room for future time optimization.

![Figure 4: Photo-realistic Photon Beams (my implementation)](image)

The artistic effects are rendered with fewer beams, however their effects are still immediately noticeable. In Figure 5, the attenuation along the beam is replaced instead with a sine wave
function, causing the radiance to ebb based on the input frequency. In Figure 6, the beam width is not considered, leading to an almost completely uniform shading along the horizontal lengths of the beams. The artistic effects are rendered with fewer beams, however their effects are still immediately noticeable. In Figure 5a, the attenuation along the beam is replaced instead with a sine wave function, causing the radiance to ebb based on the input frequency. In Figure 5b, the beam width is not considered, leading to an almost completely uniform shading along the horizontal lengths of the beams.

\[ f_{\text{phys}}(\sigma, t_b^c) = x + \sin y t_b^c \]

(b) \[ f_{\text{phys}}(\Phi_b, r_b) = 1 \]

Figure 5: Non-Physical Photon Beams

The source code for my photon beam implementation can be found [here](#), which contains both photonbeam.cpp and photonbeam.h.

4 Conclusion

In this project I implemented two systems for Global Illumination, and greatly improved both my programmatic skill (the second code is much cleaner) and my general knowledge regarding the mathematics underlying computer graphics. I mentioned possible room for improvements in my results, which, to once again summarize, are increased physical accuracy for the beam radiance estimate implementation, and more expedited render times for the photon beam implementation. Other miscellaneous skills gained from this project were learning how to use Github and Latex, which, although unrelated to the general purpose of my project, are rather invaluable skills. Having focused on such a small subset of ray-tracing thus far, I hope to code a workable ray tracer in the future for a more macro view of rendering.

References

