CGRCW2 - Raytracer/Pathtracer

November 25, 2024

Contents

1 Basic Raytracer Features **2**
- 1.1 a. Image Writing . . . 2
- 1.2 b. Virtual Pin-hole Camera . . . 2
- 1.3 c. Intersection Tests . . . 2
- 1.4 d. Blinn-Phong Shading . . . 3
- 1.5 e. Shadows . . . 4
- 1.6 f. Tone Mapping . . . 5
- 1.7 g. Reflection . . . 5
- 1.8 h. Refraction . . . 6

2 Intermediate Raytracer features **6**
- 2.1 a. Textures . . . 6
- 2.2 b. Acceleration Hierarchy . . . 8

3 Advanced Raytracer Features **9**
- 3.1 a. Pixel Sampling . . . 9
- 3.2 b. Lens Sampling . . . 9
- 3.3 c. BRDF Sampling . . . 10
- 3.4 d. Light Sampling . . . 11

4 Conclusion **12**

1 Basic Raytracer Features

1.1 a. Image Writing

The image writing feature enables the raytracer to output images in the PPM format.
Implementation:

- Used C++ standard file I/O streams to write image data.
- Stored the image as a 2D array of colour values and exported it in PPM format.

Evaluation: A PPM image displaying a gradient background from green to red was successfully generated, confirming ray direction calculation and pixel colour mapping.

Figure 1: Generated gradient background.

1.2 b. Virtual Pin-hole Camera

The virtual pinhole camera generates rays for rendering based on position, orientation, and field of view.
Implementation:

- Calculated camera basis vectors (w, u, v) to define orientation in 3D space.
- Derived viewport dimensions using the vertical field of view.
- Computed the lower-left corner of the viewport and directional vectors (horizontal and vertical) relative to the camera's position.
- Implemented ray generation through the viewport for given (u, v) coordinates.

Evaluation: The camera generated rays that accurately mapped to the scene geometry, ensuring correct image projection in the render.

1.3 c. Intersection Tests

Intersection tests determine if and where a ray intersects objects in the scene, enabling accurate geometry rendering.
Implementation:

- Developed the 'Hittable' interface and 'HitRecord' structure to store intersection details (point, normal, hit distance).
- Implemented a 'Sphere' class using the quadratic formula for ray-sphere intersection.

- Created a ‘HittableList‘ to manage multiple objects and evaluate the nearest intersection.
- Ensured surface normals in ‘HitRecord‘ always face the incoming ray.

Evaluation: Tests with multiple spheres validated correct hit points and normals. The first rendered image, generated using a virtual pinhole camera and ray-sphere intersection, featured a sphere at $(0, 0, -1)$ with radius 0.5. The camera was positioned at $(3, 3, 2)$, looking towards $(0, 0, -1)$ with a 60° FOV. The sphere rendered successfully with surface normal shading and a gradient background.

Figure 2: First rendered scene with a single sphere.

1.4 d. Blinn-Phong Shading

Blinn-Phong shading provides realistic lighting effects by combining ambient, diffuse, and specular components.
Implementation:

- Developed a ‘BlinnPhongMaterial‘ class extending the base ‘Material‘ class for shading calculations.
- Ambient component: constant colour added to the surface.
- Diffuse component: dot product between surface normal and light direction.
- Specular component: based on the angle between the half-vector (average of view and light directions) and surface normal, raised to a shininess factor.
- Supported multiple light sources by accumulating their contributions.
- Replaced basic shading in the rendering loop with material-based shading at intersections.

Evaluation: The first render using Blinn-Phong shading featured a red sphere at $(0, 0, -1)$ (radius 0.5) and a ground sphere at $(0, -100.5, -1)$ (radius 100). A light source at $(5, 5, -5)$ illuminated the scene. The rendered image showed specular highlights, diffuse shading, and ambient lighting, confirming the model's correct implementation.

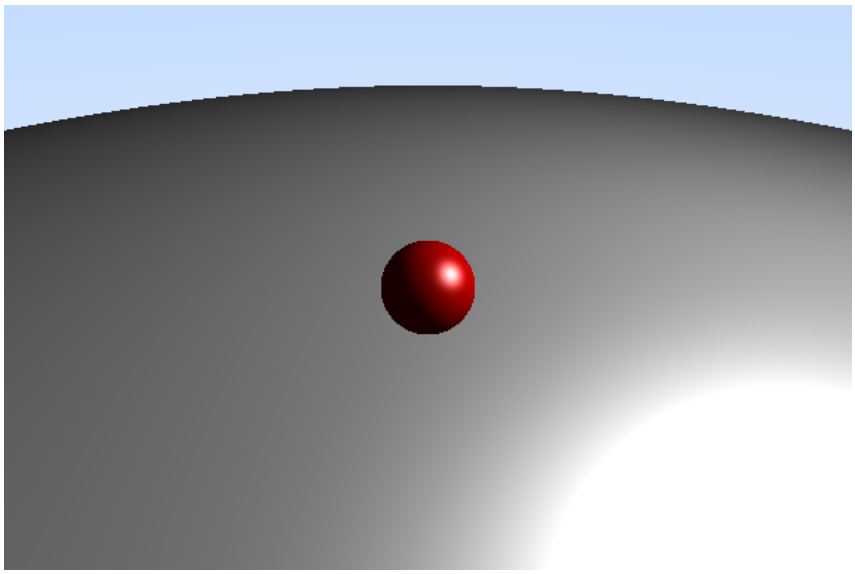

Figure 3: First render using Blinn-Phong shading with one light source.

1.5 e. Shadows

Shadows enhance realism by accounting for light obstruction caused by objects.
Implementation:

- Cast shadow rays from hit points to each light source.
- Checked for intersections along shadow rays to determine if a point is in shadow.
- Integrated shadow calculations into the ‘BlinnPhongMaterial‘ class, ensuring light contributions are excluded for shadowed points.
- Adjusted diffuse and specular shading based on shadowing.

Evaluation: The implementation was tested with scenes containing multiple objects and a single light source. Two red spheres at $(0, 0, -1)$ and $(-1.2, 0, -1)$, along with a large ground sphere at $(0, -100.5, -1)$, demonstrated accurate shadows on the ground and between objects, significantly improving depth and realism.

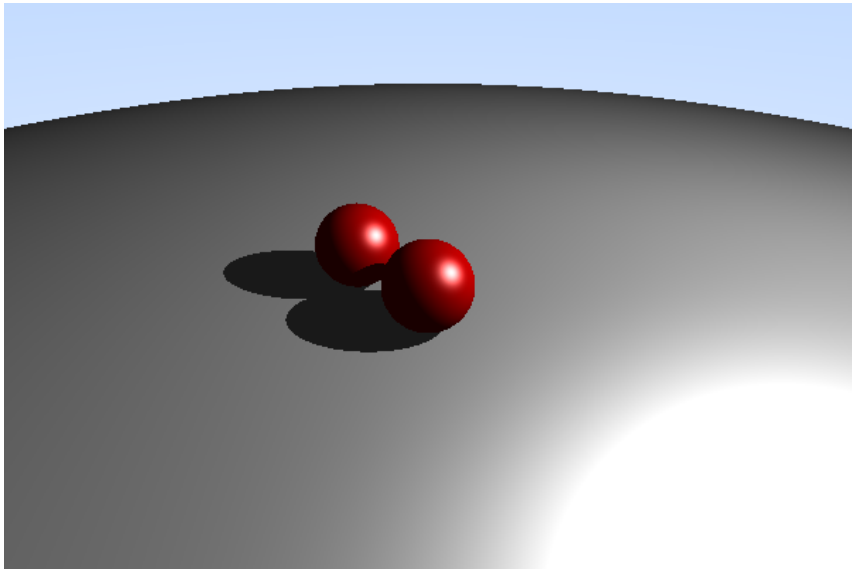

Figure 4: Rendered scene with shadows, showing accurate light obstruction.

1.6 f. Tone Mapping

Tone mapping compresses high dynamic range (HDR) values into a displayable range, enhancing image quality.
Implementation:

- Created a ‘ToneMapper‘ interface and implemented ‘ReinhardToneMapper‘ for global tone mapping.
- Applied the formula: $\text{colour}_{\text{mapped}} = \frac{\text{colour}}{\text{colour}+1}$.
- Integrated tone mapping into the image save pipeline to process all pixel values before output.
- Added gamma correction for consistent display.

Evaluation: Images with high intensity values were rendered with balanced brightness, avoiding overexposure while preserving detail. Using the same scene as in the shadows evaluation, tone mapping effectively compressed HDR values, particularly improving brightness balance around reflections on the ground sphere.

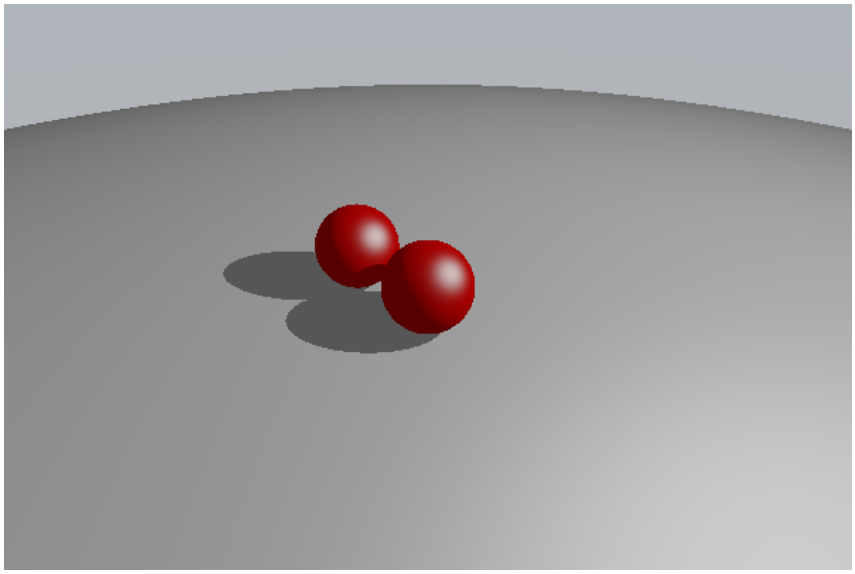

Figure 5: Rendered scene with tone mapping applied.

1.7 g. Reflection

Reflection simulates mirror-like surfaces, enhancing realism by rendering object and environment reflections.
Implementation:

- Calculated reflection vectors using $\text{reflect} = \text{view} - 2 \cdot (\text{view} \cdot \text{normal}) \cdot \text{normal}$.
- Cast secondary rays along reflection vectors to compute reflected colours.
- Combined reflective contributions with Blinn-Phong shading in the rendering pipeline.

Evaluation: A scene featuring a red sphere at $(0, 0, -1)$, a reflective sphere at $(-1.2, 0, -1)$, and a ground sphere was rendered. The reflective sphere accurately mirrored the red sphere and surrounding environment, with reflections behaving correctly at various angles. This confirmed the successful integration of reflection into shading.

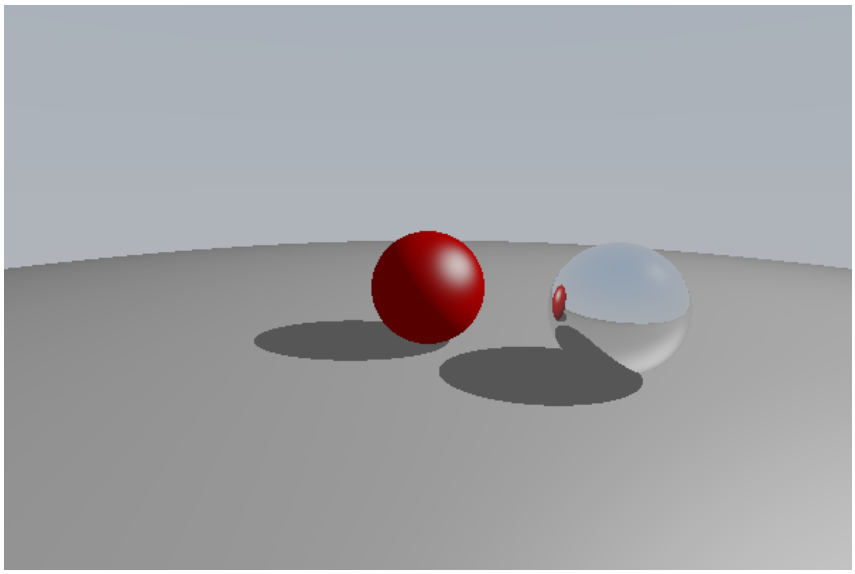

Figure 6: Rendered scene with a reflective sphere.

1.8 h. Refraction

Refraction simulates transparent materials like glass, bending light through objects for added realism.
Implementation:

- Calculated refraction vectors using Snell's law.
- Applied Fresnel effects with Schlick's approximation for blending reflection and refraction.
- Accounted for total internal reflection (TIR) when the incidence angle exceeded the critical angle.
- Cast secondary rays for refracted light and combined their contributions with Phong shading.

Evaluation: Refraction was tested under various scenarios:

- **Basic Refraction:** Light bending was validated (Fig. 11), though without Fresnel effects, visuals were less realistic.
- **With Fresnel and TIR:** Improved realism with Fresnel effects (angle-dependent reflection) and accurate TIR for critical angles (Fig. 12).
- **Hollow Glass Sphere:** Demonstrated correct handling of nested IOR transitions with a hollow sphere (Fig. 9).

2 Intermediate Raytracer features

2.1 a. Textures

Textures add surface details to objects, enabling the application of realistic patterns and images.
Implementation:

- Developed an abstract 'Texture' class with methods to calculate colour based on texture coordinates.

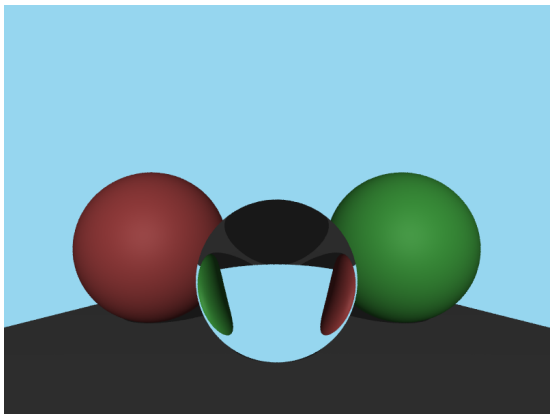

Figure 7: Basic Refraction (No Fresnel or TIR).

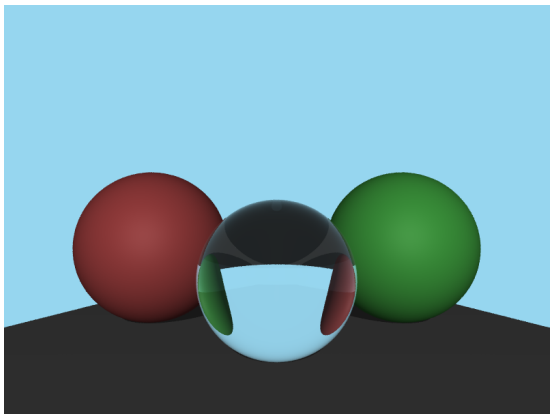

Figure 8: Refraction with Fresnel Effects and TIR.

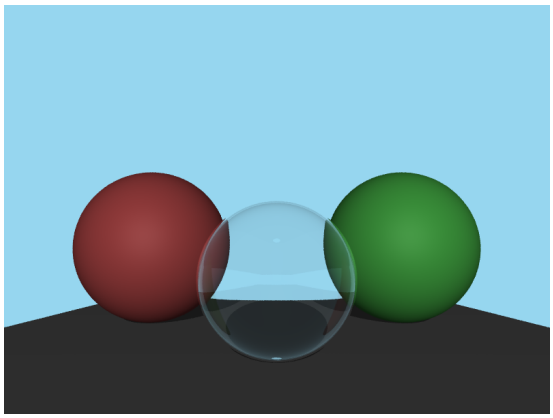

Figure 9: Hollow Glass Sphere (Air Interior).

- Implemented 'ImageTexture' for mapping PPM texture files:
  - Loaded PPM files, storing pixel data as RGB arrays.
  - Used bilinear interpolation for smooth mapping, clamping u and v to $[0, 1]$.
- Supported planar mapping for triangles, cylindrical mapping for cylinders, and spherical mapping for spheres.

Evaluation: Textures were tested on spheres, cylinders, and triangles.

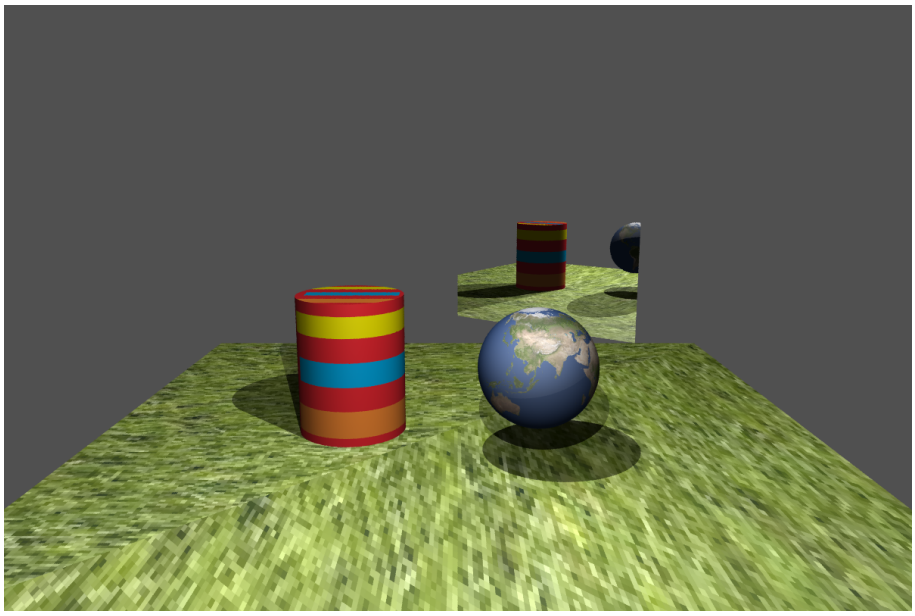

Figure 10: Rendered scene showcasing texture mapping on spheres, cylinders, and planar surfaces.

2.2 b. Acceleration Hierarchy

The acceleration hierarchy, implemented using a Bounding Volume Hierarchy, accelerates rendering performance by reducing the number of intersection tests in complex scenes.
Implementation:

- Created an Axis-Aligned Bounding Box class:
  - Implemented ray-box intersection tests.
  - Added methods to compute the surrounding box of two AABBs.
- Developed a 'BVHNode' class to recursively partition the scene into a binary tree:
  - Sorted objects along a random axis for optimal tree construction.
  - Leaf nodes stored individual objects, while internal nodes grouped objects with their AABBs.
  - Implemented hit logic to traverse the tree, prioritising bounding box checks before object intersections.
- Updated 'Hittable' and derived classes to include a 'bounding_box' method for BVH.

Evaluation: BVH significantly reduced unnecessary intersection tests, improving rendering times. For the scene 'scenes/scene_bvh.json', performance results are:

```
With BVH:
Total time: 25.5 seconds
Average time per pixel: 0.031875 ms

Without BVH:
Total time: 37.5 seconds
Average time per pixel: 0.046875 ms
```

The BVH implementation achieved a 32% reduction in total render time.

3 Advanced Raytracer Features

3.1 a. Pixel Sampling

Pixel sampling employs multiple techniques to balance image quality and performance.
Implementation:

- **Uniform Random Sampling:** Basic Monte Carlo integration with random pixel sampling.
- **Stratified Sampling:** Divides each pixel into a grid for evenly distributed samples, reducing noise.
- **Importance Sampling:** Dynamically allocates samples based on luminance importance, focusing on high-variance areas.
- **Post-Process Filtering:** Applied Gaussian blur and bilateral denoising to further reduce noise.

Evaluation: Uniform sampling established a baseline, while stratified and importance sampling significantly reduced noise and optimised computation. Post-processing filters improved the final image by minimising residual noise.

- **Uniform Sampling:** Rendered a 1000x1000 image with 500 samples in 243.981 seconds (Fig. 11).
- **Importance Sampling:** Rendered the same scene with adaptive sampling (250–10,000 samples), reducing render time by 75% to 65.723 seconds, albeit with slightly more noise (Fig. 12).

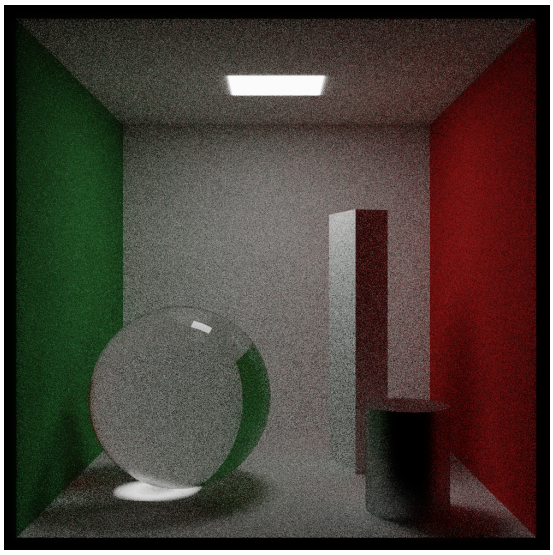

Figure 11: Uniform Sampling Render

3.2 b. Lens Sampling

Lens sampling simulates depth of field by modelling the camera lens as an aperture, where rays are traced through different parts of the lens. This creates realistic blurring for objects outside the focus distance, mimicking real cameras.
Implementation:

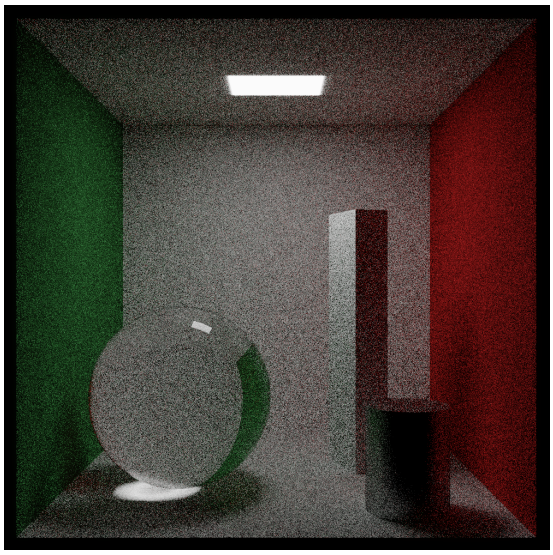

Figure 12: Importance Sampling Render

- Extended the `Camera` class to include lens sampling for depth of field:
  - Introduced a lens_radius variable, calculated from the aperture size.
  - Implemented random_in_unit_disk method to sample random points on a unit disk, scaled by the lens_radius.
- Modified Camera::get_ray:
  - Offset the ray origin using sampled points on the lens aperture: `offset = u * rd.x + v * rd.y; ray_origin = origin + offset`.
  - Adjusted the ray direction to point to the focus plane: `direction = target - ray_origin`.

Evaluation: Depth of field was tested in scene/path_dof.json with an aperture of 0.6 and focus distance of 6.0. The centre cube remained sharp, while objects outside the focus distance blurred realistically. Fig. 13 demonstrates the accurate depth of field effect.

3.3 c. BRDF Sampling

BRDF sampling models how light interacts with surfaces, enabling realistic simulation of direct and indirect lighting.
Implementation:

- Implemented the `brdf` function for each material:
  - *Diffuse*: Used Lambertian BRDF, returning $\frac{m\_albedo}{\pi}$ for perfect diffuse reflection.
  - *Metal*: Calculated specular reflection using a microfacet model with roughness and albedo parameters.
  - *Dielectric/Emissive*: BRDF not applied to these materials.
- Incorporated BRDF sampling in `Pathtracer::trace`:
  - *Direct Lighting*: Sampled light sources, checked for shadows, and computed light contributions using the material's BRDF.

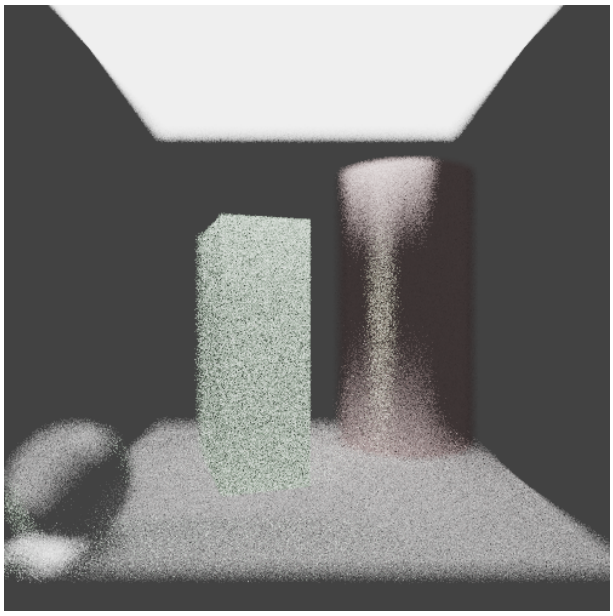

Figure 13: Depth of field effect using lens sampling (aperture = 0.6, focus distance = 6.0).

  - *Indirect Lighting*: Generated scattered rays using the BRDF's probability distribution and recursively traced rays for indirect contributions, weighted by BRDF values and adjusted for the PDF.

- Implemented Russian Roulette termination to limit recursion depth based on throughput.
- Used importance sampling to favour high-BRDF-value directions, reducing noise and improving convergence.

Evaluation: BRDF sampling accurately represented material behaviours:

- Diffuse surfaces displayed soft shading and correct light scattering.
- Metallic surfaces showed realistic specular reflections.
- Balanced integration of direct and indirect lighting produced natural shadows and illumination.

The effects were tested in `scene/cornell_box.json`, which includes all material types.

3.4 d. Light Sampling

Light sampling calculates direct illumination by evaluating contributions from light sources, including shadows and attenuation.
Implementation:

- Sampled each light by generating direction vectors from the intersection point to the light source.
- Computed light intensity using the inverse square law for attenuation based on distance.
- Cast shadow rays to detect occlusions and determine if light contributes to the surface.
- For unblocked lights, calculated direct lighting as:

  Light Contribution = (attenuated intensity)·(BRDF value)·cos(angle between light direction and surface

- Integrated light sampling within the direct lighting component of the path tracing algorithm for accurate representation of direct illumination.

Evaluation: Light sampling was validated in `scene/cornell_box.json`, demonstrating accurate direct illumination, including shadows and realistic light attenuation.

4 Conclusion

The final Cornell box scene was rendered at a resolution of 1000x1000 with a minimum of 500 samples and a maximum of 10000 samples per pixel. Gaussian blur and denoising were also applied to improve render quality. The result is shown in Fig. 14.

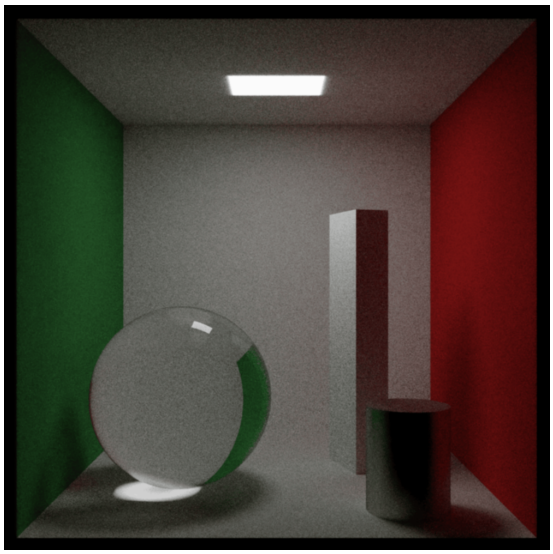

Figure 14: Final render with all features combined.