CSC4140 Assignment VI

Computer Graphics

April 24, 2022

Geometry

Student ID: 119010434

Student Name: Zhang Qihang

This assignment represents my own work in accordance with University regulations.

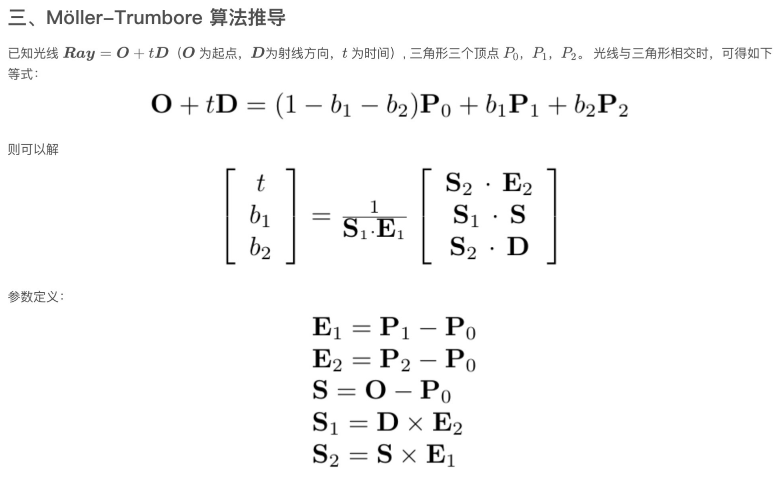
Signature:

1 Overview

I have implemented the ray-tracing pipeline, with Ray intersection, BVH, Direct Illumination and Indirect Illumination and Adaptive Sampling.

2 Task1

Using the Moller-Trumbore Algorithm, the algorithm uses vector and matrix calculations to quickly obtain the intersection and barycentric coordinates without pre-calculating the plane equation containing triangles.



三、Möller–Trumbore 算法推导

已知光线 $\boldsymbol{Ray} = \boldsymbol{O} + t\boldsymbol{D}$（$\boldsymbol{O}$ 为起点，$\boldsymbol{D}$为射线方向，t 为时间），三角形三个顶点 P_0，P_1，P_2。光线与三角形相交时，可得如下等式：

$$\mathbf{O} + t\mathbf{D} = (1 - b_1 - b_2)\mathbf{P}_0 + b_1\mathbf{P}_1 + b_2\mathbf{P}_2$$

则可以解

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\mathbf{S}_1 \cdot \mathbf{E}_1} \begin{bmatrix} \mathbf{S}_2 \cdot \mathbf{E}_2 \\ \mathbf{S}_1 \cdot \mathbf{S} \\ \mathbf{S}_2 \cdot \mathbf{D} \end{bmatrix}$$

参数定义：

$$\begin{aligned} \mathbf{E}_1 &= \mathbf{P}_1 - \mathbf{P}_0 \\ \mathbf{E}_2 &= \mathbf{P}_2 - \mathbf{P}_0 \\ \mathbf{S} &= \mathbf{O} - \mathbf{P}_0 \\ \mathbf{S}_1 &= \mathbf{D} \times \mathbf{E}_2 \\ \mathbf{S}_2 &= \mathbf{S} \times \mathbf{E}_1 \end{aligned}$$

Figure 1: Moller-Trumbore Algorithm

I use some simple graph to replace normal texture graph.

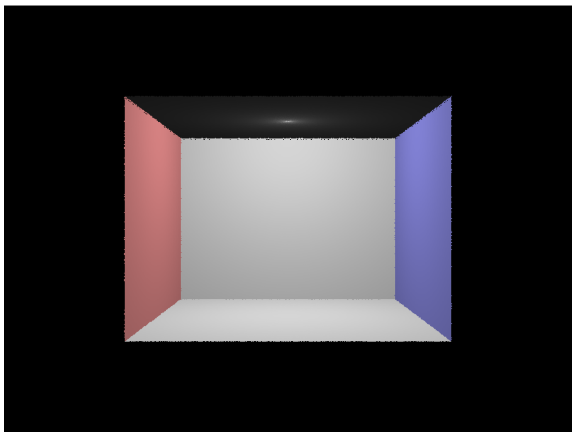


Figure 2: Graph 1

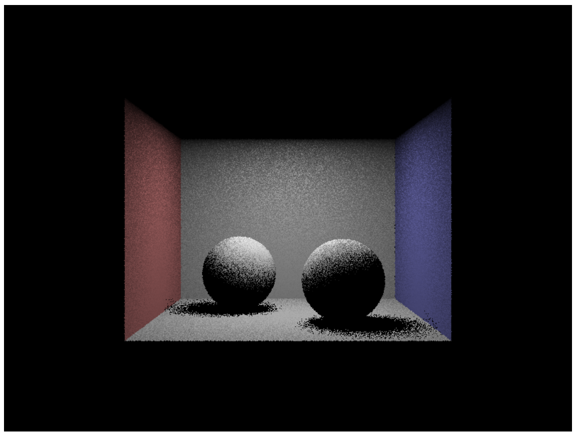


Figure 3: Graph 2

3 Task2

3.1 BVH algorithm

First, it is necessary to calculate the midpoint of the segmentation along each axis according to the object in the root BOX, and then calculate the segmentation on the three axes of x, y, and z respectively, and the criterion used is yielded the lowest cost. The number of objects in the left and right sub-BVHs is multiplied by the area of the BBOX. If the cost calculated by a certain axis is the smallest, this axis is used as the axis for the final BBOX cutting.

It can greatly reduce the number of computations required for intersection and improve the speed of intersection computation.

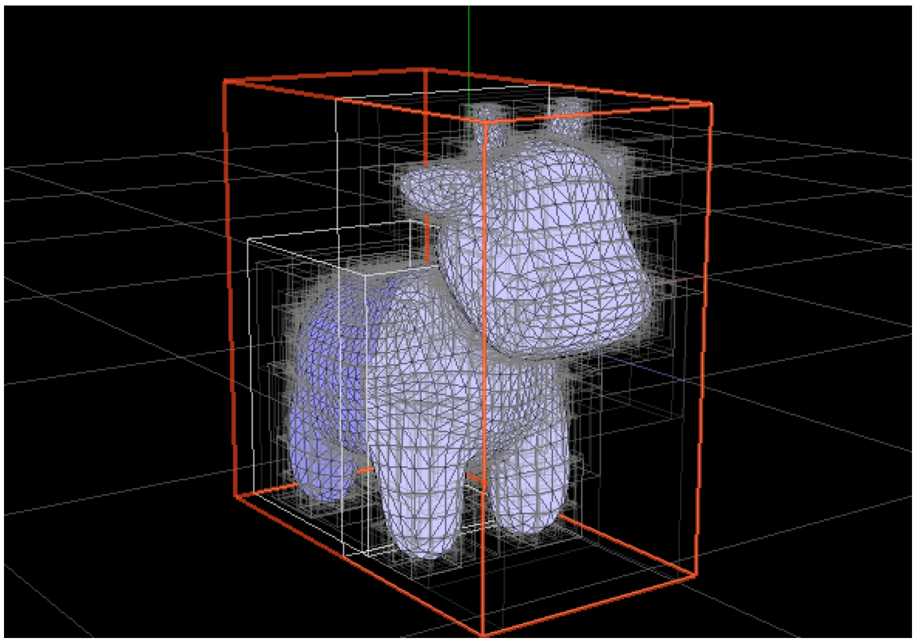


Figure 4: BVH

4 Task3

Uniform Hemisphere Sampling Renders

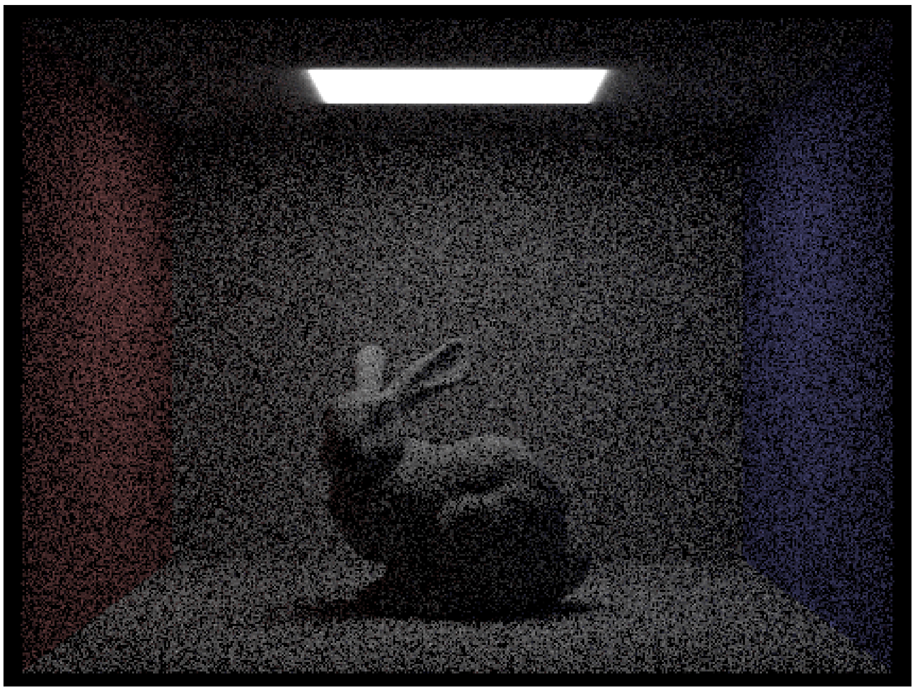


Figure 5: Uniform Hemisphere Sampling Renders with -t 8 -s 16 -l 8 -H

Importance Sampling Renders

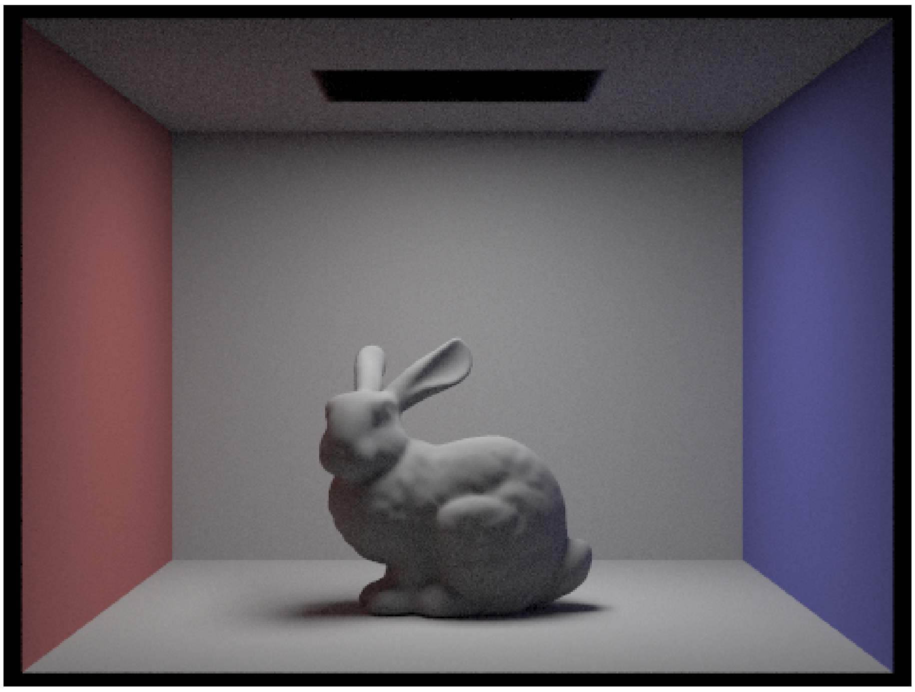


Figure 6: Importance Sampling Renders with -t 8 -s 64 -l 32 -m 6

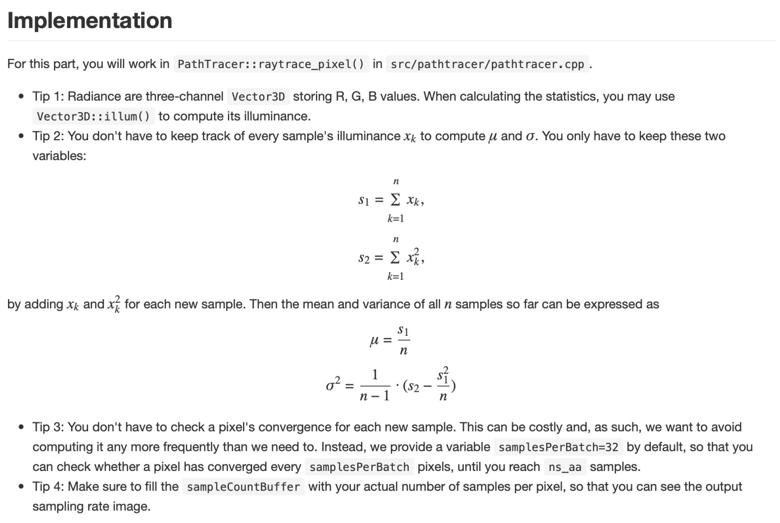
5 Task4

Indirect lighting takes into account the contribution of light bounces between non-emissive objects in the scene. Indirect lighting ray tracing find the space for all possible light paths from the light source reflect object.

Because of the time limit, I can't get all of these graphs, you can check the final gragh in Task 5.

6 Task5

The calculation here can be terminated directly when the light on a pixel reaches convergence. Implementing this sampling technique improves computational efficiency as it optimizes the amount of pixel super-sampling.



Implementation

For this part, you will work in `PathTracer::raytrace_pixel()` in `src/pathtracer/pathtracer.cpp` .

- Tip 1: Radiance are three-channel `Vector3D` storing R, G, B values. When calculating the statistics, you may use `Vector3D::illum()` to compute its illuminance.
- Tip 2: You don't have to keep track of every sample's illuminance x_k to compute μ and σ. You only have to keep these two variables:

$$s_1 = \sum_{k=1}^{n} x_k,$$

$$s_2 = \sum_{k=1}^{n} x_k^2,$$

by adding x_k and x_k^2 for each new sample. Then the mean and variance of all n samples so far can be expressed as

$$\mu = \frac{s_1}{n}$$

$$\sigma^2 = \frac{1}{n-1} \cdot (s_2 - \frac{s_1^2}{n})$$

- Tip 3: You don't have to check a pixel's convergence for each new sample. This can be costly and, as such, we want to avoid computing it any more frequently than we need to. Instead, we provide a variable `samplesPerBatch=32` by default, so that you can check whether a pixel has converged every `samplesPerBatch` pixels, until you reach `ns_aa` samples.
- Tip 4: Make sure to fill the `sampleCountBuffer` with your actual number of samples per pixel, so that you can see the output sampling rate image.

Figure 7: Adaptive Sampling Algo.

6.1 Extra

I use picture to explain how to implemented edge split operation with boundary face.

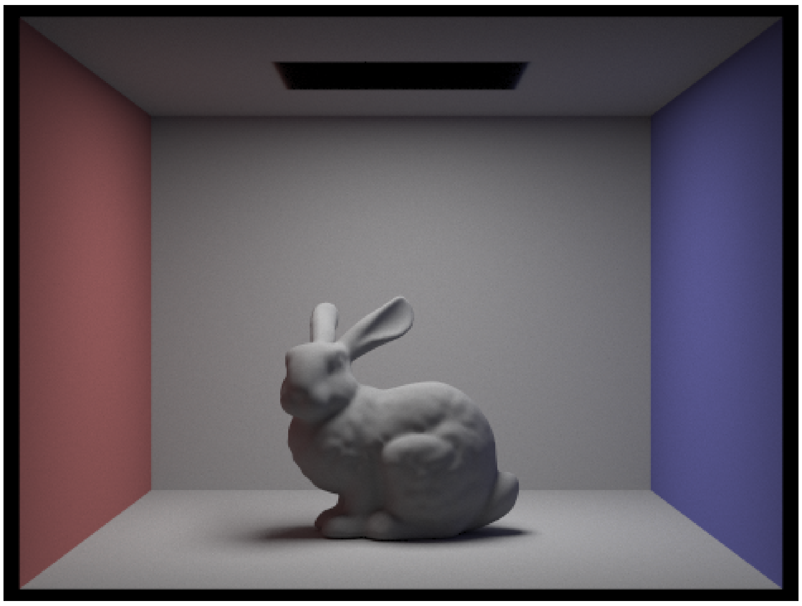


Figure 8: Bunny

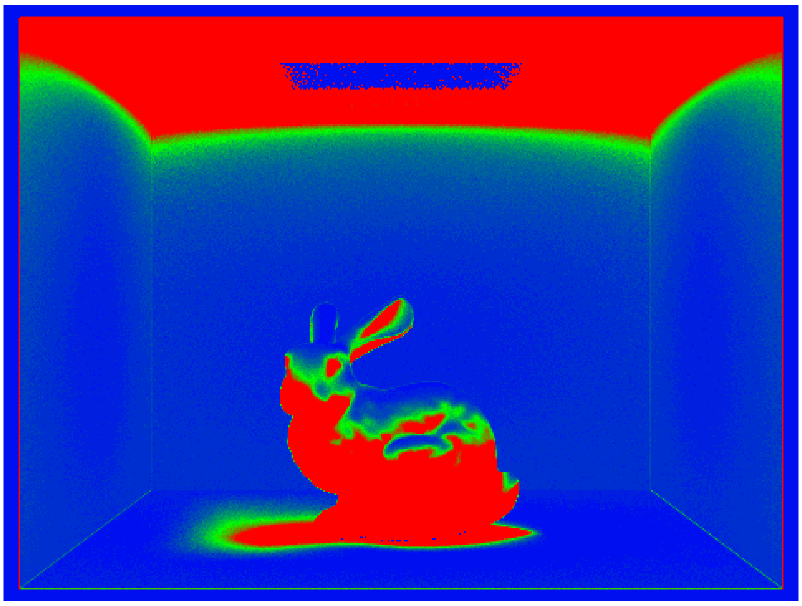


Figure 9: Bunny with Adaptive Sampling