🎓 Ray Tracing Project Report

Acceleration Structures, Super sampling Techniques,
and System Optimization

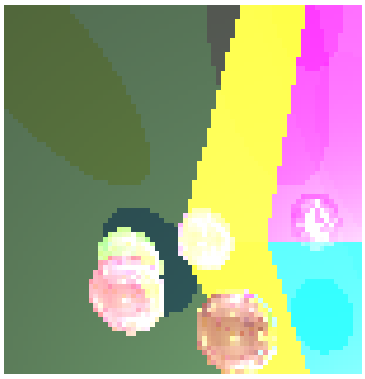
Prepared by:

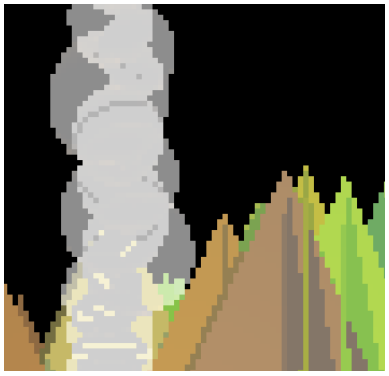
Lidan Rubinov
Dror Yakov Hai

With the support of Eliezer Gensburger

Minip 1: Super Sampling

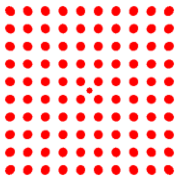
Background: **Up until this point, our project images had been pretty average, lacking detail and too pixelated.**



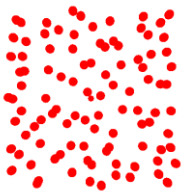


The problem is quite understandable, since in the project we used a single ray to find out what the color of each pixel was (whether for a shadow or for a color, etc.). Therefore, we defined a new class called Blackboard whose job would be to receive a target area where it would produce lots of points in one of the following methods:

GRID



RANDOM



JITTERED

For each of the points the class will construct a ray from the base point.

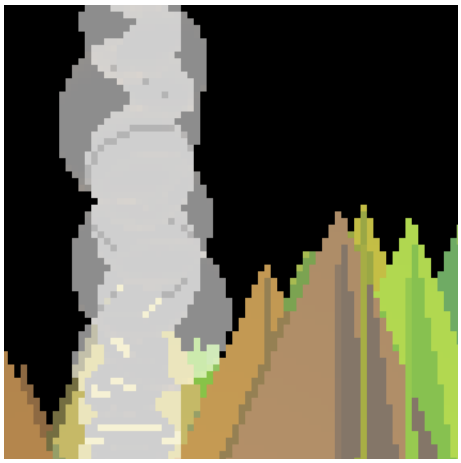
RAY = (base, vector (base - point))

Now, after we constructed the sample rays, we will use them to create effects on our images in the following ways:

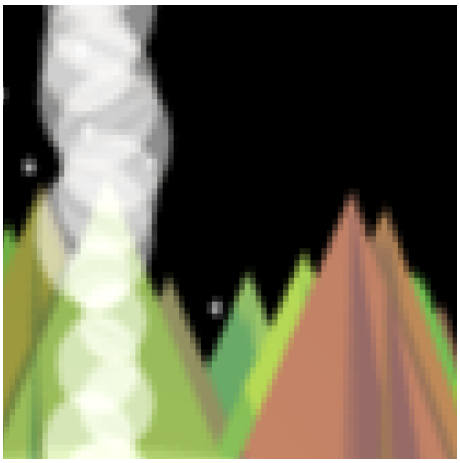
1 Anti-Aliasing (AA)

Anti-aliasing is used to smooth out jagged edges caused by discrete pixel sampling. We achieved this by shooting multiple rays within each pixel using a grid-based approach. The colors of the sub-rays are averaged to obtain the final pixel color. This reduces stair-stepping artifacts and improves image quality.

before



after



2 Soft Shadows

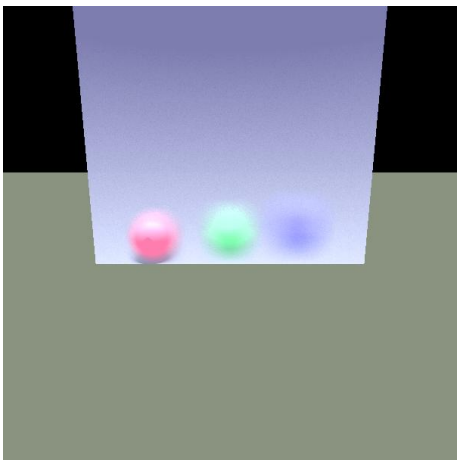
Soft shadows simulate area light sources by casting multiple rays toward the light area. Instead of a single shadow ray, multiple rays are sampled within the area light's extent. The shadow intensity is determined by the number of rays reaching the light without obstruction. This results in penumbra effects and more natural shadows.

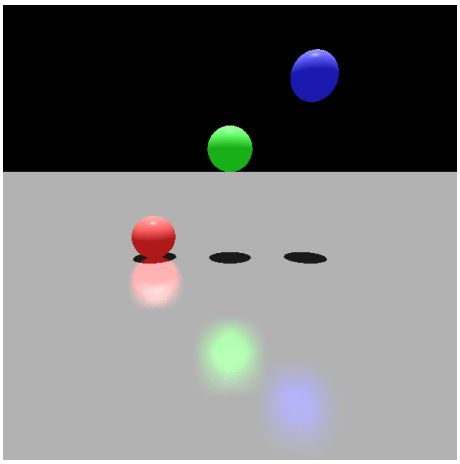
before

after

3 Glossy and Blurry Surfaces

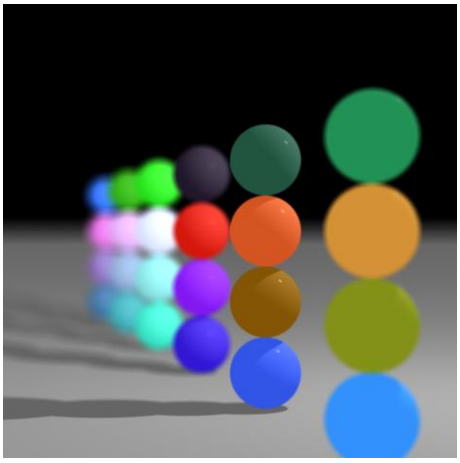
Glossy reflections and blurry refractions simulate imperfect surfaces. Instead of a perfect reflection/refraction direction, we jitter rays within a cone defined by the surface roughness. Multiple rays are traced and their resulting colors averaged to produce a blurry effect that mimics realistic materials.

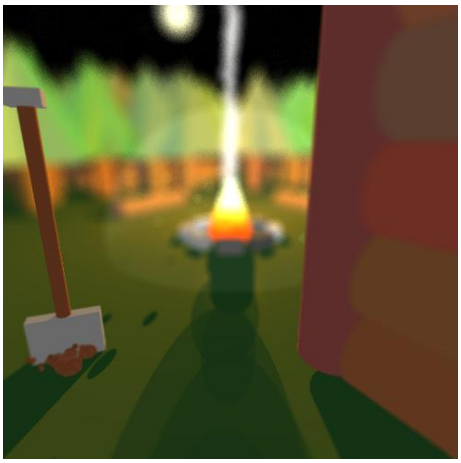




4 Depth of Field (DoF)

Depth of Field simulates a camera lens with finite aperture. Rays are generated from various positions on a lens (aperture), and all converge to a focus point. Objects near the focus distance remain sharp, while others appear blurred. This was achieved by constructing rays that originate from the aperture and aim toward a common focus point.



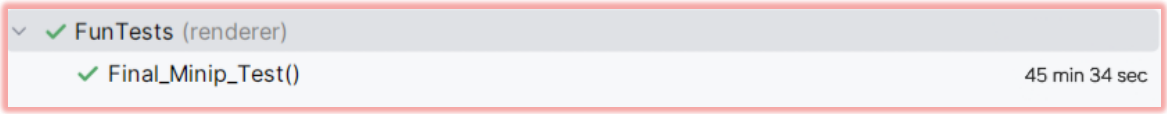


Together, these super sampling techniques enhance visual realism by simulating complex optical phenomena and reducing visual artifacts, making the rendered images more lifelike and immersive.

🚀 Minip 2: Acceleration and Super sampling Techniques in Ray Tracing

🔧 Introduction

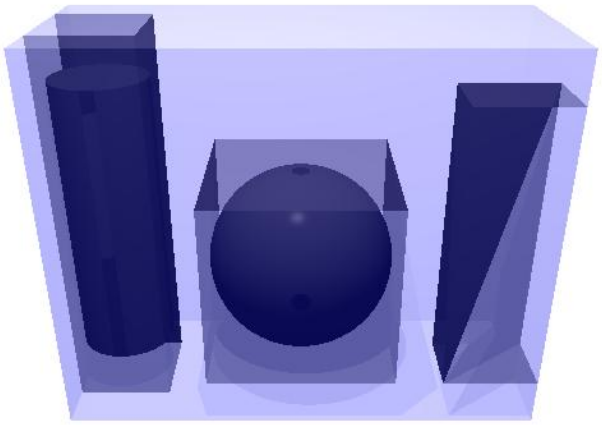
In high-quality ray tracing, achieving visually stunning results often requires advanced sampling techniques and efficient data structures. Super sampling methods such as anti-aliasing, soft shadows, depth of field, and glossy reflections greatly enhance image realism—but tuning their parameters (like sample count, aperture size, focus distance, and recursion depth) can dramatically increase render times.



To mitigate this, we introduce multi-threaded rendering and spatial acceleration structures. Multi-threading allows pixel-level parallelism, where each thread is responsible for tracing rays and computing colors for separate image regions, significantly reducing total rendering time. However, even with threads, optimization is essential when handling complex geometry and light interactions across large scenes. This is where acceleration grids and adaptive techniques come into play.

📦 Axis-Aligned Bounding Boxes (AABB)

AABBs are bounding volumes used to enclose individual geometries or entire groups of objects. They provide a quick rejection test for ray-geometry intersection: if a ray does not intersect the AABB, we can safely skip intersection tests for all inner geometries, saving computational effort. Each AABB is defined by a minimum and maximum point, creating a rectangular volume aligned to world axes. The intersection test with the ray is efficient and forms the first line of defense in reducing unnecessary calculations.



3D Digital Differential Analyzer (3D DDA)

To traverse a regular grid of voxels efficiently, we use the 3D DDA algorithm. This algorithm steps through a 3D voxel grid in the direction of the ray by incrementally moving along the voxel boundaries in x, y, and z axes. It uses the ray direction and precomputed tDelta values to determine the smallest next intersection time and advances to the corresponding voxel. This results in a highly optimized walk-through space, skipping empty regions and minimizing intersection checks.

$$\text{voxel}_x = \left\lfloor \frac{x_0 - x_{\min}}{\text{voxelSize}_x} \right\rfloor \qquad \text{voxel}_y = \left\lfloor \frac{y_0 - y_{\min}}{\text{voxelSize}_y} \right\rfloor \qquad \text{voxel}_z = \left\lfloor \frac{z_0 - z_{\min}}{\text{voxelSize}_z} \right\rfloor$$

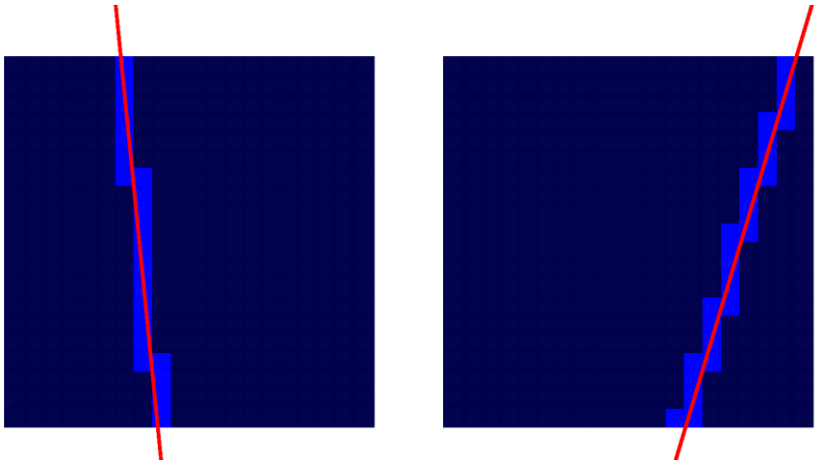
$$\text{tDelta}_x = \frac{\text{voxelSize}_x}{|d_x|} \qquad \text{tDelta}_y = \frac{\text{voxelSize}_y}{|d_y|} \qquad \text{tDelta}_z = \frac{\text{voxelSize}_z}{|d_z|}$$

$$\text{tMax}_x = \text{tDelta}_x \cdot \begin{cases} 1 - \left(\frac{x_0 - x_{\min}}{\text{voxelSize}_x} - \left\lfloor \frac{x_0 - x_{\min}}{\text{voxelSize}_x} \right\rfloor \right) & \text{if } d_x > 0 \\ \left(\frac{x_0 - x_{\min}}{\text{voxelSize}_x} - \left\lfloor \frac{x_0 - x_{\min}}{\text{voxelSize}_x} \right\rfloor \right) & \text{if } d_x < 0 \\ \infty & \text{if } d_x = 0 \end{cases}$$

$$\text{tMax}_y = \text{tDelta}_y \cdot \begin{cases} 1 - \left(\frac{y_0 - y_{\min}}{\text{voxelSize}_y} - \left\lfloor \frac{y_0 - y_{\min}}{\text{voxelSize}_y} \right\rfloor \right) & \text{if } d_y > 0 \\ \left(\frac{y_0 - y_{\min}}{\text{voxelSize}_y} - \left\lfloor \frac{y_0 - y_{\min}}{\text{voxelSize}_y} \right\rfloor \right) & \text{if } d_y < 0 \\ \infty & \text{if } d_y = 0 \end{cases}$$

$$\text{tMax}_z = \text{tDelta}_z \cdot \begin{cases} 1 - \left(\frac{z_0 - z_{\min}}{\text{voxelSize}_z} - \left\lfloor \frac{z_0 - z_{\min}}{\text{voxelSize}_z} \right\rfloor \right) & \text{if } d_z > 0 \\ \left(\frac{z_0 - z_{\min}}{\text{voxelSize}_z} - \left\lfloor \frac{z_0 - z_{\min}}{\text{voxelSize}_z} \right\rfloor \right) & \text{if } d_z < 0 \\ \infty & \text{if } d_z = 0 \end{cases}$$

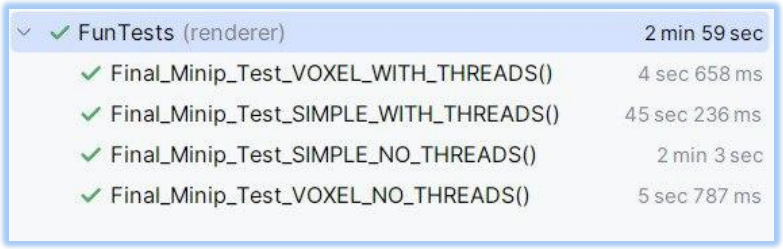
The initial voxel is calculated from the ray's origin using flooring and offsetting based on voxel size. The algorithm tracks tMax for each axis, which indicates the parametric distance to the next voxel boundary and updates the current voxel accordingly. It terminates when the ray exits, the grid bounds or finds a valid intersection.



Render Time Considerations

Rendering time is influenced by the number of rays per pixel, the complexity of the scene, and the number of secondary rays produced by effects like shadows, reflections, and refractions. Super sampling introduces multiple rays per pixel—anti-aliasing alone can quadruple the ray count or more. When combined with effects like soft shadows and glossy surfaces, this can lead to exponential growth in ray tracing workload.

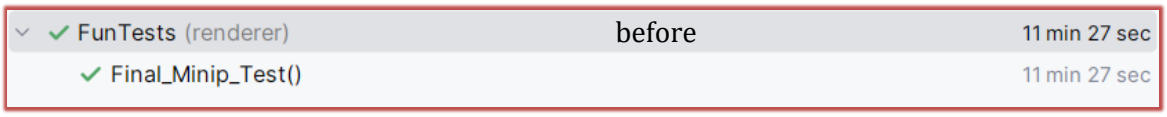
By integrating acceleration techniques like voxel grids, AABBs, and parallel threads, we reduce the per-ray cost and spread the workload efficiently across available CPU cores. Still, adaptive super sampling remains critical to prioritize expensive sampling only in regions that exhibit high-frequency changes, thus balancing quality with performance.



Adaptive Super sampling

Adaptive super sampling improves performance by focusing extra sampling only where it's visually necessary, typically where color or lighting changes rapidly across a pixel. Instead of shooting a fixed grid of rays per pixel, we recursively subdivide the pixel and sample corners. If the resulting colors are similar (within a specified threshold), we stop. Otherwise, we subdivide further.

This approach avoids over-sampling uniform areas (like the sky or flat walls), while preserving sharp edge and fine detail around geometry boundaries, soft shadows, or highlights. It plays particularly well with depth of field and blurry reflections, which tend to introduce subtle gradients that benefit from adaptive sampling.



Together, these acceleration techniques—bounding volumes like AABB, spatial partitioning with voxel grids, efficient traversal via 3D DDA, and parallelized execution through multi-threading—create a powerful framework that drastically reduces rendering time while maintaining high visual fidelity, enabling complex scenes to be rendered with both speed and precision.

Phong Shading Model

Phong Shading is a powerful per-pixel shading technique used to create smooth, realistic lighting on 3D objects. Unlike simpler methods like Grouard shading, which calculate light only at the vertices and interpolate the colors, Phong shading calculates the lighting at **every single pixel** on a surface.

What does that mean?

First, the system estimates how each point on a surface is oriented in space — this is done by interpolating the normals (direction vectors) between the vertices of the shape. This creates a smooth gradient of orientation across the surface.

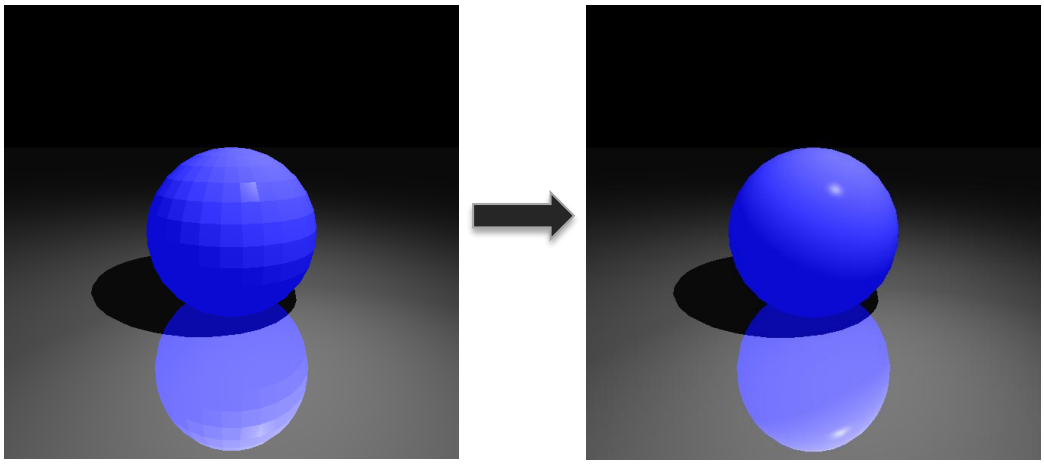
Next, using these interpolated normals, the lighting is calculated for every pixel based on the direction of the light source and the viewer's position. The result is a **smooth transition of light and shadow**, with **realistic highlights** that reflect properly as the object or camera moves. This gives even low polygon objects the appearance of being curved and detailed.

Why is Phong shading useful?

Because it offers a great balance: it's more realistic than vertex-based shading, but still efficient enough for real-time rendering. It's especially effective for shiny or smooth materials like metal, plastic, or skin — anything that needs soft lighting transitions and responsive specular highlights.

When should you use Phong shading?

Any time you want a model to look smooth and polished, even if it's made of simple geometry. It's ideal for curved surfaces, detailed lighting effects, and polished materials in both real-time engines and offline rendering.



Our Journey with GPT

Throughout this project, we led the development of a custom ray tracing engine, tackling challenges in rendering, optimization, and system design. From implementing features like anti-aliasing, depth of field, soft shadows, and glossy/blurry surfaces to refining adaptive super sampling and voxel traversal, we made every architectural and technical decision ourselves.

To support our work and accelerate experimentation, we used GPT as a technical assistant. Whenever we had a specific question—whether to structure adaptive sampling logic, deriving equations for 3D DDA, or improving AABB-based acceleration, we described the issue in natural language and received helpful suggestions, formulas, or code examples tailored to our needs.

This approach allowed us to stay focused on the creative and engineering aspects of the project, while using GPT as a rapid reference and idea generator. We developed a flexible, multi-threaded renderer with spatial acceleration and adaptive sampling that intelligently responds to scene complexity. Along the way, we also documented our work thoroughly, producing styled reports and technical explanations.

We directed the process entirely by reviewing, modifying, and integrating suggestions according to our goals. GPT served as a supporting tool in our hands, helping us move faster without compromising our vision or control.

This experience wasn't just about solving problems—it was about combining our knowledge and creativity with smart tools to bring a cinematic-quality renderer to life.

Using GPT-4 throughout this journey gave us a responsive and knowledgeable partner—one that helped us validate ideas, accelerate development, and refine our technical decisions, while keeping full creative control in our hands.

The end
Thank u for reading.