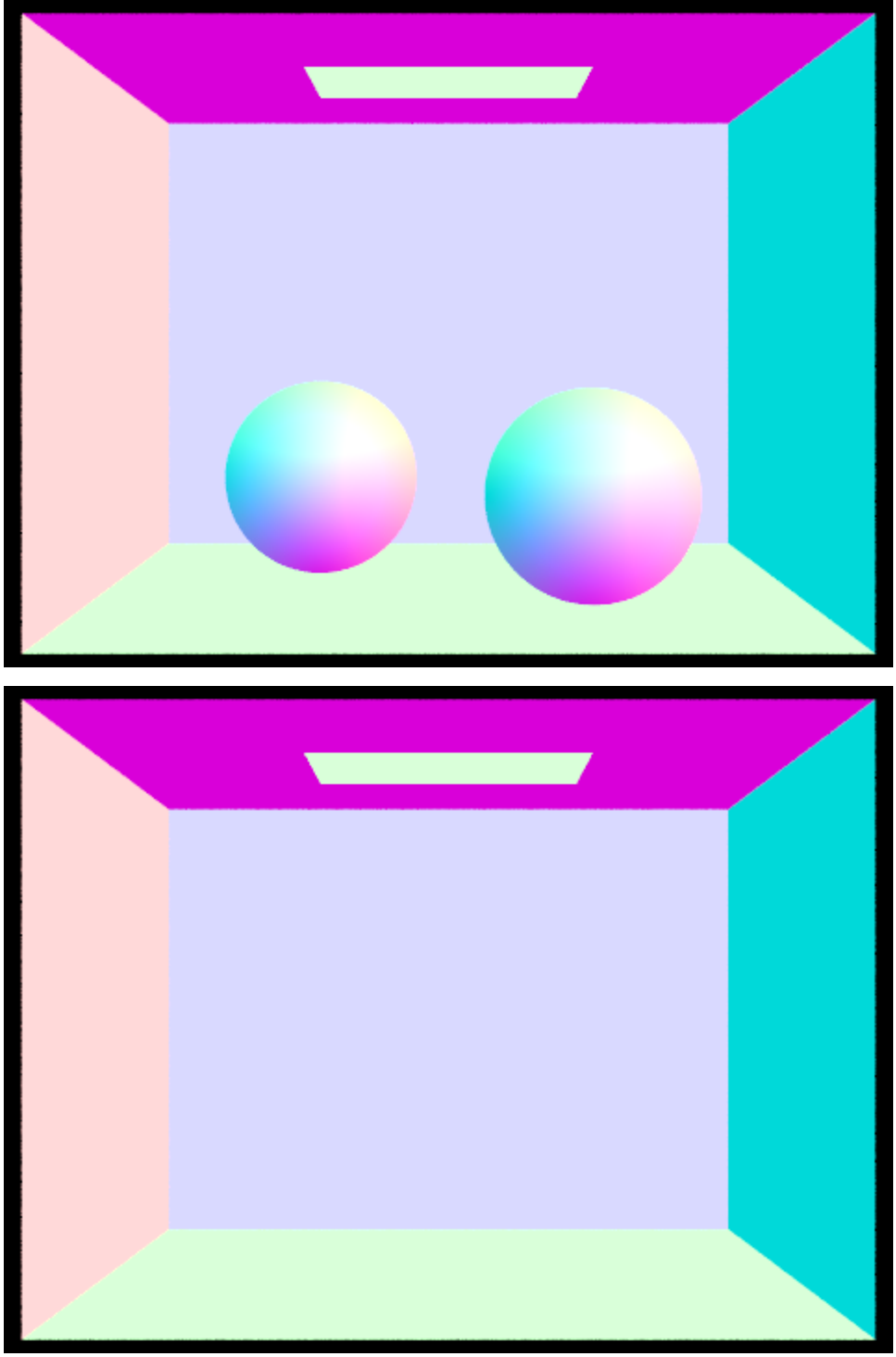
Part 1

- Explain the triangle intersection algorithm you implemented in your own words.
  To calculate the triangle intersecion, we need to first calculate the intersection point of a ray and the plane that the triangle layed on. In this question, we can use a arbitrary point of triangle vertex and the face normal to represent the plane.
  Then, getting the intersection point, we need to do the interior test. Many methods can accomplish this task, and I chose the easiest one. By calculating the norm of the cross product of each vertex, we can both get the barycentric coordinate and the intersection result.

- Show images with normal shading for a few small $.dae$ files.



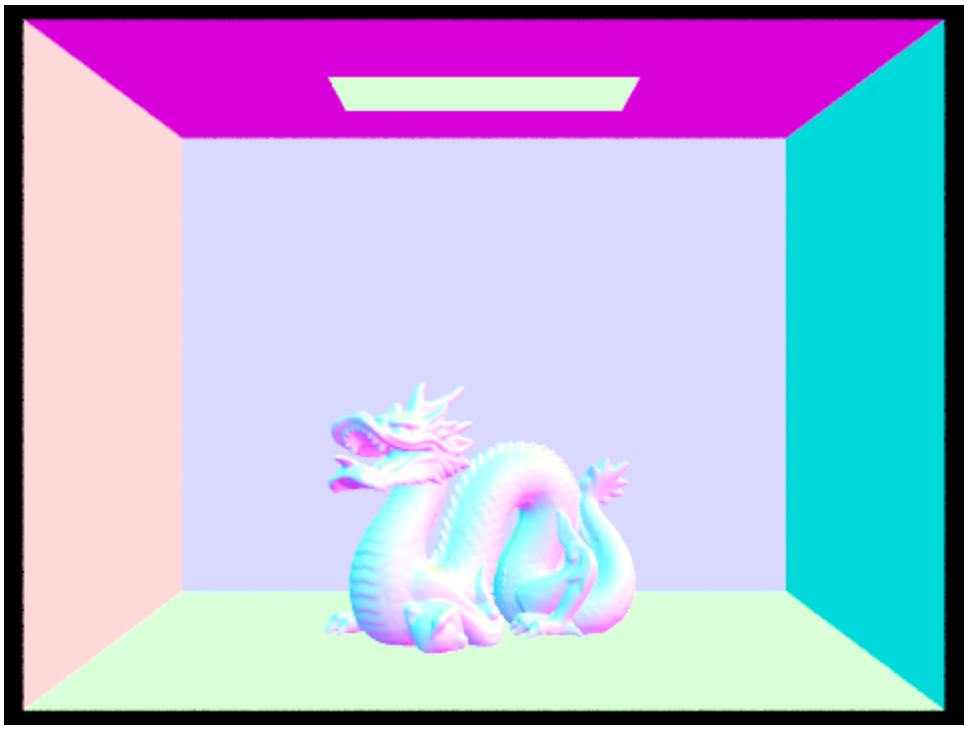
Part 2

- Walk through your BVH construction algorithm. Explain the heuristic you chose for picking the splitting point. The most fundamental BVH construction will require a partition, where we can use `std::partition` from the

`<algorithm>` library to easily partition the given primitives. A point that will be ignored is to make sure that every node in the tree will only have zero or two children node. One child node will slow down the BVH searching algorithm.

I tried SAH. Although it is not stable, when it works, the performance do increase a lot.

- Show images with normal shading for a few large *.dae* files that you can only render with BVH acceleration.



- Compare rendering times on a few scenes with moderately complex geometries with and without BVH acceleration. Present your results in a one-paragraph analysis.

```
sydian@DESKTOP-EGJQLM3:~/CSC4140-ComputerGraphics/Assignment5/RayTracing1/build$ ./pathtracer -t 32 -s 32 -l 6 -m 5 -r 480 360 -f coil_normal.png ../dae/sky/CBspheres.dae
[PathTracer] Input scene file: ../dae/sky/CBspheres.dae
[PathTracer] Rendering using 32 threads
[PathTracer] Collecting primitives... Done! (0.0000 sec)
[PathTracer] Building BVH from 14 primitives... Done! (0.0000 sec)
[PathTracer] Rendering... 100%! (1.1639s)
[PathTracer] BVH traced 3832991 rays.
[PathTracer] Average speed 3.2933 million rays per second.
[PathTracer] Averaged 2.510251 intersection tests per ray.
[PathTracer] Saving to file: coil_normal.png... Done!
[PathTracer] Job completed.
```

```
sydian@DESKTOP-EGJQLM3:~/CSC4140-ComputerGraphics/Assignment5/RayTracing1/build$ ./pathtracer -t 32 -s 32 -l 6 -m 5 -r 480 360 -f coil_normal.png ../dae/sky/CBspheres.dae
[PathTracer] Input scene file: ../dae/sky/CBspheres.dae
[PathTracer] Rendering using 32 threads
[PathTracer] Collecting primitives... Done! (0.0000 sec)
[PathTracer] Building BVH from 14 primitives... Done! (0.0000 sec)
[PathTracer] Rendering... 100%! (0.8793s)
[PathTracer] BVH traced 3526118 rays.
[PathTracer] Average speed 4.0101 million rays per second.
[PathTracer] Averaged 2.177953 intersection tests per ray.
[PathTracer] Saving to file: coil_normal.png... Done!
[PathTracer] Job completed.
```

Part 3

- Walk through both implementations of the direct lighting function.
  Two methods are implemented for direct lighting.
  Random hemisphere and importance sampling.

For random hemisphere, we simply random sampled numerous points at the hemisphere origined at the hit point. Then we will sum them together to represent the incoming radiance L_{in}. Further, we will fill the bsdf for diffuse material. For diffuse material, the outgoing direction will distribute uniformly on the hemisphere, thus we only need to randomly return a direction on the hemisphere in the local space. Then, we will calculate the cosine term using w_{in}, n. And bsdf will be a color value as a constant. The resulting radiance can be calculated then.

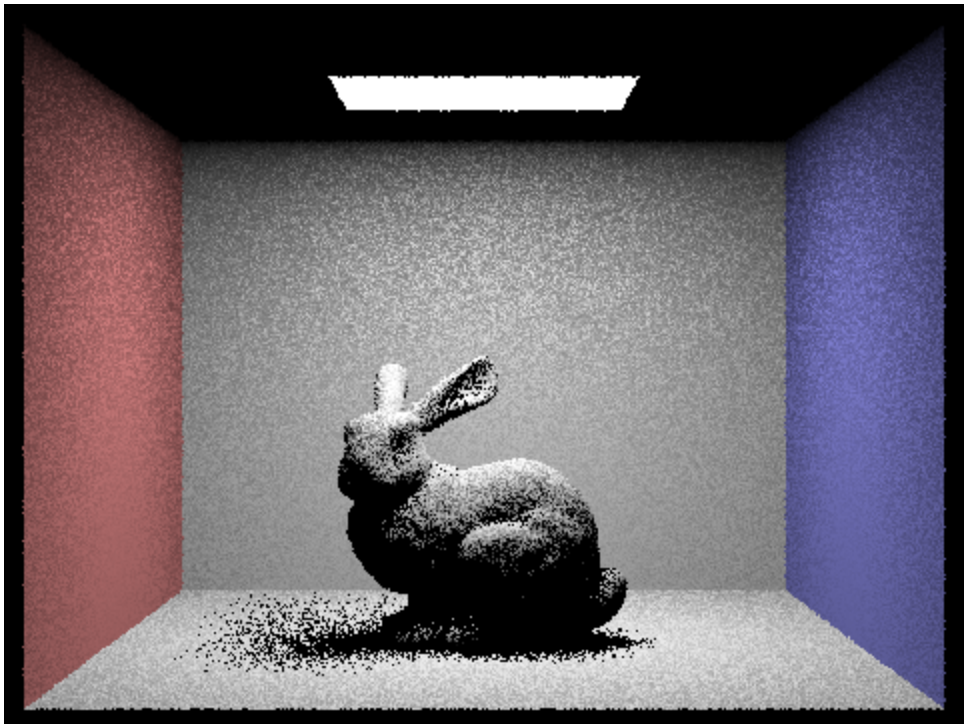
For importance sampling, all the shading process are almost the same, while we need to explicitly pick out one sample

direction based on the given light direction and find out if occulusion occurs.

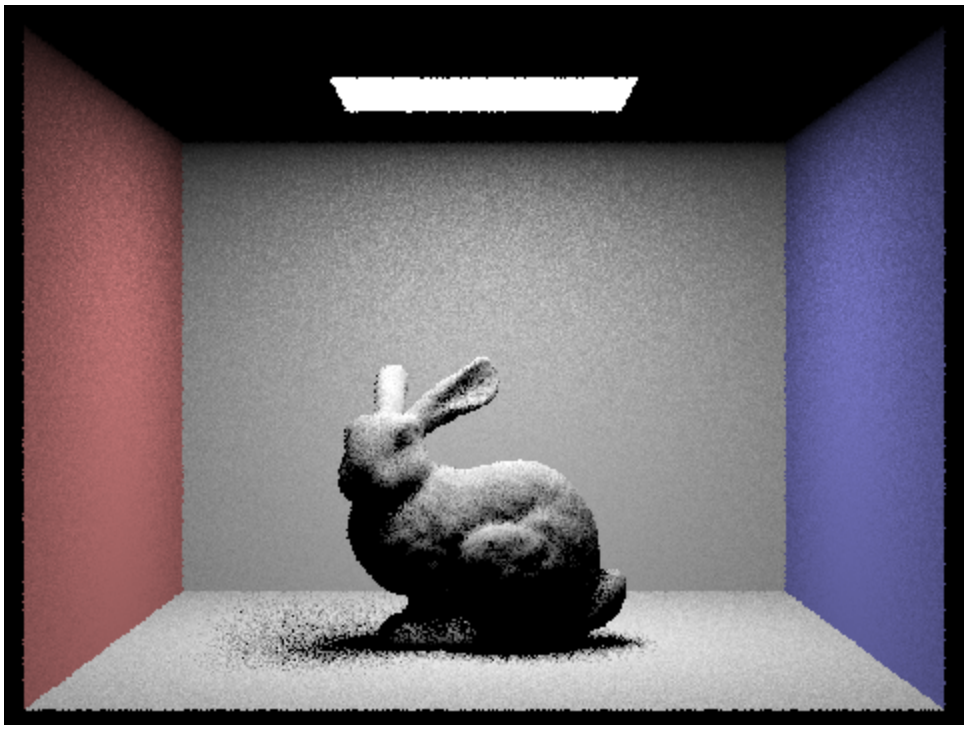
- Show some images rendered with both implementations of the direct lighting function.

- Focus on one particular scene with at least one area light and compare the noise levels in soft shadows when rendering with 1, 4, 16, and 64 light rays (the `-l` flag) and with 1 sample per pixel (the `-s` flag) using light sampling, **not** uniform hemisphere sampling.

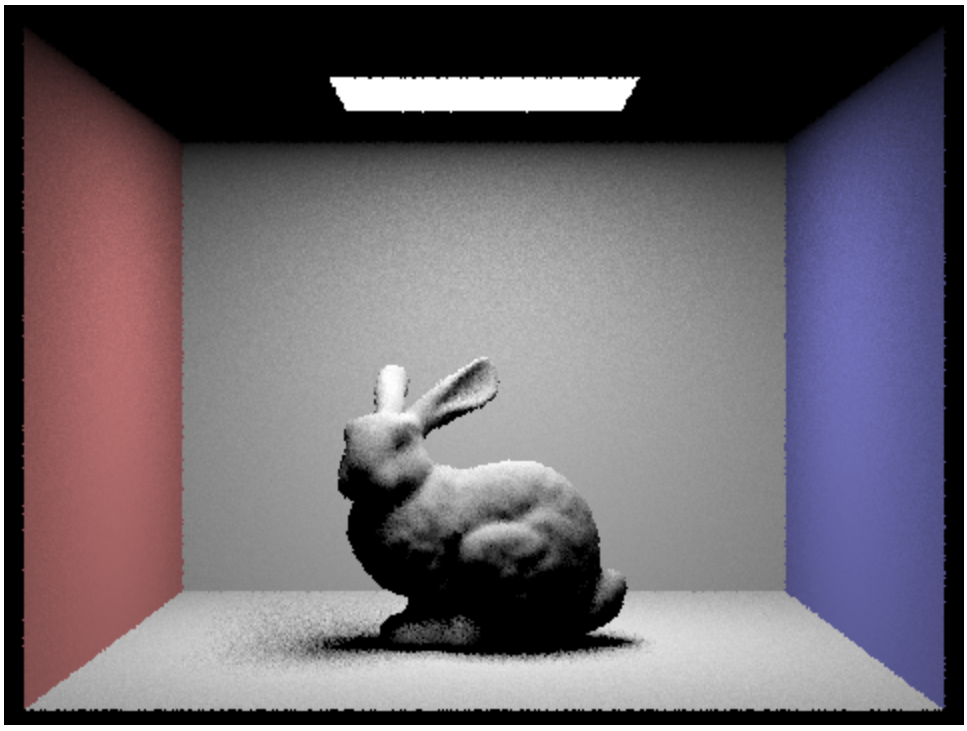
- 1



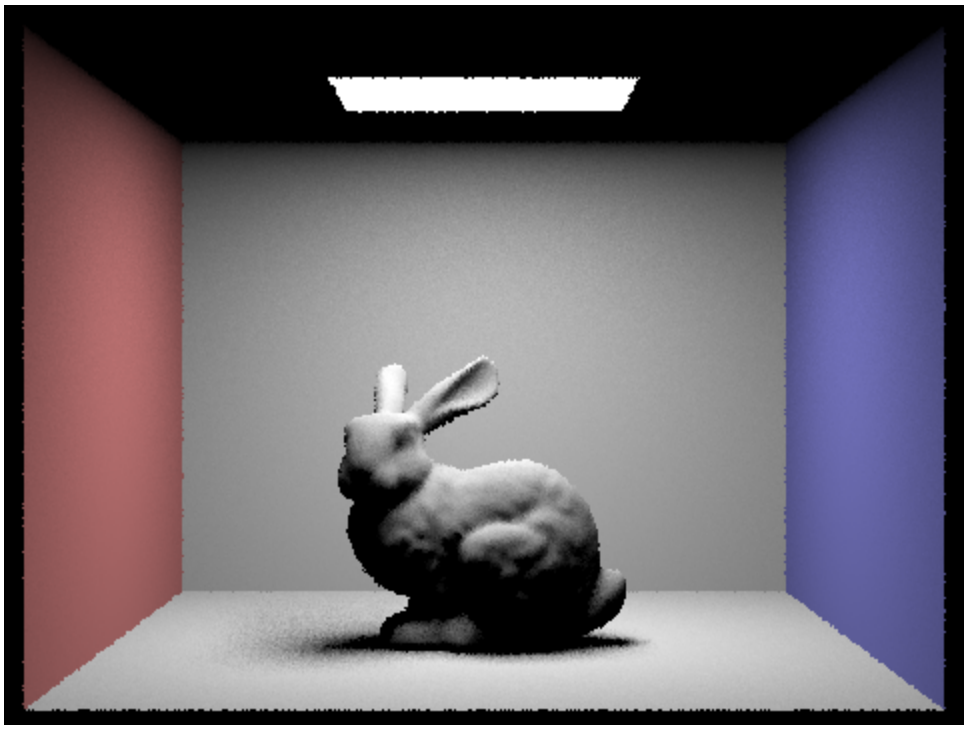
- 4



- 16



- 64



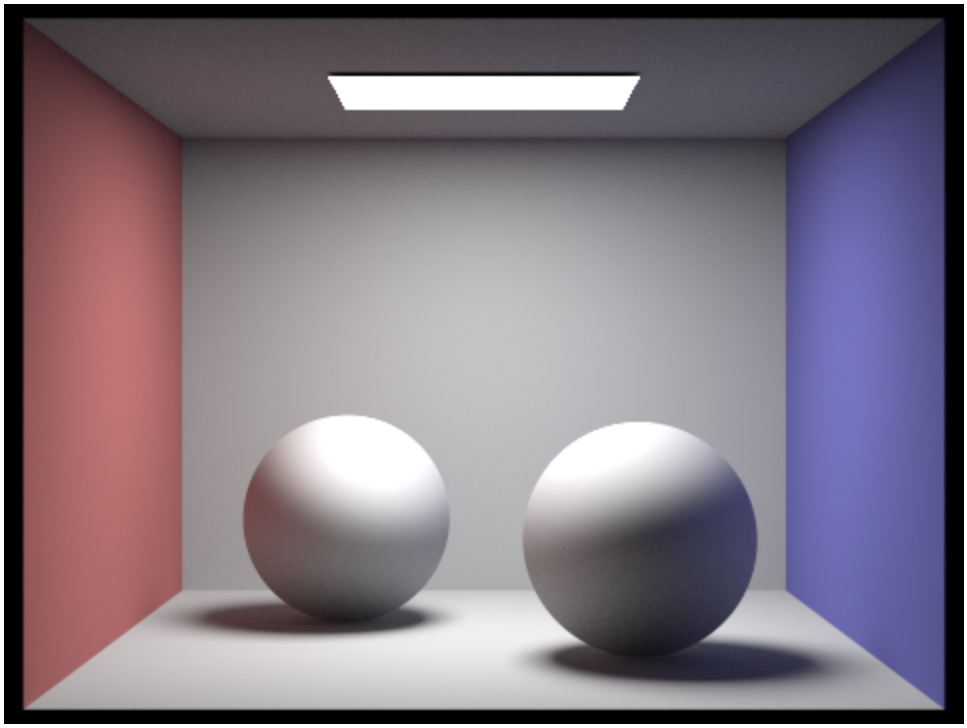
- Compare the results between uniform hemisphere sampling and lighting sampling in a one-paragraph analysis.

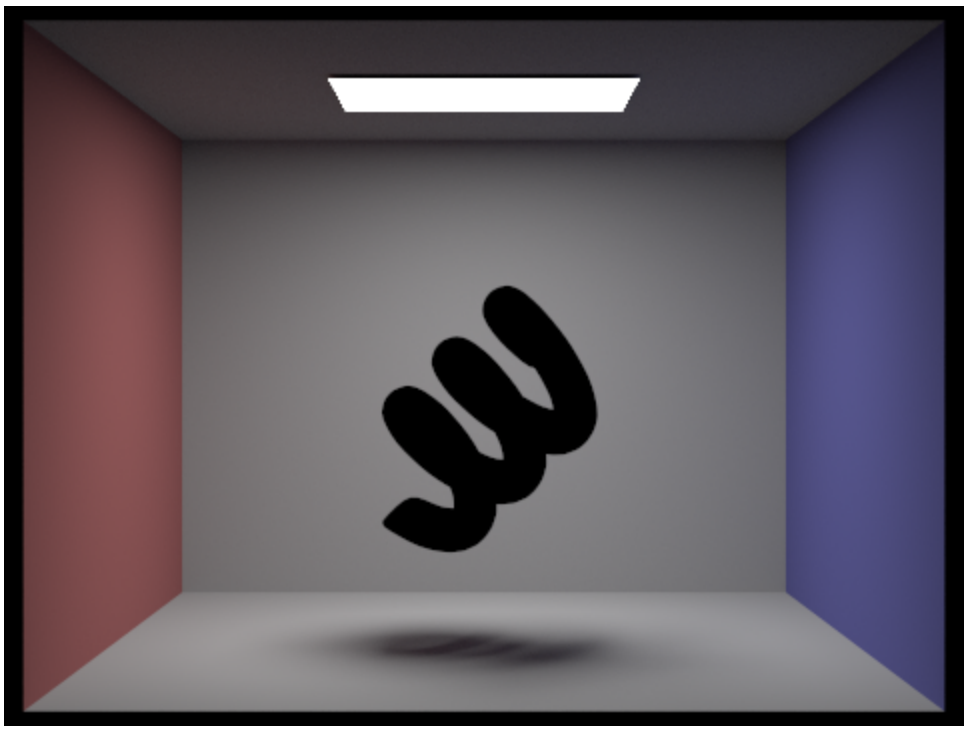
Part 4

- Walk through your implementation of the indirect lighting function.
  Indirect lighting consist of multiple bounces. In each bounce, the calculation of radiance is the same as the part 3. We will skip this part and only focus on the termination process. For termination, we will use both `ray_depth` and `coin_flip` method. It should be noted that using `coin_flip` will also add a `p` term to the

denominator as `coin_flip` itself is a variation of Monte-carlo method.

- Show some images rendered with global (direct and indirect) illumination. Use 1024 samples per pixel.



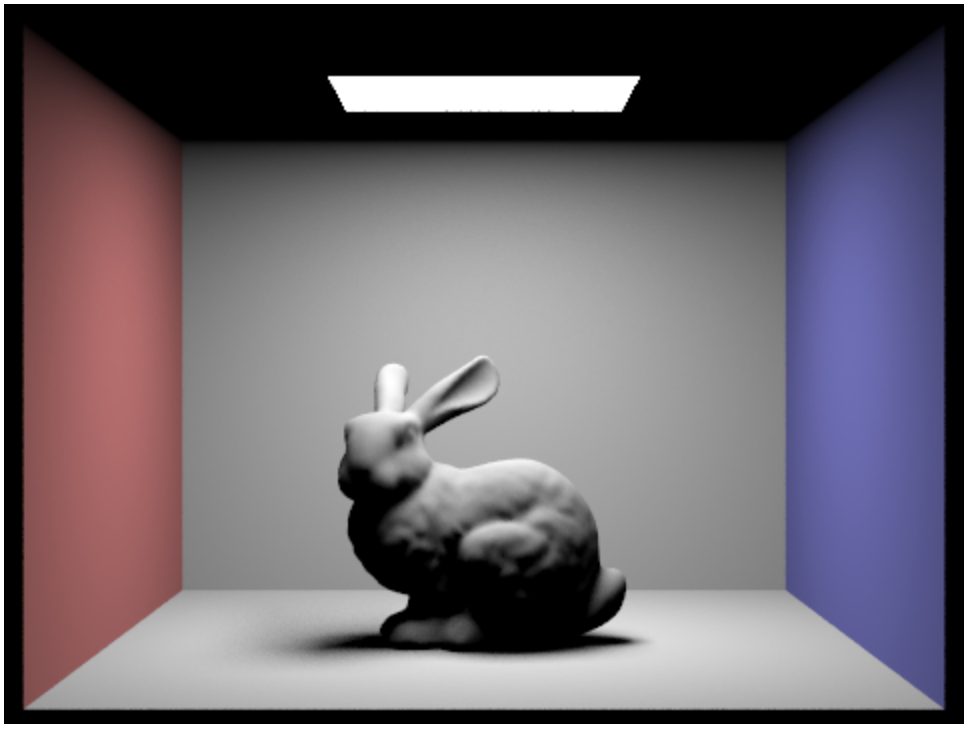


- Pick one scene and compare rendered views first with **only** direct illumination, then **only** indirect

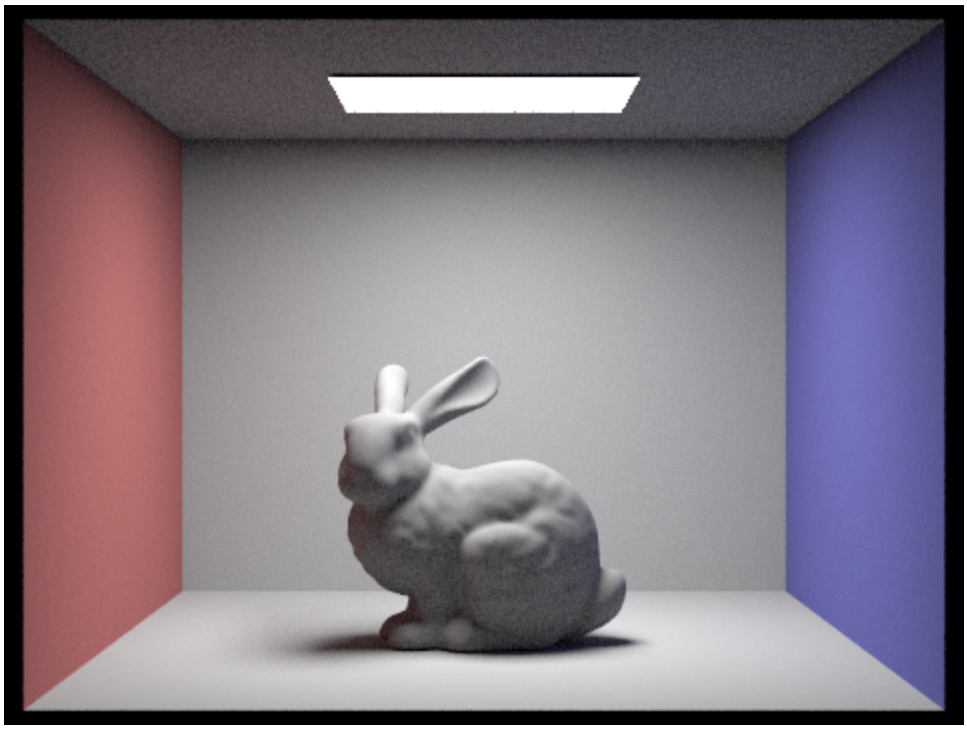
illumination. Use 1024 samples per pixel. (You will have to edit `PathTracer::at_least_one_bounce_radiance(...)` in your code to generate these views.)

- For *CBbunny.dae*, compare rendered views with `max_ray_depth` set to 0, 1, 2, 3, and 100 (the `-m` flag). Use 1024 samples per pixel.
- 0

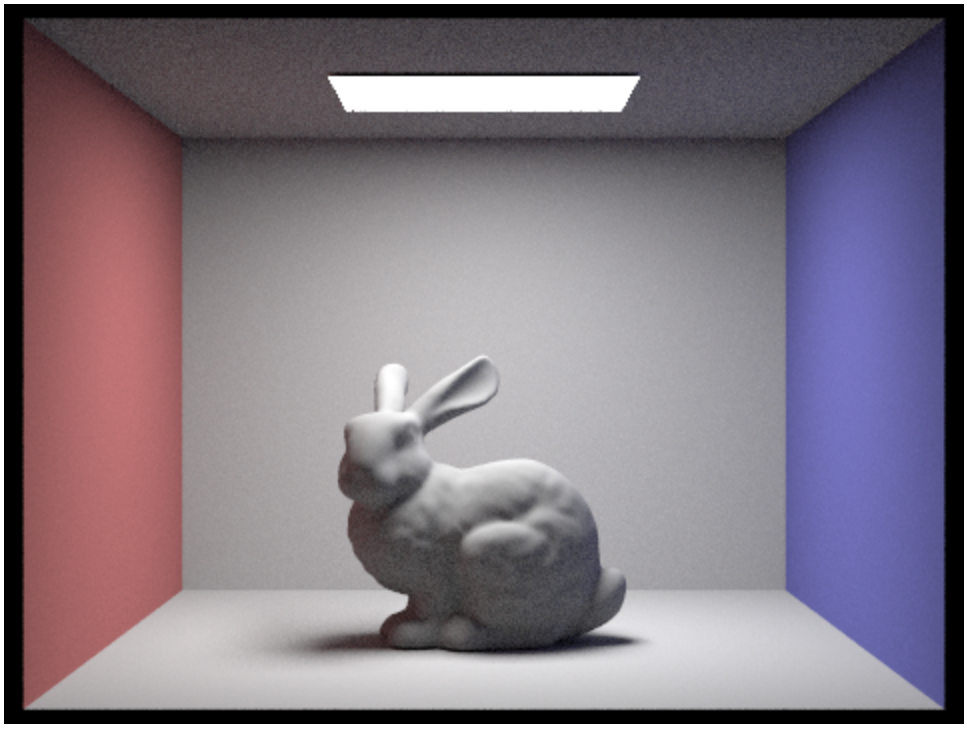
- 1



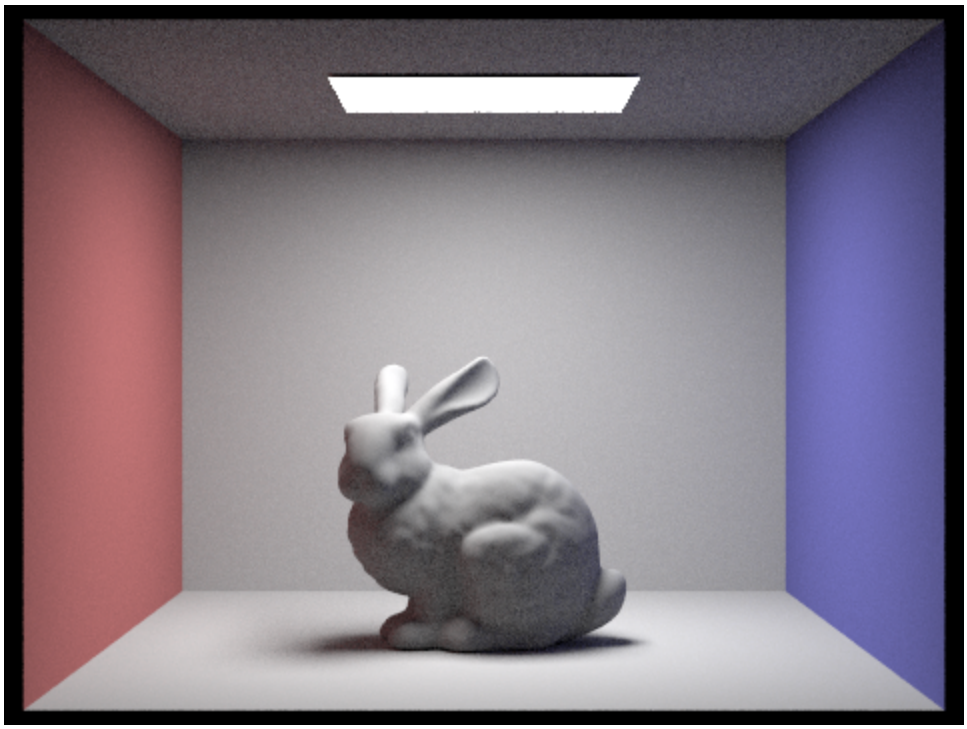
- 2



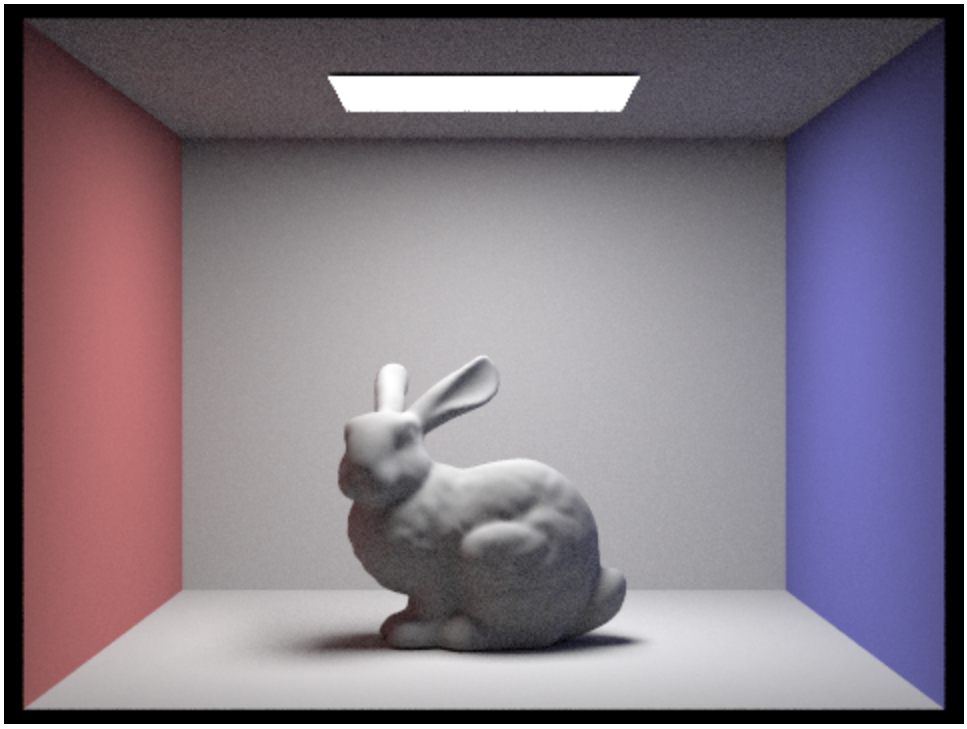
- 3



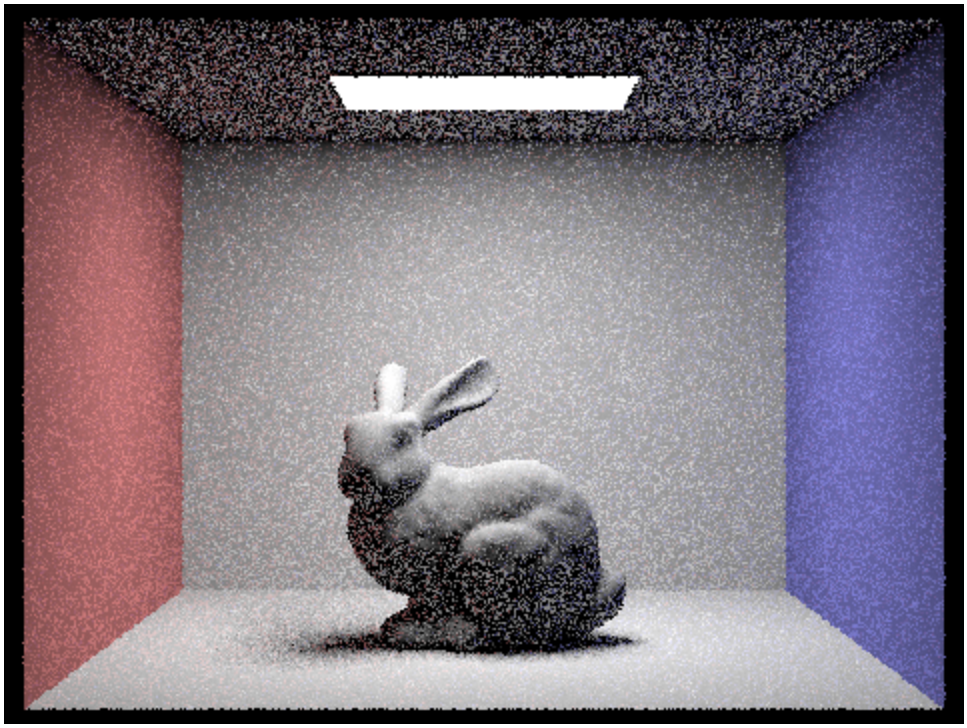
- 4



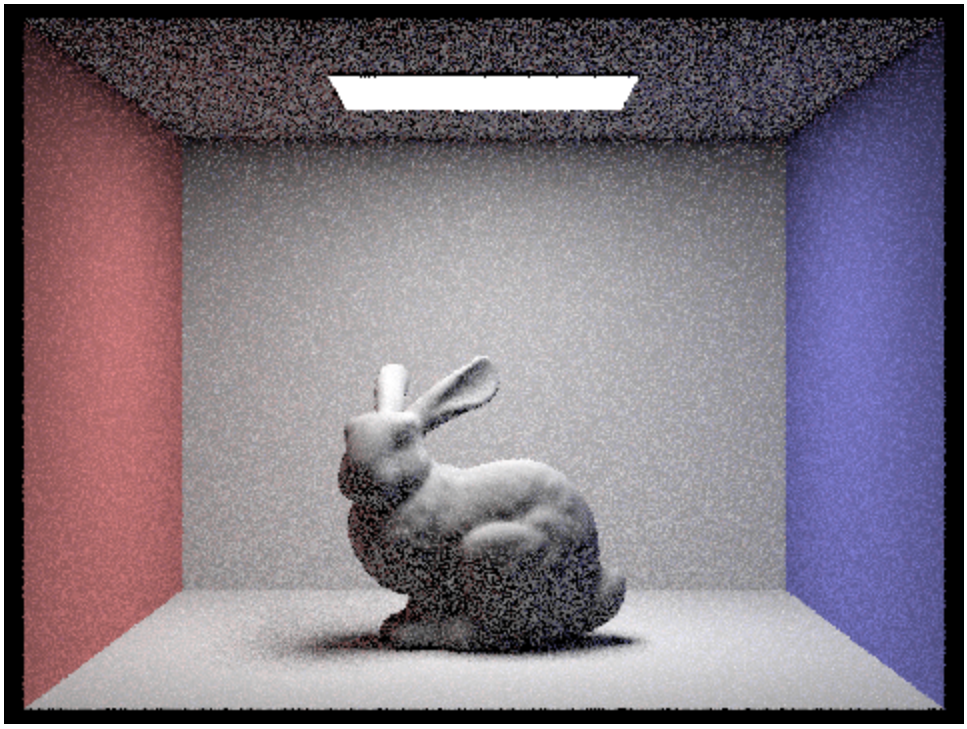
- 100



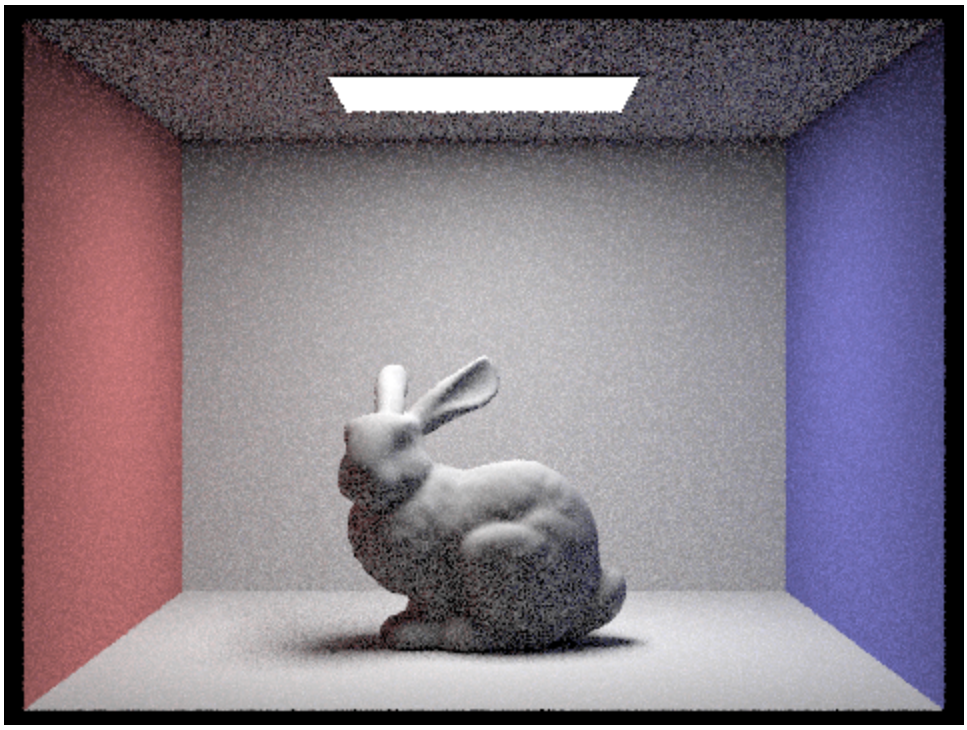
- Pick one scene and compare rendered views with various sample-per-pixel rates, including at least 1, 2, 4, 8, 16, 64, and 1024. Use 4 light rays.
- 1



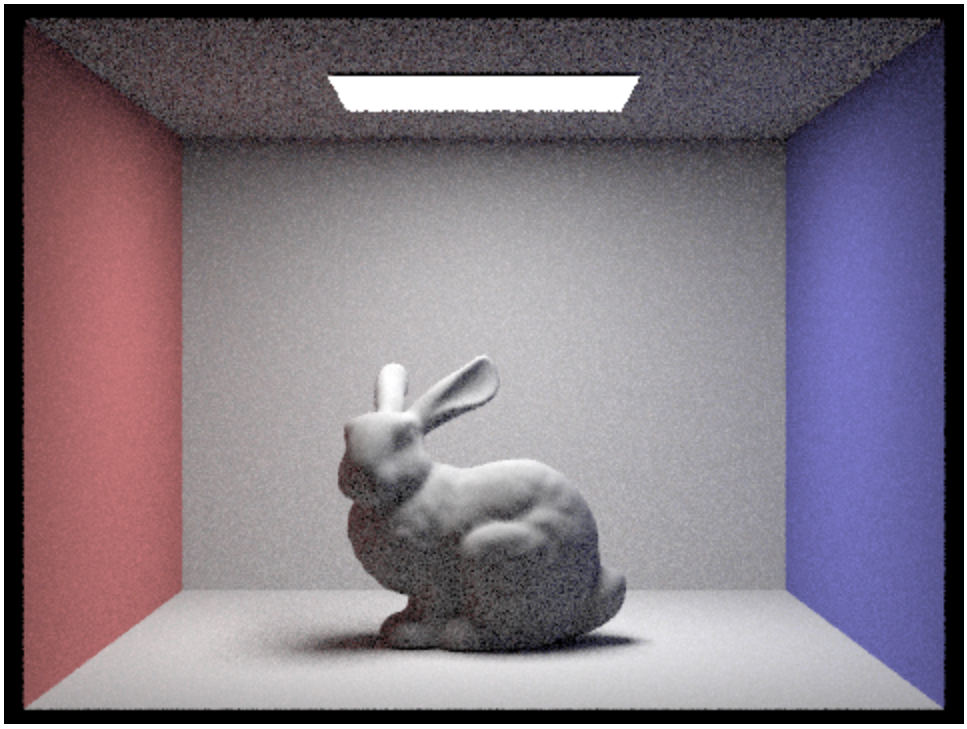
- 2



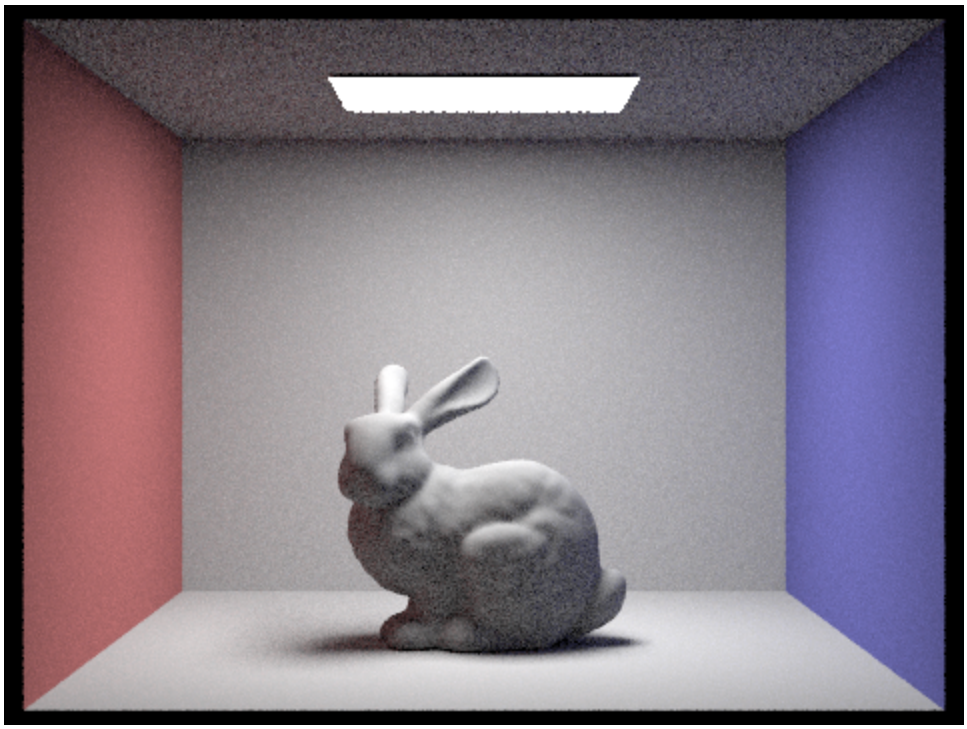
- 4



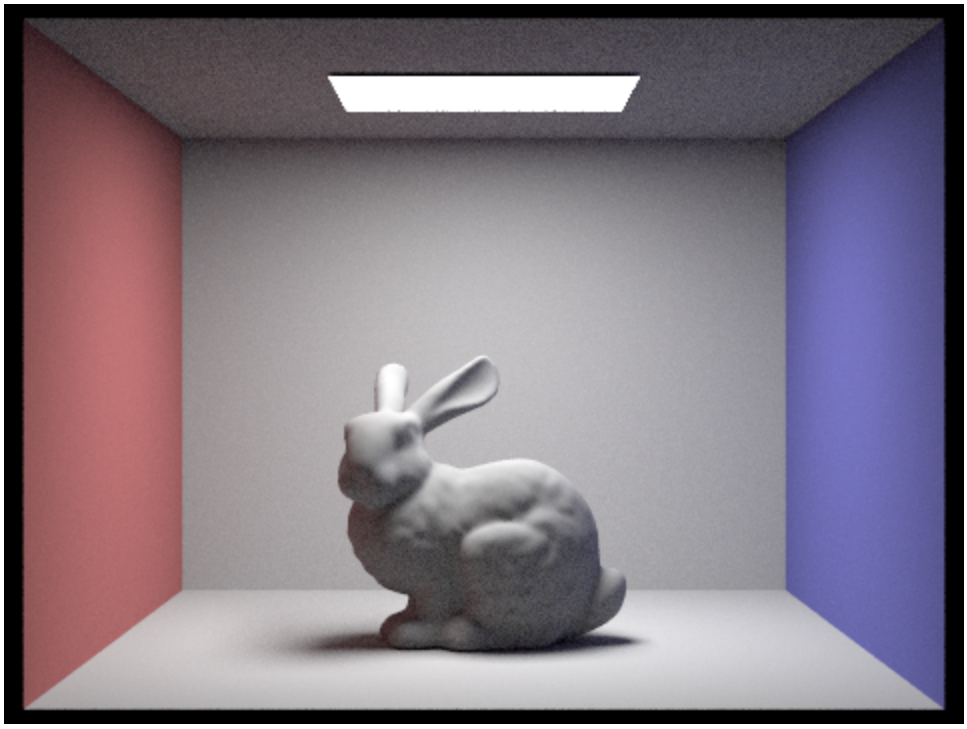
- 8



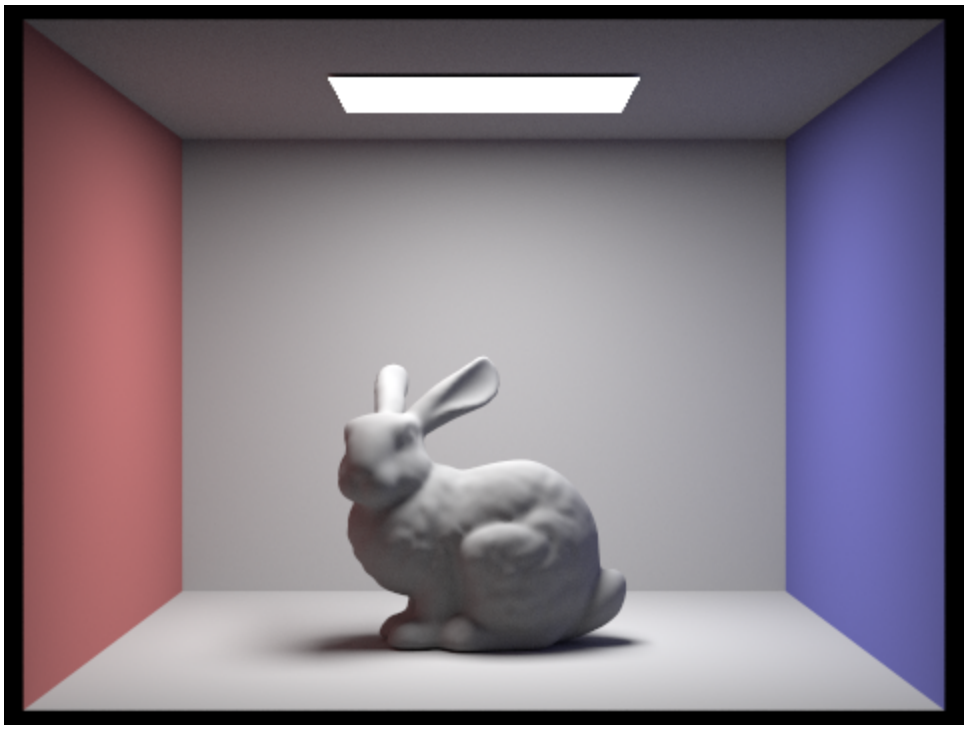
- 16



- 64



- 1024



Part 5

- Walk through your implementation of the adaptive sampling.

In adaptive sampling, we need to do supersampling within each pixel and calculate the statistics of their radiance. If the statitics satisfy a condition, we will terminate the supersampling. If not(which implies this pixel belongs to an edge), we will need higher sampling rate.

- Pick one scene and render it with at least 2048 samples per pixel. Show a good sampling rate image with clearly visible differences in sampling rate over various regions and pixels. Include both your sample rate image, which shows your how your adaptive sampling changes depending on which part of the image you are rendering, and your noise-free rendered result. Use 1 sample per light and at least 5 for max ray depth.!