CSC4140 Assignment 6

Computer Graphics

April 21, 2022

Ray Tracing

This assignment is 10% of the total mark.

Student ID: 120090266

Student Name: Feng Yutong

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

Signature: Feng Yutong

1 Overview

This project implement various methods of ray tracing, including generating rays, creating volum hierarchy (BVH) data structures, achieving direct and indirect illumination from multiple ray bounces to adaptive sampling. Rendering speed and effect have greatly improved through each step.

2 Implement and Results

2.1 Part 1: Ray Generation and Scene Intersection

Ray generation requires to build an image space and a camera space first. Their coordinates can easily be transformed with given formula. Then I use Samplers to sample some rays, once these rays intersect with objects, we render them by noramls. To check intersection with sphere, we can simply substitue the ray formula into sphere formula and compare the result t with the parameters t of ray (see Figure 1). To check intersection with triangle, I use Moller Trumbore Algorithm (see Figure 2), a faster approach based on barycentric coordinate. The basic idea is still substituding formula. If $0 < b_1 < 1, 0 < b_2 < 1, 0 < 1 - b_1 - b_2$, then the hit point is inside the triangle.

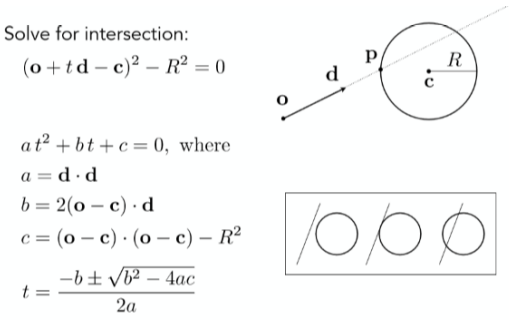


Figure 1: Intersection with sphere

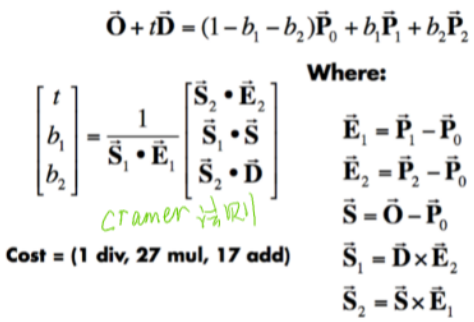
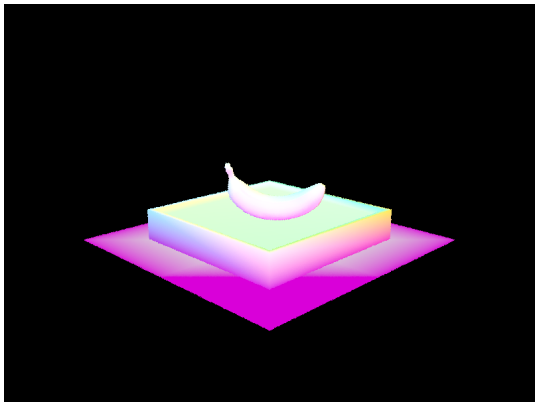
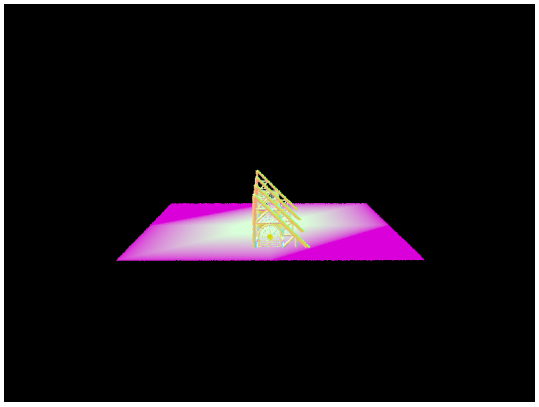


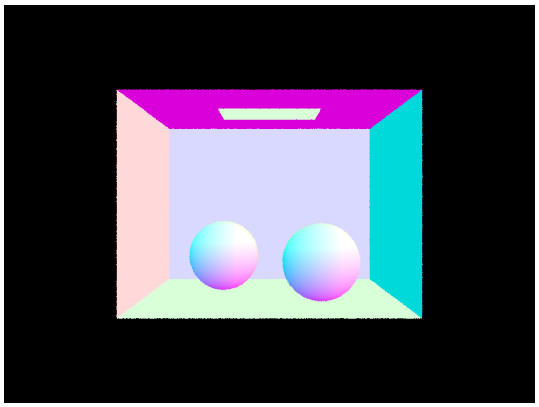
Figure 2: Moller Trumbore Algorithm



banana



building



CBsphere

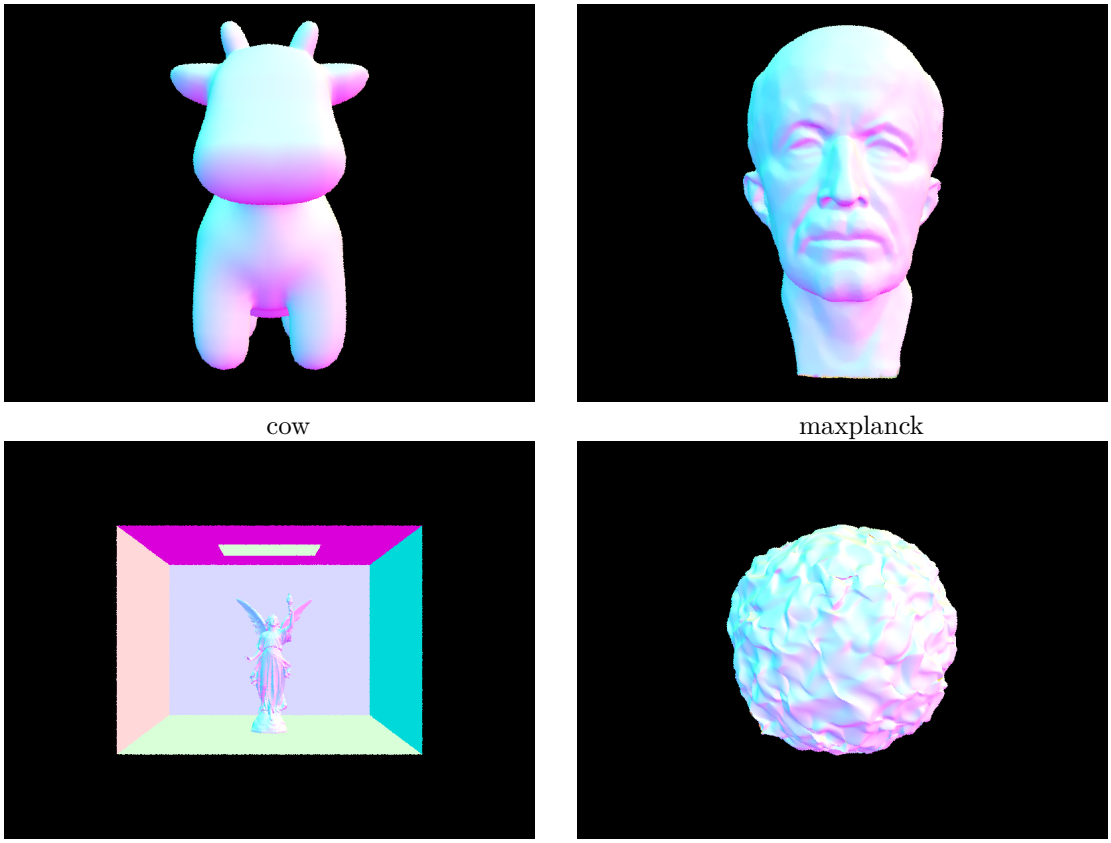
Figure 3: Normal sahding for small .dae files

2.2 Part 2: Bounding Volume Hierarchy

This part divides the orignial primitives into subsection. I build a BVH tree. For every node, I find the average centroid all of its primitives. Primitives with centroids smaller than average are stored in the left child while the rest are stored in the right child. The comparison is based on the longest axis. Repeat this process until the primitives of each child are less than max_leaf_size. I replace the start and end iterator with a vector prim for primitives are not listed in centroid order and it is hard to assign iterator for children. However, I use dynamic memory, only leaf children have prim storing primitives. So the space complex is O(n), the same level with given method.

Extra Point: I implement BVH using a stack instead of recursively calling the funciton. This may avoid potentially stack overflow and speed up program.

Figure 4 shows images with normal shading for a few large .dae files with BVH acceleraion. For cow.dae, rendering times with and without BVH acceleration are 0.4s and 36s. For maxplancky.dae, rendering times with and without BVH acceleration are 0.58s and 390s. With BVH acceleration, once the ray does not intersect with one node, we can just cut off the whole subtree (i.e., no intersection with a set means definitely no intersection with its subset), which can save a lot of useless intersection checking.



cow maxplanck

CBlucy blob

Figure 4: Normal sahding for large .dae files

2.3 Part 3: Direct illumination

Hemisphere sampling:

1. Sample wi around a hemisphere enclosing the hit point hit_p.

2. Convert wi to world space wi_w. Create a ray with origin hit_p and direciton wi_w. Set r.min_t = EPS_F and r.max_t = INF_D to alleviate numerical precision.

3. Find intersection by BVH.

4. Add the result of emsission multiplied by BSDF, cosine between wi and normal and (1/PDF) to total spectrum and calculate the average value.

Light sampling:

1. Loop through the scene light of a given hit point hit_p.

2. Convert wi to oject space w_in. Skip if w_in.z $<$ 0 for the sampled light point is behind surface.

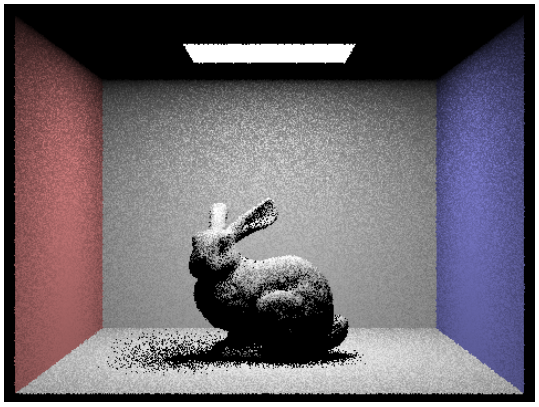
3. Get automatically calculated wi, PDF and distToLight when sampling light.

4. Create a ray with origin hit_p and direciton wi. Set r.min_t = EPS_F and r.max_t = distToLight to alleviate numerical precision.

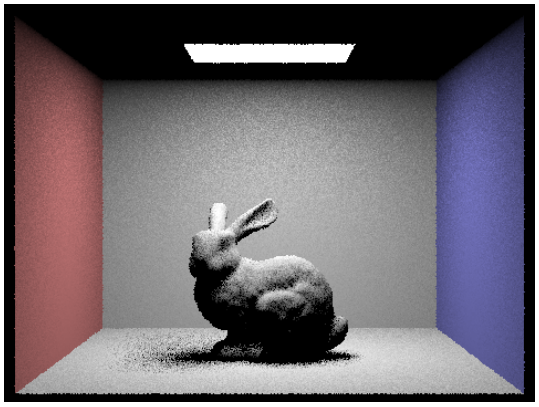
5. Find intersection by BVH.

4. If no surface is blocking, add the result of emsission multiplied by BSDF, cosine between wi and normal and (1/PDF) to total spectrum and calculate the average value.

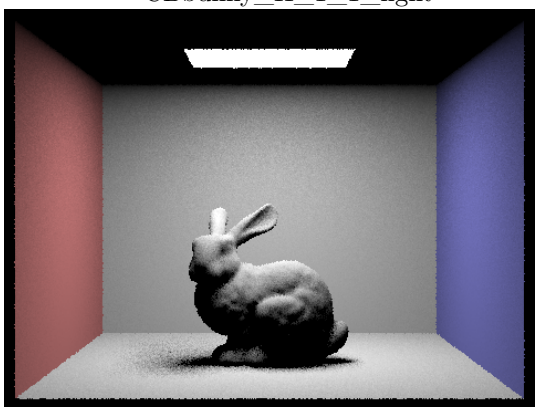
From Figures below, we can see light sampling results in less noise. Hemisphere sampling sample around the hemisphere close to the hit point, but the incoming radiance is zero for most directions. Thus, by sampling around the light within the solid angle to the hit light can reduce this noise, especially using fewer sample points.



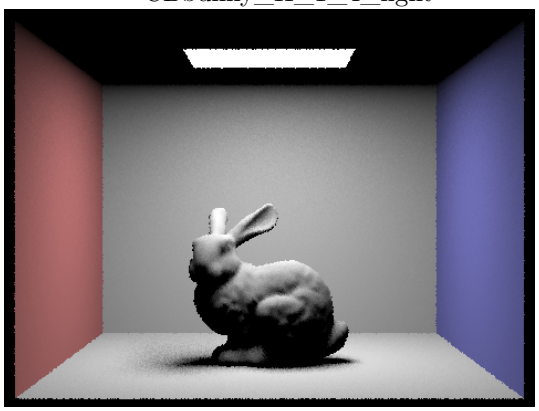
CBbunny_H_1_1_light



CBbunny_H_1_4_light

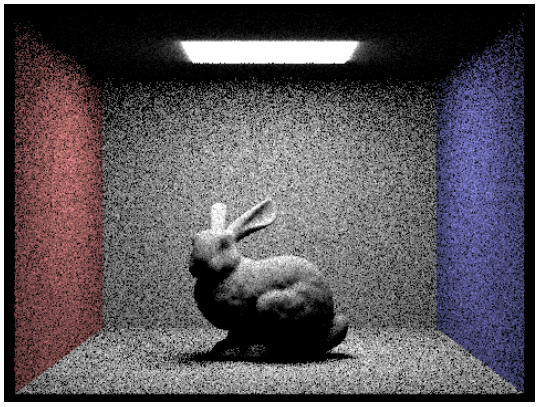


CBbunny_H_1_16_light

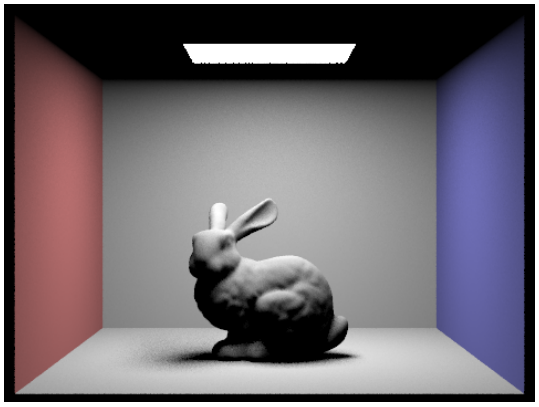


CBbunny_H_1_64_light

Figure 5: Rendering with 1, 4, 16, and 64 light rays
./pathtracer -t 8 -s 1 -l X -H -f CBbunny_H_1_X.png -r 480 360 ../dae/sky/CBbunny.dae



CBbunny_H_16_8_hemishpere



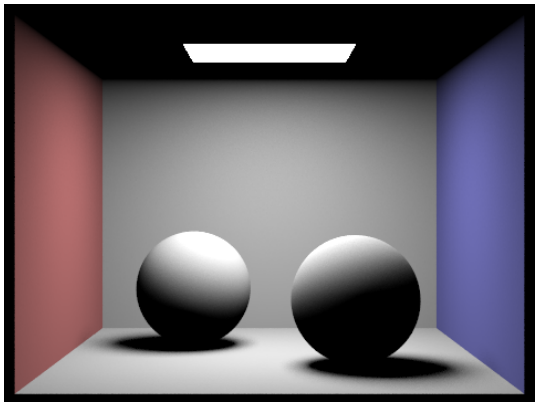
CBbunny_H_16_8_light

Figure 6: Comparsion between hemisphere sampling and light sampling
command:./pathtracer -t 8 -s 16 -l 8 -H -f CBbunny_H_16_8.png -r 480 360 ../dae/sky/CB-bunny.dae

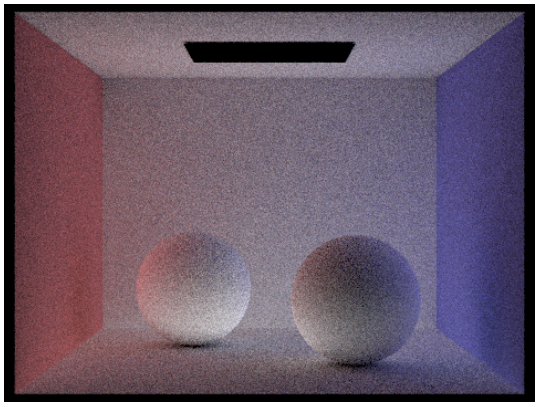
2.4 Part 4: Global illumination

Direct lighting includes zero_bounce_radiance and one_bounce_radiance. Indirect includes subsequent lighting bounce, i.e. at_least_one_bounce_lighting - one_bounce_radiance.

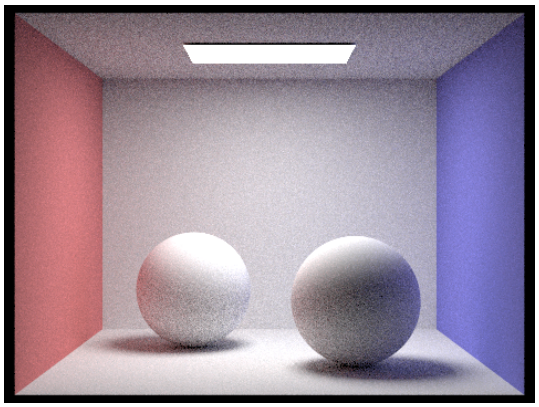
One_bounce_radiance can be either hemisphere sampling or light sampling. In at_least_one_bounce_lighting, I set the initial spectrum to be one_bounce_radiance, so it bounce at least once. Then I recursively call itself and decrease ray depth by one if ray depth is greater than one of if a Russian Roulette probalility(cpdf) is satisfied. One thing to note is that when computing average spectrum, cpdf(0.6) is divided.



direct illumination

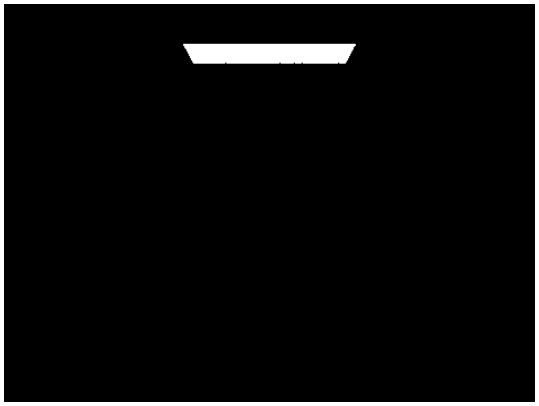


indirect illumination

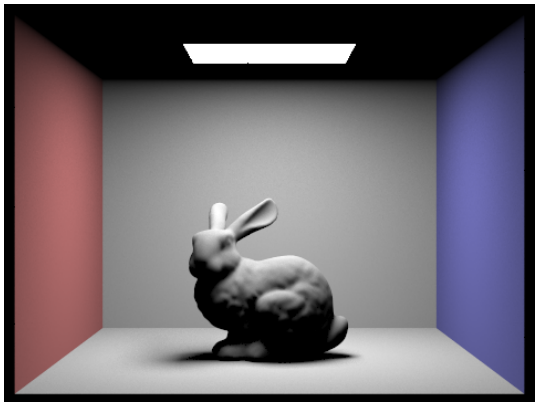


global illumination

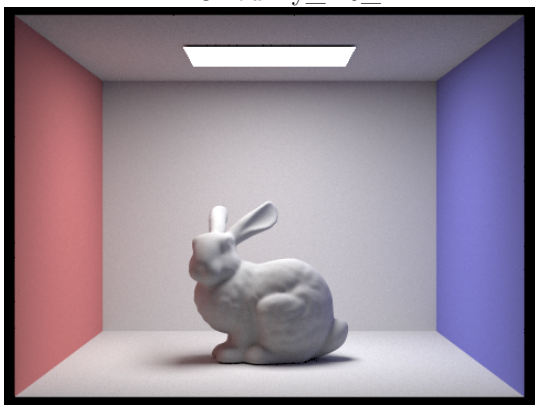
Figure 7: Comparsion between different illumination
command:./pathtracer -t 8 -s 64 -l 16 -m 5 -r 480 360 -f X.png ../dae/sky/CB-spheres_lambertian.dae



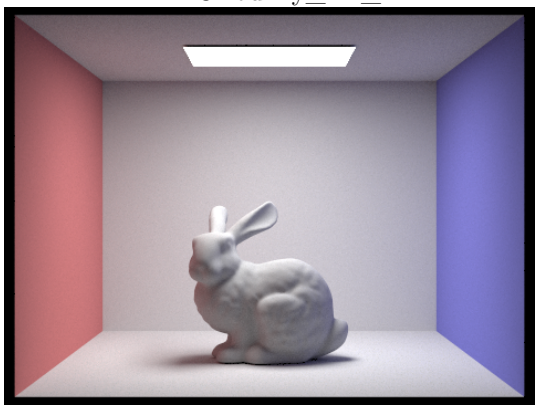
CBbunny_m0_



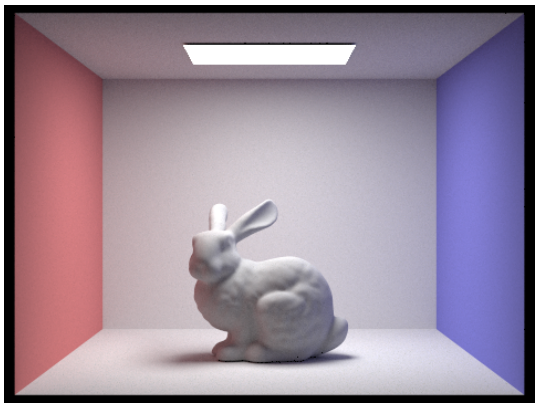
CBbunny_m1_



CBbunny_m2_

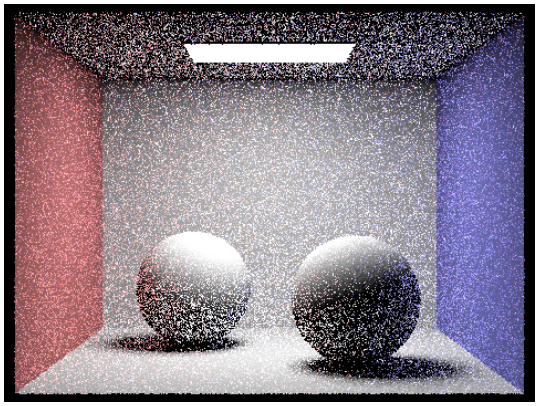


CBbunny_m3_

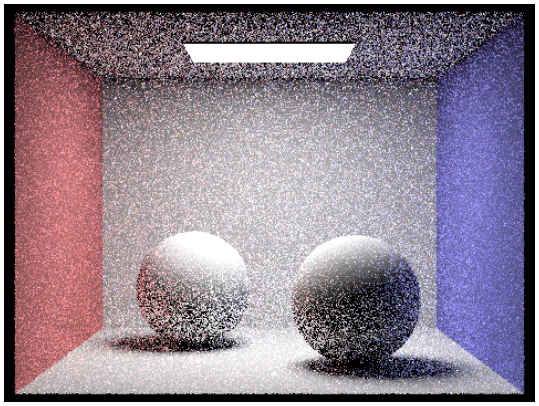


CBbunny_m100_

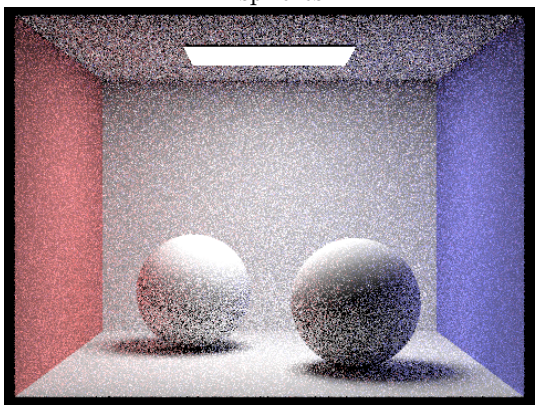
Figure 8: Comparsion between differenet max_ray_depth
command:./pathtracer -t 8 -s 16 -l 8 -H -f -m X CBbunny_mX_.png -r 480 360 ../dae/sky/CB-bunny.dae



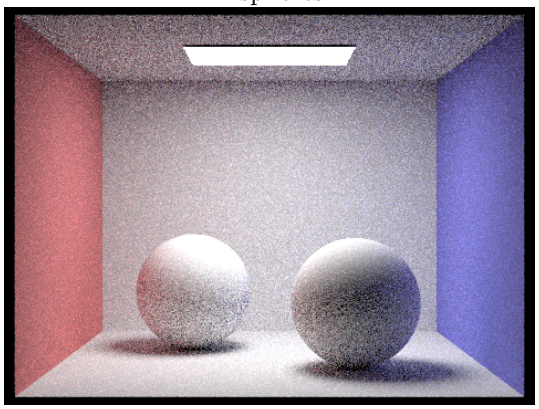
spheres1



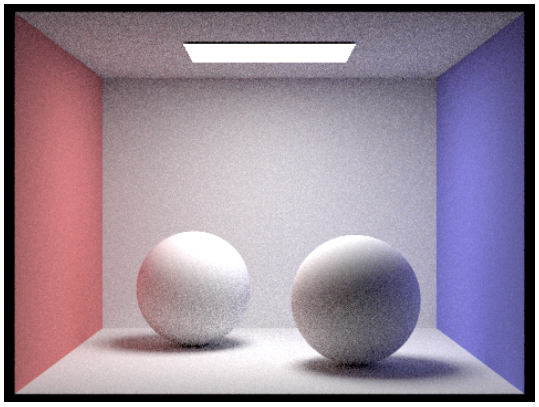
spheres2



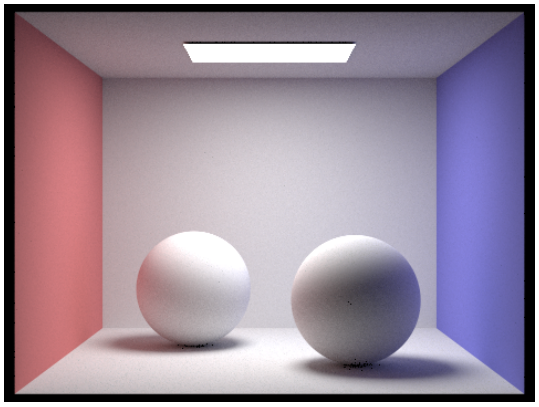
spheres4



spheres16



spheres64

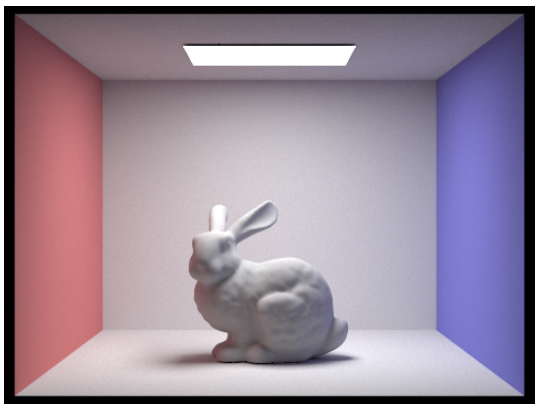


spheres1024

Figure 9: Comparsion between differenet sample per pixel
command:./pathtracer -t 8 -s X -l 4 -m 5 -r 480 360 -f spheresX.png ../dae/sky/CB-spheres_lambertian.dae

2.5 Part 5: Adaptive Sampling

To check the convergence, we simply check if $I \leq maxTolerance \cdot \mu$ where maxTolerance=0.05, $I = 1.96 \cdot \frac{\sigma}{\sqrt{n}}$, $\sigma$is standard deviation and μ is mean. To save time, we just check when n%samplesPerPatch == 0. If condition is met, stop loop and update actual sample number.



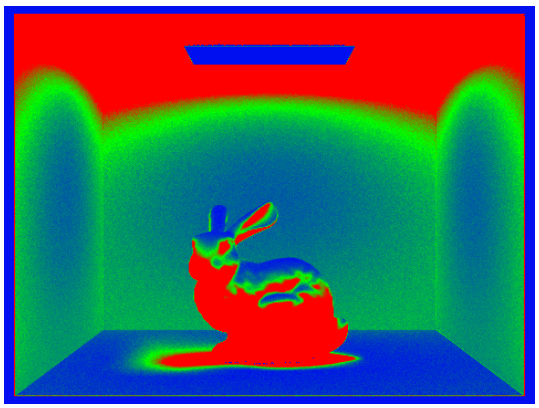


Figure 10: Adpative sampling
command:./pathtracer -t 8 -s 2048 -a 64 0.05 -l 1 -m 5 -r 480 360 -f bunny.png ../dae/sky/CB-bunny.dae