A Feature-Rich CPU Ray Tracer with Volumetric and Camera Effects

Aaron Miller – Dec 14th, 2025

Abstract

This project implements a physically-based CPU ray tracer supporting textured primitives, triangle meshes, and emissive materials. The renderer includes a spatial acceleration structure, stochastic sampling, and parallel execution. Several advanced features were implemented, including volumetric rendering, motion blur, depth of field, procedural textures, cube mapping, and smooth shading. The system was designed to be modular and extensible while maintaining reasonable performance on complex scenes.

1 System Overview

- CPU-based offline ray tracer
- Recursive path tracing
- Parallelized per-pixel rendering
- Modular components:
  - Camera
  - Geometry
  - Textures
  - Materials
  - Acceleration Structure

2 Implemented Features

2.1 Core Functionality: These features form the baseline physically-based rendering pipeline.

- Configurable Camera – Camera is fully customizable with position, direction, FOV, and resolution.
- Anti-Aliasing – Stochastic sampling on and around each rendered fragment removes aliasing.
- Ray-Sphere Intersections – Calculated with quadratic equation roots.
- Ray-Triangle Intersections – Calculated with plane-intersection and barycentric coordinates.
- Textured Spheres and Triangles – Implements UV texture coordinate mapping to wrap textures around geometry.
- Spatial Acceleration Structure (BVH) – Spatially subdivides world into nested, overlapping bounding boxes to allow object intersection checking with logarithmic time-complexity.
- Diffuse, Specular, Dielectric, and Emissive Materials – Portrays plausible light reflection, refraction, and emission. When relevant, materials use randomly chosen ray directions within a range instead of splitting rays.

2.2 Camera Effects: These features allow further creativity in scene composition.

- Motion Blur – Rays carry a time component and can move according to any arbitrary input position function. Bounding boxes are created using adaptive sampling analysis of the functions range over the chosen time interval.
- Depth of Field – Uses a thin lens camera model to simulate a focus distance. Objects diverging from the focus plane are distorted by a configurable defocus angle.

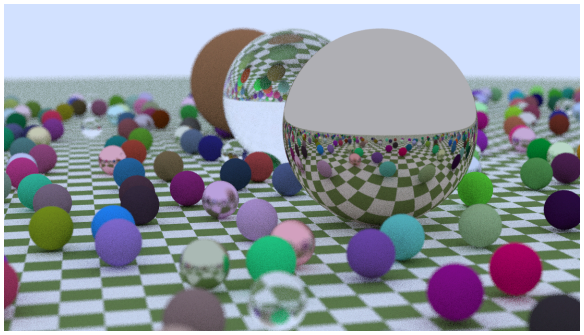
2.3 Advanced Rendering Features: These features significantly improve realism, flexibility, and performance.

- Triangle mesh loading (OBJ) – Parses OBJ files into triangles meshes for placement and transformation within a scene.
- Volumetric Rendering – Implements constant density participating media, e.g. smoke, fog, etc.
- Procedural Textures – Uses Perlin Noise to create deterministically random lambertian textures.
- Cube Maps – Wraps six axis aligned images around the world, creating a background environment.
- Quads – Quadratic primitives using plane-intersection testing and highly modular UV coordinate checking for flexible recreation of other primitives
- Normal Interpolation – Smooths triangle mesh shading by calculating per-intersection normals based on nearby vertex normals.
- Object Instancing - Allows a single instance of geometry to be referenced in several instances and easily translated across the scene.
- Parallelization – Used CPU multithreading with OpenMP for per-pixel rendering and significantly faster render times.

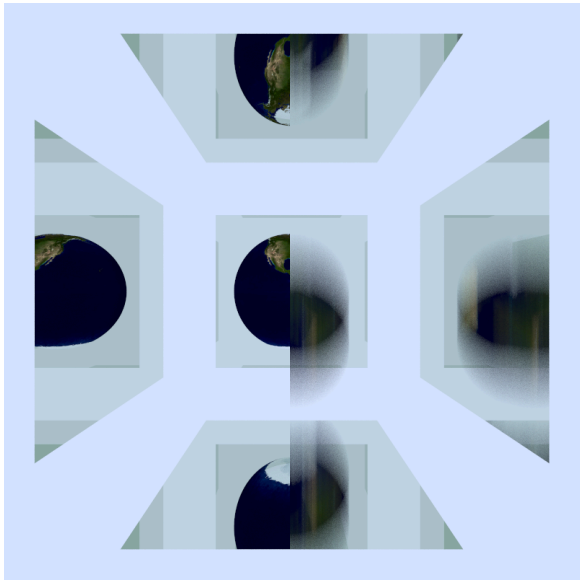
3 Results

Glass sphere with metal center in cubemap park.

Two dice triangle meshes loaded and shaded by nearby emissives. Only the right die uses normal interpolation for smoothing.



Large, complex scene rendered in only 20 seconds with BVH and multithreading.



Globe textured sphere surrounded by green-tinted specular quads.
Left: static, Right: falling

Conclusion

This project demonstrates the implementation of a feature-rich ray tracer with both physically-based materials and advanced rendering effects. Emphasis was placed on modular design, visual realism, and performance through acceleration and parallelization.

Future planned work will include importance sampling, GPU acceleration, simulation, and further exploration of linear algebra applications to ray tracing (e.g. Eckart Young Approximation in image compression and filtering).

Sources

Books:

[1] Peter Shirley, Trevor David Black, and Steve Hollasch.
Ray Tracing in One Weekend.
Online book series, 2018–2020.
https://raytracing.github.io/
(Accessed December 2025)

Software Libraries:

[2] Syoyo Fujita.
tinyobjloader: Wavefront OBJ File Loader.
Open-source C++ library.
https://github.com/tinyobjloader/tinyobjloader (Accessed December 2025)

[3] Sean Barrett.
stb_image.h: Single-File Public Domain Image Loader.
Open-source header-only library.
https://github.com/nothings/stb
(Accessed December 2025)

[4] OpenMP Architecture Review Board.
OpenMP Application Programming Interface.
Parallel programming standard.
https://www.openmp.org/
(Accessed December 2025)

Assets:

[5] Emil Persson (Humus).
Cube Map Textures.
Personal website, licensed under Creative Commons Attribution 3.0 Unported.
https://www.humus.name
(Accessed December 2025)

[6] Unknown author.
Low-Poly Dice Model with Levels of Detail.
OpenGameArt.org.
Licensed under CC0 1.0 Universal (Public Domain Dedication).
https://opengameart.org/
(Accessed December 2025)