CS 184: Computer Graphics and Imaging, Spring 2019

Project 3-1: Pathtracer

George Zhang, CS184-georgebzhang

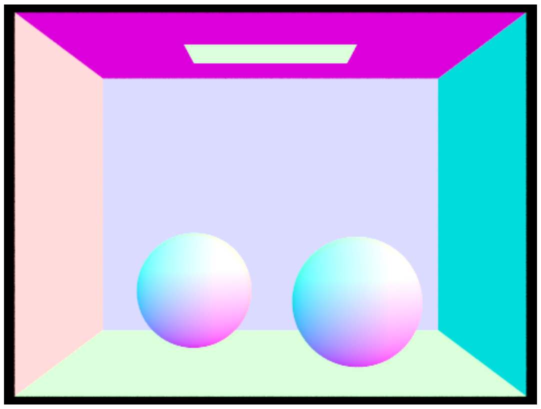
Overview

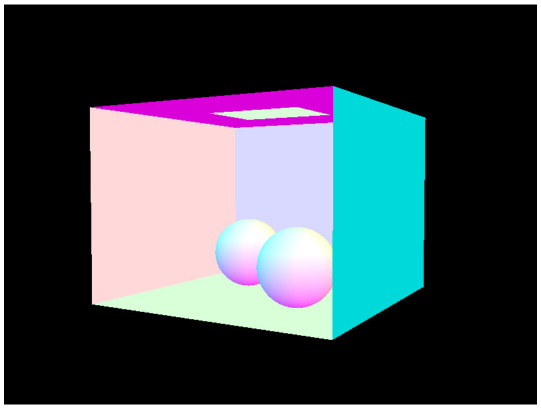
In this project, I explored how ray tracing can be implemented, from generating rays to checking intersections to creating bonding volume hierarchy (BVH) data structures to direct and indirect illumination from multiple ray bounces to adaptive sampling methods to check if samples have converged early to speed up render times.

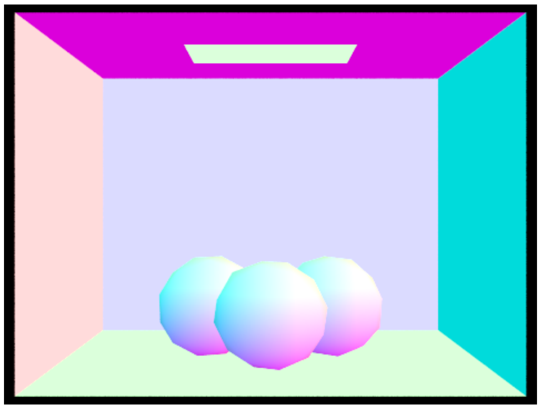
Part 1: Ray Generation and Scene Intersection

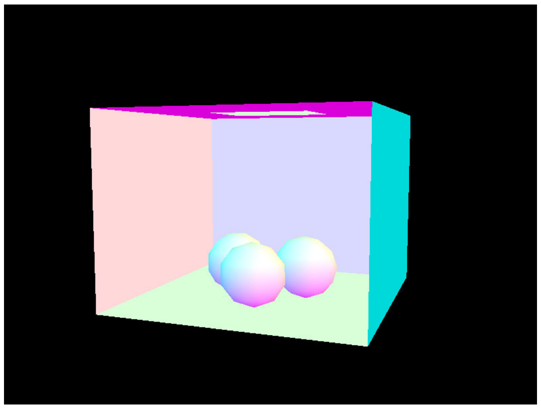
I used the Moller Trumbore algorithm using Cramer's rule, as described in http://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection. This involves setting the ray path expression equal to the plane expression. We must make sure that the Barycentric coordinates w, u, v sum to one. If t_min < t < t_max, then t is valid.

I used the settings "-t 8 -s 64 -l 16 -m 6 -r 480 360 -f" on hive25 machine.


Spheres (4 sec).


Spheres (rotated).


Gems (30 sec).


Gems (rotated).

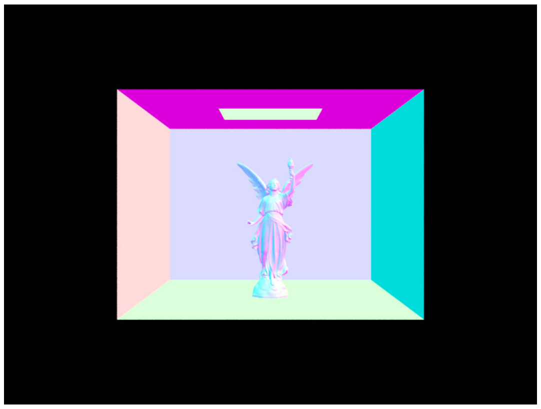
Part 2: Bounding Volume Hierarchy

I found the average centroid of all primitives. All primitives with centroids greater than average were stored in one child. The rest were stored in the other child. This process was repeated until each child had less than max_leaf_size primitives. I actually progammed this iteratively (using a stack), though recursively may have been much more understandable. Nonetheless, it seems to work, and can potentially avoid stack overflow. Indeed, this tree approach yielded much faster render times. For instance, the cow scene rendered 116 times faster with BVH!

I used the settings "-t 8 -s 64 -l 16 -m 6 -r 480 360 -f" on hive25 machine.

Cow (2088 sec).

Cow (18 sec).

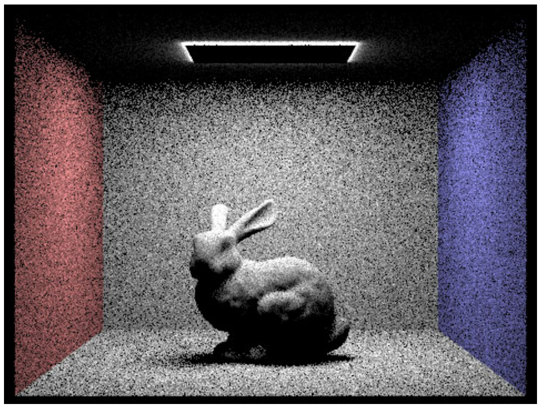

Lucy (21 sec).

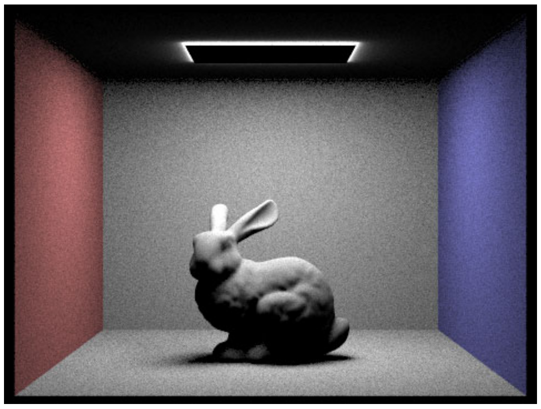
Max Planck (22 sec).

Part 3: Direct Illumination

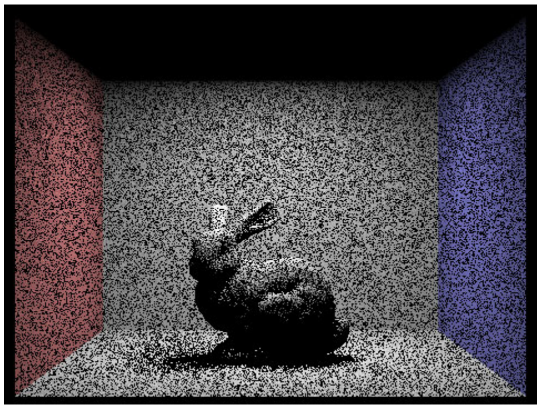
Hemisphere sampling: Sample wi around a hemisphere enclosing the hit point hit_p. Convert wi to world space wi_world. The PDF is constant at 1/(2*pi). Create a ray with origin hit_p + EPS_D * wi_world and direction wi_world. Find intersection of that ray using BVH. If an intersection is found, get the emission multiplied by BSDF and wi.z (cosine between wi and normal (0,0,1)) and (1/PDF). Increment the total spectrum by this amount, and divide by num_samples when finished with the loop.

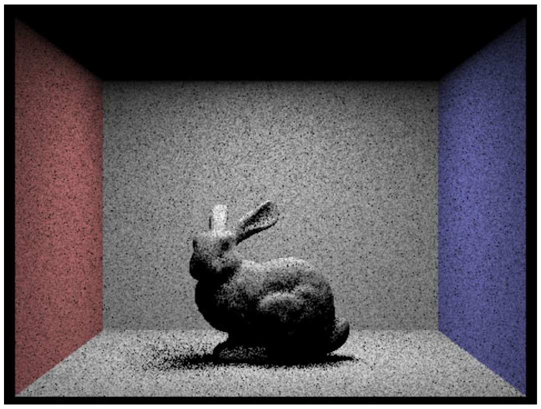
Ligt sampling: For a given hit point hit_p, loop through each scene light. Create variables for the PDF, wi, and distToLight. These values will get automatically computed when sampling around the light. Convert wi to object space w_in. If w_in is less than 0, then sampled light point lies behind surface, so disregard it. Create a ray with origin hit_p + EPS_D * wi and direction wi. Set the max_t of the ray to distToLight. Find intersection of that ray using BVH. If there is no intersection, then no surface is blocking the light contribution, thus we increment the total spectrum by the emission multiplied by the BSDF, w_in.z, and 1/PDF.

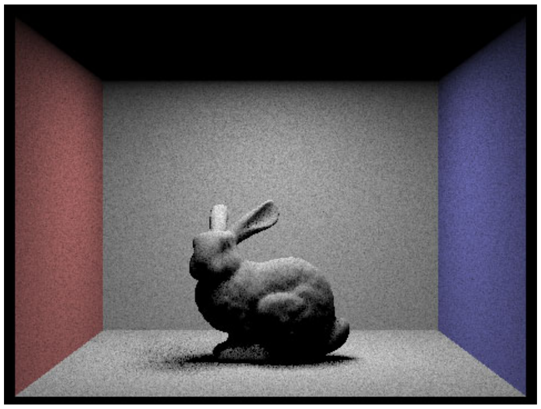

Hemisphere sampling: Bunny (29 sec).

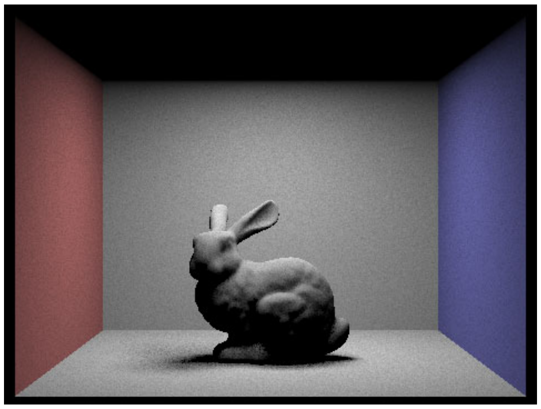

Hemisphere sampling: Bunny (377 sec).

For the left image, I used "./pathtracer -t 8 -s 16 -l 8 -m 6 -H -f CBbunny_16_8.png -r 480 480 ../dae/sky/CBbunny.dae". For the right image, I used "./pathtracer -t 8 -s 64 -l 32 -m 6 -H -f CBbunny_64_32.png -r 480 360 ../dae/sky/CBbunny.dae"


Light sampling: Bunny, 1 light rays (0.3 sec).


Light sampling: Bunny, 4 light rays (0.7 sec).

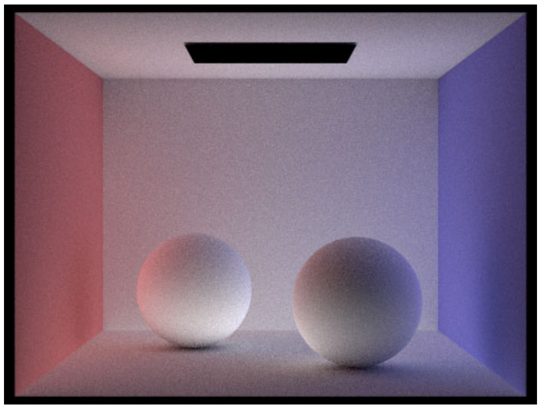

Light sampling: Bunny, 16 light rays (3 sec).

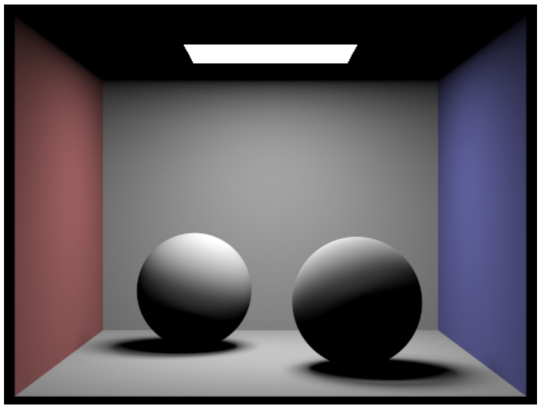

Light sampling: Bunny, 64 light rays (10 sec).

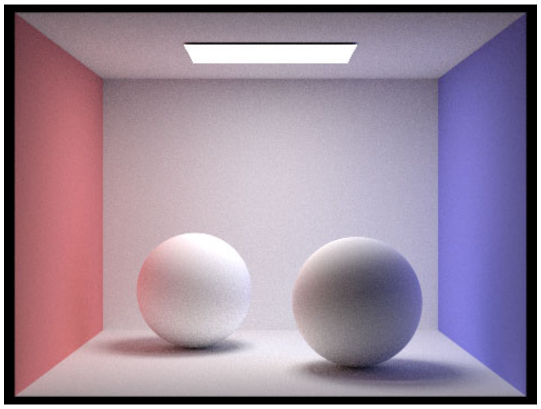
In hemisphere sampling, we sample around the hemisphere enclosing the hit point. in light sampling, we sample around the light within the solid angle to the hig point. The latter yields less noise. This can be seen through hemisphere sampling (in Lecture 12, Slide 35), where sampling around the hemisphere enclosing the hit point yields a lot of noise, especially when using few sample points. In Lecture 12, Slide 41, we see that light sampling does not suffer as much from using fewer sample points.

Part 4: Global Illumination

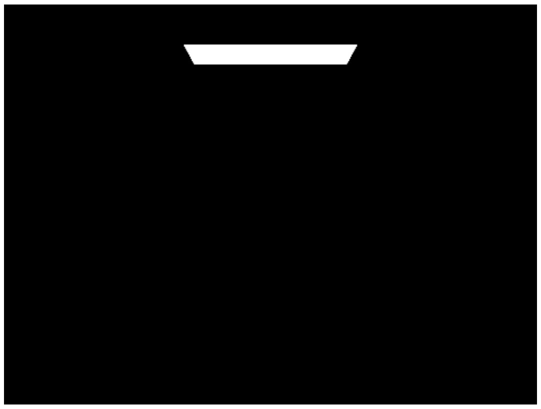
Direct lighting includes zero_bounce_radiance and one_bounce_radiance. However, the at_least_one_bounce_radiance function computes the first one bounce and subsequent indirect lighting bounces. That first bounce is direct lighting, thus indirect lighting is the result from at_least_one_bounce_radiance minus one_bounce_radiance. one_bounce_radiance is simply the direct lighting (hemisphere sampling or importance (light) sampling) functions developed in Part 3. at_leasT_one_bounce_radiance is called only if max_ray_depth (set by -m flag) is greater than 1. In at_least_one_bounce_radiance, we call one_bounce at least once. We decrement the depth of the ray from max_ray_depth to max_ray_depth - 1. Then, as long as the ray's depth is greater than 1 or if a Russian Roulette probability terminates the function, we return L_out.

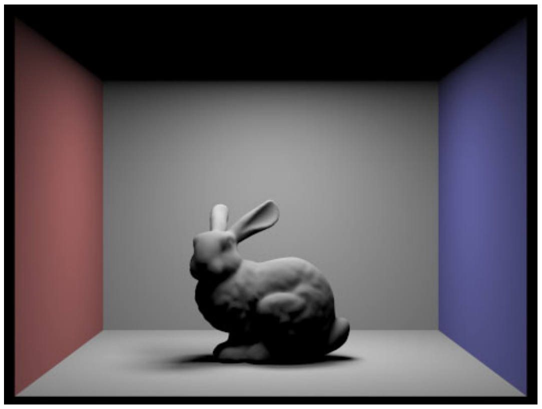

Indirect Illumination: Spheres.

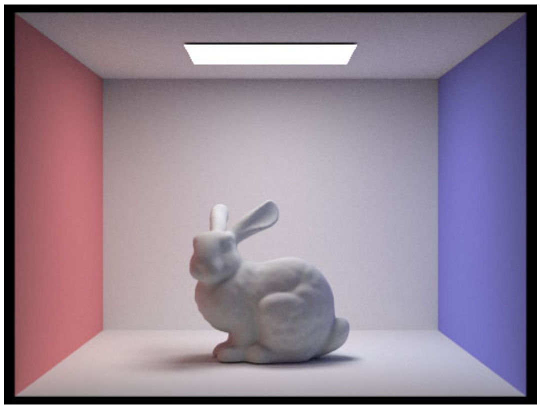

Direct Illumination: Spheres.

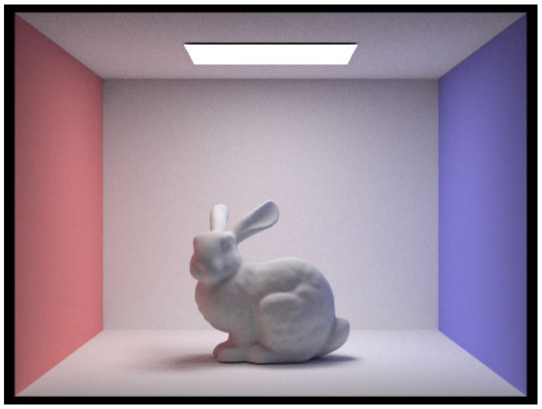

Global Illumination: Spheres.

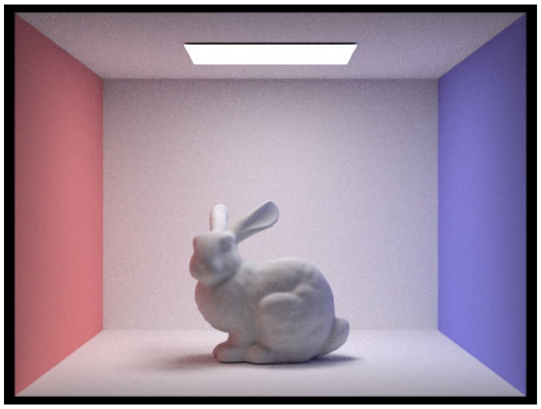
The command "./pathtracer -t 8 -s 64 -l 16 -m 5 -r 480 360 -f spheres.png ../dae/sky/CBspheres_lambertian.dae" was used.


m = 0.

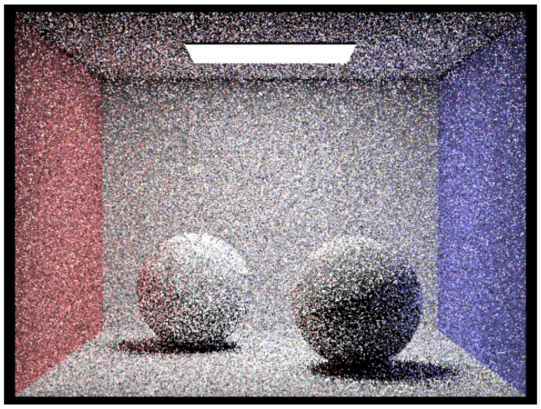

m = 1.

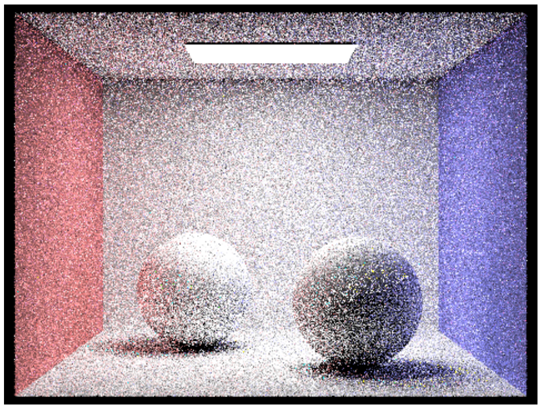

m = 2.

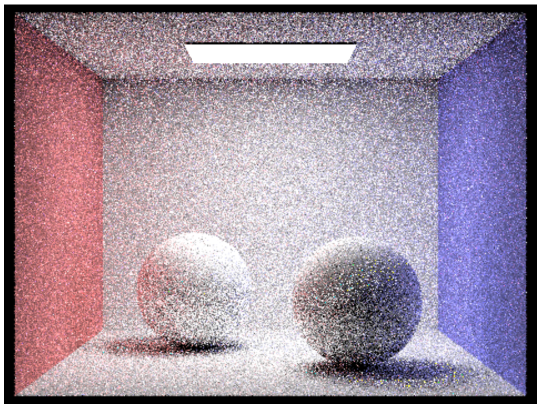

m = 3.

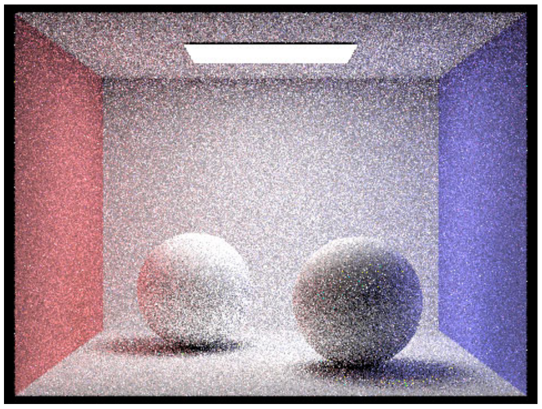

m = 100.

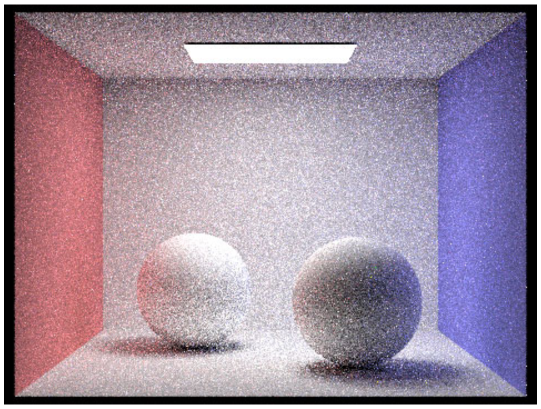
I used the command: "./pathtracer -t 8 -s 1024 -l 16 -m X -r 480 360 -f bunnymX.png ../dae/sky/CBbunny.dae" where X is the max ray depth.

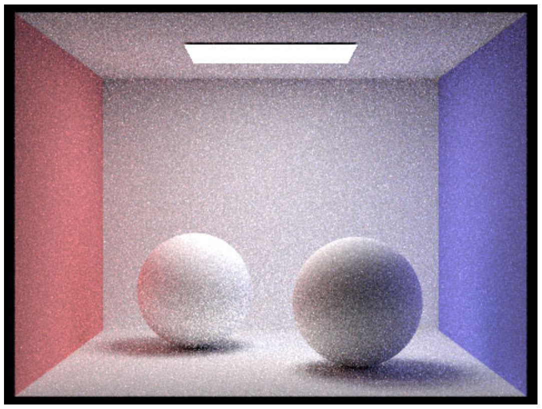

s = 1.

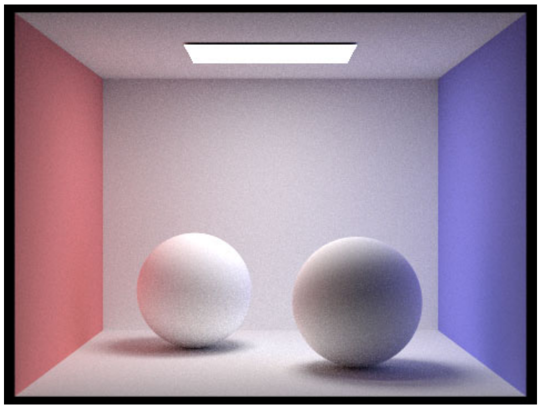

s = 2.


s = 4.


s = 8.


s = 16.

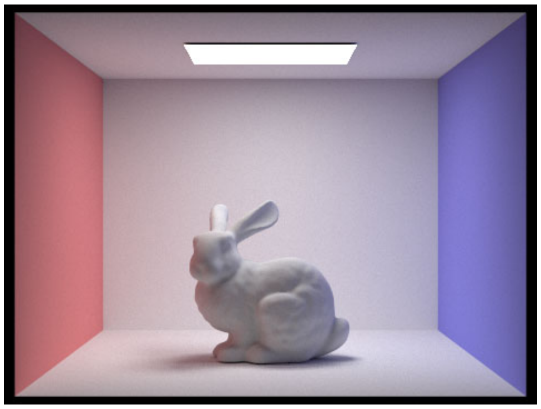

s = 64.

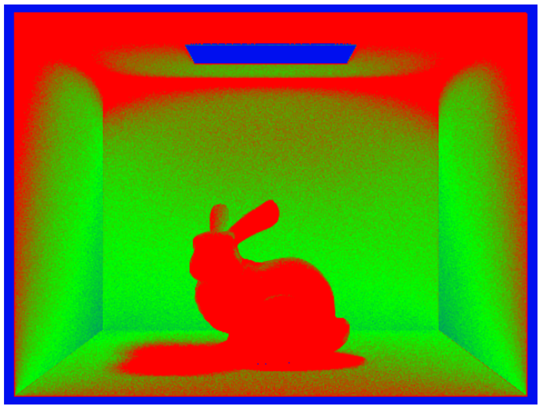

s = 1024.

I used the command: "./pathtracer -t 8 -s X -l 4 -m 5 -r 480 360 -f spheressX.png ../dae/sky/CBspheres_lambertian.dae" where X is the samples-per-pixel rates.

Part 5: Adaptive Sampling

To check convergence every samplesPerBatch samples, I simply used a modulo check: n % samplesPerPatch == 0. If the convergence condition (I less than or equal to maxTolerance * mu), then the loop is broken, and n_converge (which may be less than num_samples) is stored into sampleCountBuffer.


Bunny (sample).


Bunny (rate).

I ran the command "-t 8 -s 2048 -a 64 0.05 -l 1 -m 5 -r 480 360".