Assignment 3: PathTracer

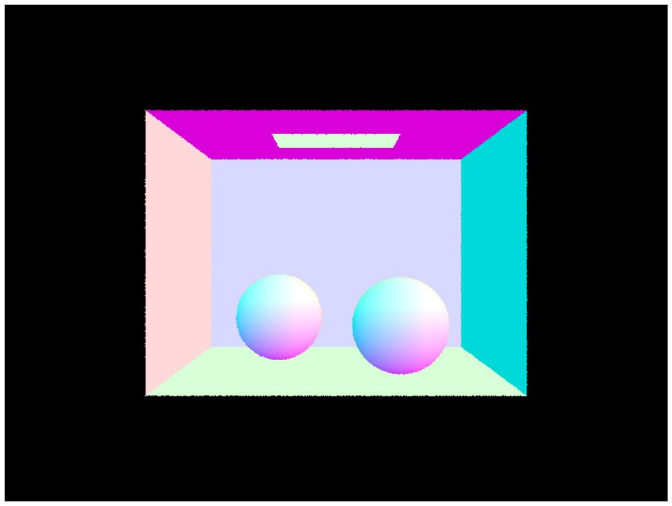
Shikai Qiu

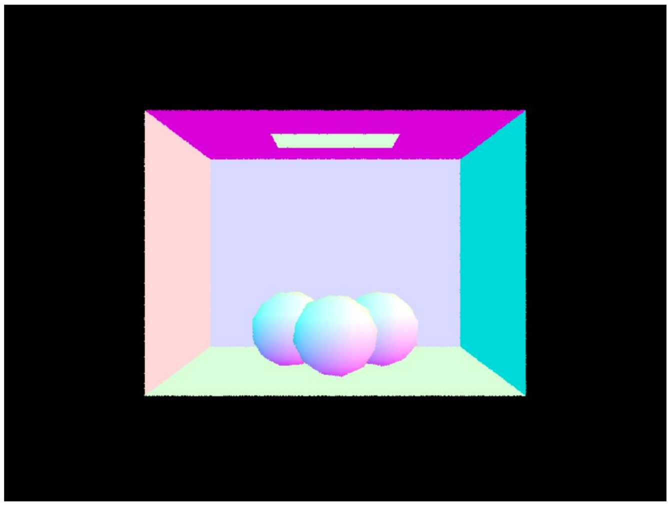
In this project, I implement some core routines of a physically-based renderer using a pathtracing algorithm, relying on ray-scene intersection, acceleration using Bounding Volume Hierarchy, and physically based lighting effects. Sampling is used heavily to evaluate integrals efficiently and accurately. Importance sampling is used to evaluate radiance integrals with reduced variance, while Russian Roulette is used to create unbiased estimators for infinite-dimensional path integrals over light trajectories.

Part 1: Ray Generation and Intersection

Ray generation and primitive intersection are the most basic components in ray tracing. In ray generation, rays passing through points uniformly distributed over a pixel are generated for evaluating the integral of radiance over the pixel. For each ray, its intersection with the scene is determined by the primitive intersection algorithms. In case of triangles and spheres, these algorithms can be simple and exact.

The triangle intersection algorithm is implemented by finding the parameter t for which the ray intersects the triangle and the barycentric coordinates of the intersection via the Möller-Trumbore algorithm. It claims a valid intersection if and only if the t value if between the minimum and maximum allowed t value of the ray and that all three barycentric coordinates are between 0 and 1.


Spheres with normal shading


Gems with normal shading

Part 2: Bounding Volume Hierarchy

The BVH construction algorithm constructs a binary BVH-tree recursively. In each call starting from the root, the algorithm checks whether the current number of primitives is within a maximum value. If so, the current node is set to a leaf, with all current primitives saved to a member variable of the node. Otherwise the algorithm recursively constructs a left and right node by splitting the list of primitives into two. A primitive will either go to the left node or the right node based on whether the coordinate of its centroid is greater than that of the mid point of the current bounding box along its longest axis.

1 Part 1: Ray Generation & Scene Intersection 20 / 20

✓ + 5 pts Explanation of ray generation

✓ + 10 pts Explanation of triangle/sphere intersection

✓ + 5 pts Show normal intersection

- 2 pts Overly simplistic explanations or insufficient results

+ 0 pts Blank/Incorrect

Assignment 3: PathTracer

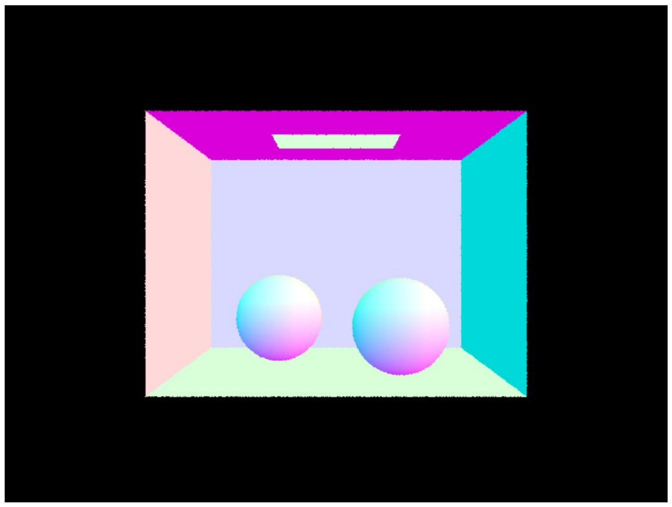
Shikai Qiu

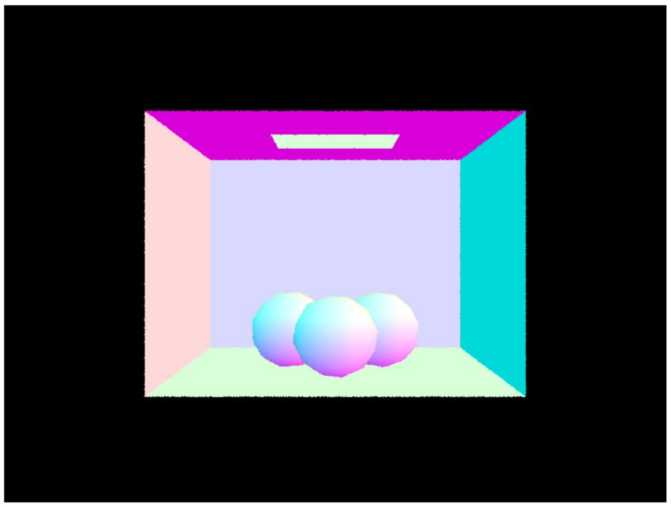
In this project, I implement some core routines of a physically-based renderer using a pathtracing algorithm, relying on ray-scene intersection, acceleration using Bounding Volume Hierarchy, and physically based lighting effects. Sampling is used heavily to evaluate integrals efficiently and accurately. Importance sampling is used to evaluate radiance integrals with reduced variance, while Russian Roulette is used to create unbiased estimators for infinite-dimensional path integrals over light trajectories.

Part 1: Ray Generation and Intersection

Ray generation and primitive intersection are the most basic components in ray tracing. In ray generation, rays passing through points uniformly distributed over a pixel are generated for evaluating the integral of radiance over the pixel. For each ray, its intersection with the scene is determined by the primitive intersection algorithms. In case of triangles and spheres, these algorithms can be simple and exact.

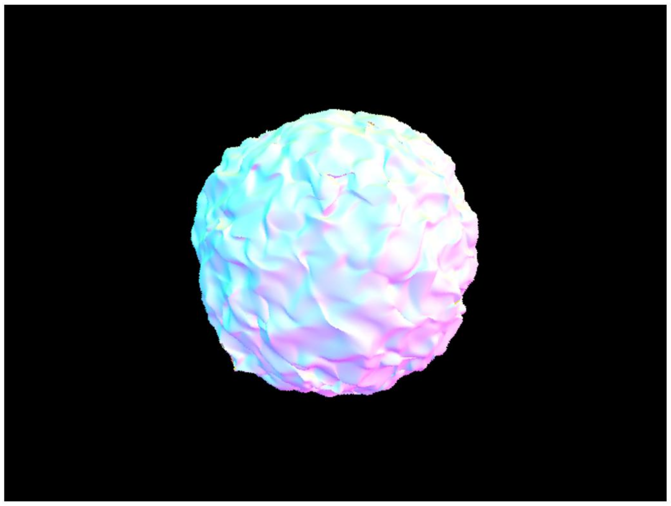
The triangle intersection algorithm is implemented by finding the parameter t for which the ray intersects the triangle and the barycentric coordinates of the intersection via the Möller-Trumbore algorithm. It claims a valid intersection if and only if the t value if between the minimum and maximum allowed t value of the ray and that all three barycentric coordinates are between 0 and 1.

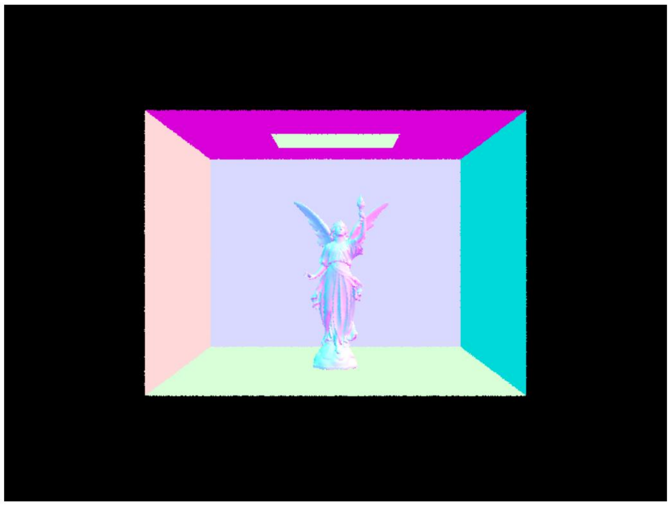

Spheres with normal shading


Gems with normal shading

Part 2: Bounding Volume Hierarchy

The BVH construction algorithm constructs a binary BVH-tree recursively. In each call starting from the root, the algorithm checks whether the current number of primitives is within a maximum value. If so, the current node is set to a leaf, with all current primitives saved to a member variable of the node. Otherwise the algorithm recursively constructs a left and right node by splitting the list of primitives into two. A primitive will either go to the left node or the right node based on whether the coordinate of its centroid is greater than that of the mid point of the current bounding box along its longest axis.

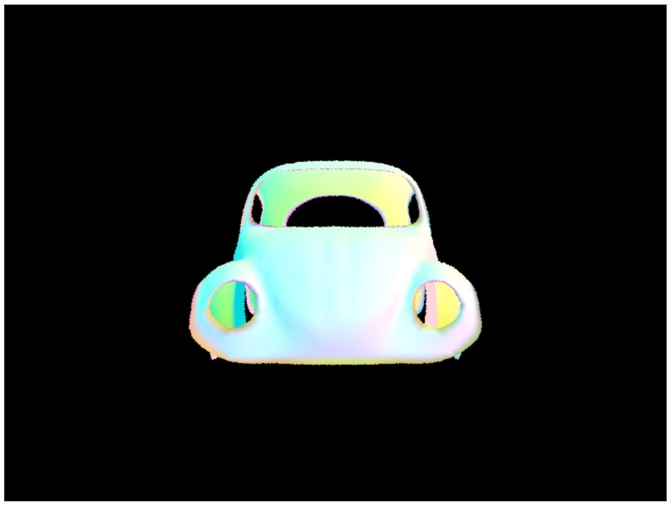

A blob with normal shading


CBlucy with normal shading

The BVH significantly reduces the rendering time as is evident from testing on the two following moderately complex geometries with 8 threads and 800 * 600 resolution and default for all other options.

beetle.dae: Without BVH: 45.44 seconds in total, 1639.4 intersection tests per ray; With BVH: 0.055s seconds in total, 1.76 intersection tests per ray.

cow.dae: Without BVH: 58.31 seconds in total, 2143.0 intersection tests per ray; With BVH: 0.08s seconds in total, 2.74 intersection tests per ray.


A beetle with normal shading

A cow with normal shading

Part 3: Direct Illumination

In this part, I implemented two methods for direct lighting: Uniform Hemisphere Sampling and Importance Sampling over Lights.

In Uniform Hemisphere Sampling, we estimate the direct lighting integral at a ray-scene intersection in a certain outgoing direction by performing uniform sampling over the unit hemisphere pointed to by the surface normal vector. For each sampled direction, we trace another ray along it and find the first intersection with the scene and record the emitted radiance at that point along the reverse ray. The average of all such sampled radiance multiplied by the appropriate BSDFs, cosine values, and a normalization factor equal to the inverse probability is returned as the Monte Carlo estimate for the integral.

In Importance Sampling over Lights, the algorithm is only different at the sampling method. Instead of uniform random sampling, we sampling only from light sources by tracing shadow rays from each light source to the intersection of interest. When the shadow ray is unobstructed by other objects, we add the sampled radiance from the light source weighted by the inverse probability density for the sampled direction to the estimate, and normalize at the end.

The following 4 images are generated with 1 sample per pixel and 1, 4, 16, and 64 sample(s) per area light respectively. We see a significant decrease in the noise level in the soft shadows below the bunny as we increase the number of sampling rate.

2 Part 2: BVH 20 / 20

✓ **+ 4 pts** **[BVH Construction] Walk through your BVH construction algorithm, including recursive case and base case**

✓ **+ 3 pts** **[BVH Construction] Explain the heuristic you chose for picking the splitting point**

✓ **+ 6 pts** **[Normal Shading] Show images with normal shading for a few large .dae files that you can only render with BVH acceleration**

✓ **+ 4 pts** **[Rendering Experiments] Compare rendering times on a few scenes with moderately complex geometries with and without BVH acceleration**

✓ **+ 3 pts** **[Rendering Experiments] 1-paragraph analysis of rendering time experiment**

+ 2 pts [Extra Credit] Surface area heuristic used and explained

+ 2 pts [Extra Credit] Iterative implementation of BVH construction and intersection

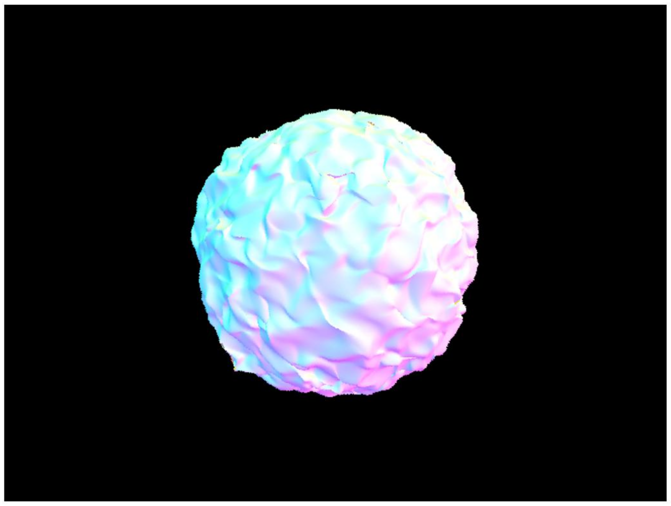
+ 2 pts [Extra Credit] Implement a more memory efficient BVH by storing all the $$\texttt{Primitive}$$ pointers in one large vector

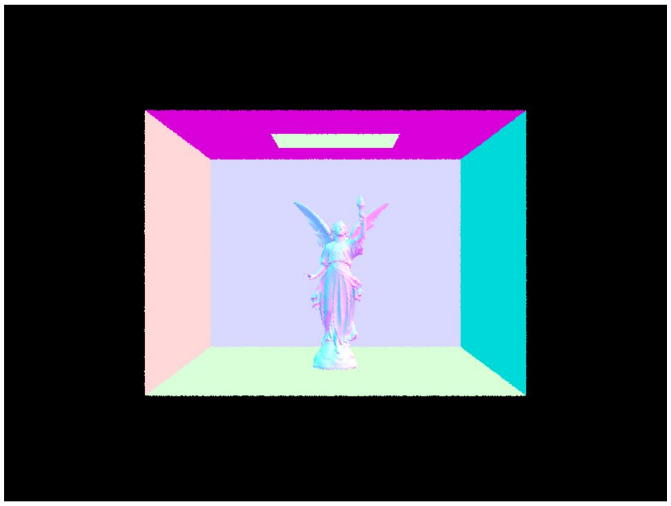
+ 3 pts [Extra Credit] Implement and compare an alternate acceleration structure

+ 3 pts [Extra Credit] Implement non-axis-aligned bounding boxes and show that it benefits ray traversal

+ 0 pts Blank or Insufficient

- 2 pts Missing key details or overly simplistic explanations

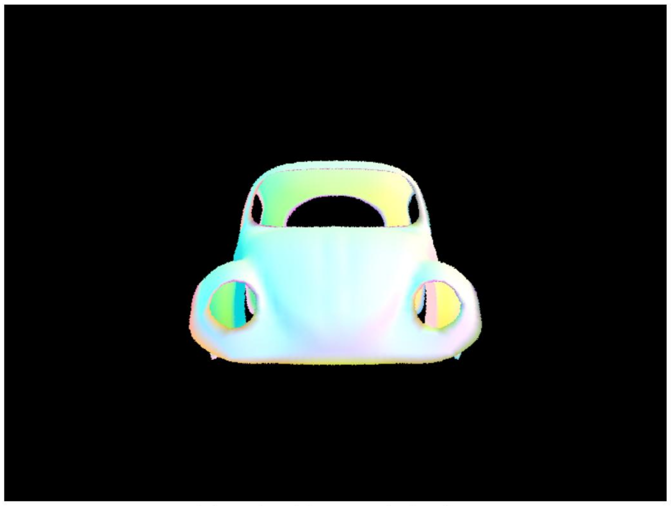

A blob with normal shading


CBlucy with normal shading

The BVH significantly reduces the rendering time as is evident from testing on the two following moderately complex geometries with 8 threads and 800 * 600 resolution and default for all other options.

beetle.dae: Without BVH: 45.44 seconds in total, 1639.4 intersection tests per ray; With BVH: 0.055s seconds in total, 1.76 intersection tests per ray.

cow.dae: Without BVH: 58.31 seconds in total, 2143.0 intersection tests per ray; With BVH: 0.08s seconds in total, 2.74 intersection tests per ray.


A beetle with normal shading

A cow with normal shading

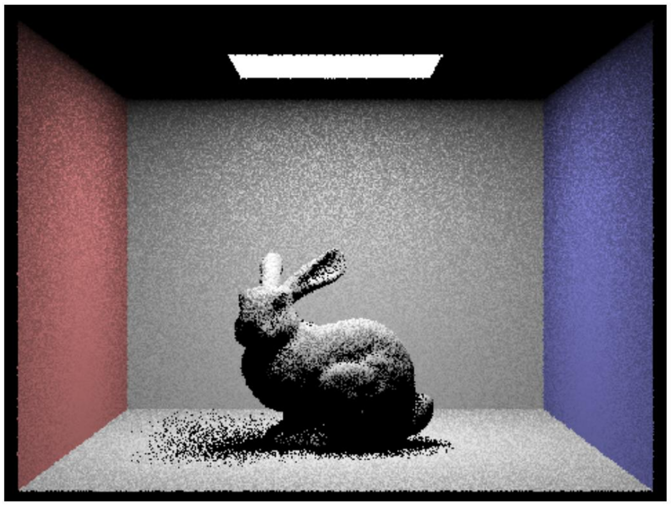
Part 3: Direct Illumination

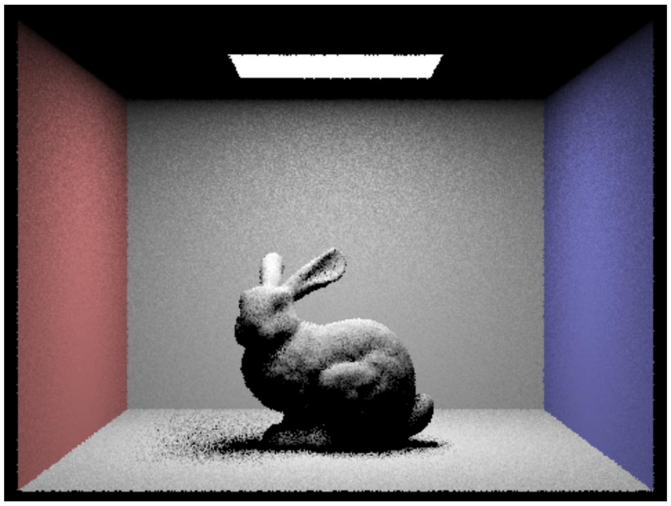
In this part, I implemented two methods for direct lighting: Uniform Hemisphere Sampling and Importance Sampling over Lights.

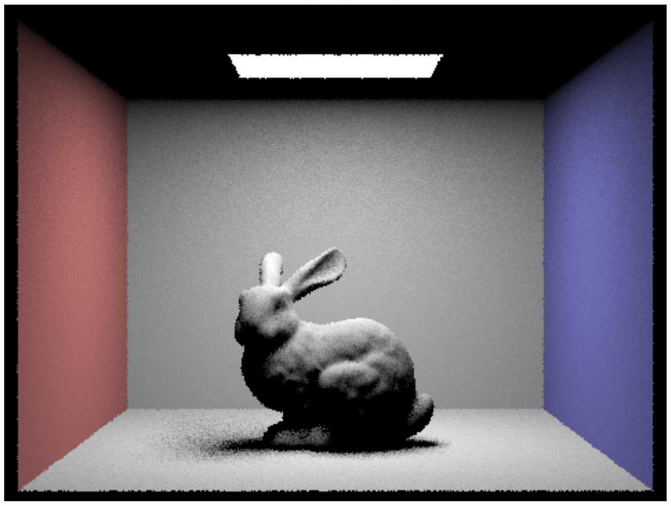
In Uniform Hemisphere Sampling, we estimate the direct lighting integral at a ray-scene intersection in a certain outgoing direction by performing uniform sampling over the unit hemisphere pointed to by the surface normal vector. For each sampled direction, we trace another ray along it and find the first intersection with the scene and record the emitted radiance at that point along the reverse ray. The average of all such sampled radiance multiplied by the appropriate BSDFs, cosine values, and a normalization factor equal to the inverse probability is returned as the Monte Carlo estimate for the integral.

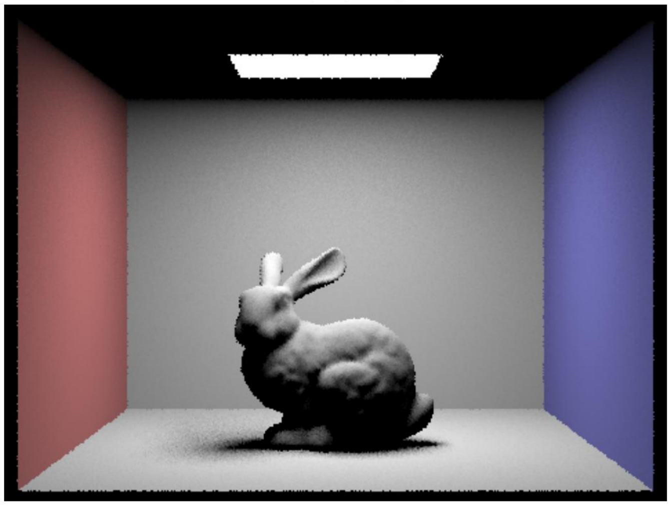
In Importance Sampling over Lights, the algorithm is only different at the sampling method. Instead of uniform random sampling, we sampling only from light sources by tracing shadow rays from each light source to the intersection of interest. When the shadow ray is unobstructed by other objects, we add the sampled radiance from the light source weighted by the inverse probability density for the sampled direction to the estimate, and normalize at the end.

The following 4 images are generated with 1 sample per pixel and 1, 4, 16, and 64 sample(s) per area light respectively. We see a significant decrease in the noise level in the soft shadows below the bunny as we increase the number of sampling rate.

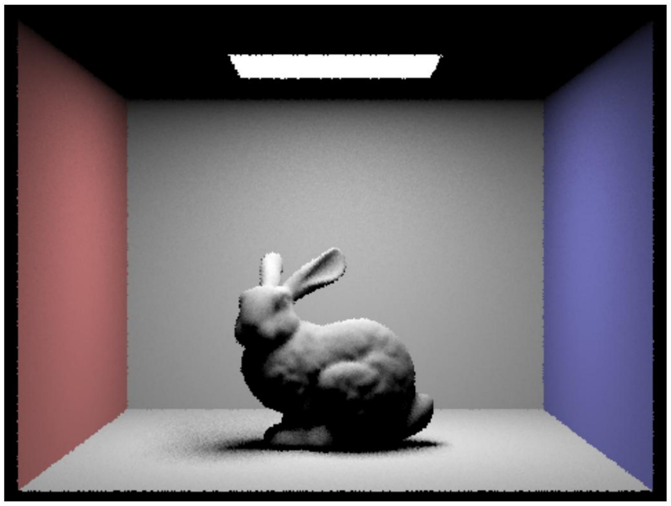

1 sample per light

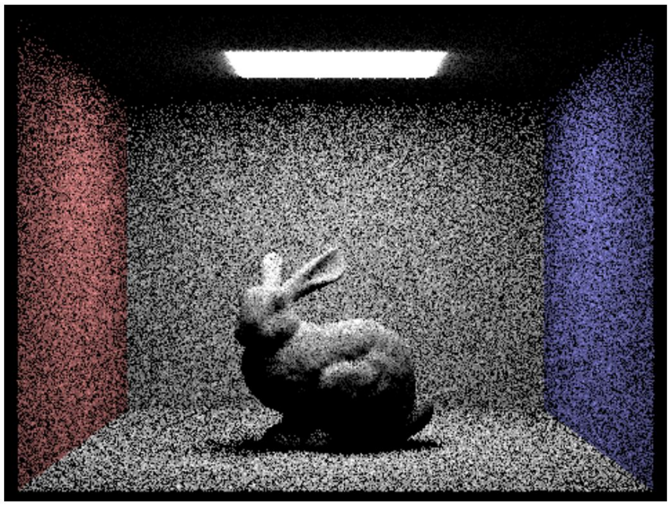

4 sample per light


16 sample per light


64 sample per light

In contrast with Uniform Hemisphere Sampling, Importance Sampling over Lights clearly does a much better job in reducing noise. This reflects the fact that the probability distribution over sampling directions in Importance Sampling closely matches the amplitude of the function whose integral we intend to estimate, which in this case is the (properly weighted) emitted radiance from light sources. This is known to lead to lower variance of the estimator. Intuitively, this is also clear as a large fraction of rays traced by the Uniform Hemisphere Sampling do not hit any light source and therefore leaves dark points throughout the image.


Importance sampling with 64 sample per light


Uniform sampling with 64 sample per light

Part 4: Direct Illumination

By definition, the indirect lighting on an object is the result of applying the one-bounce operator on the all lighting but emission in the scene. Therefore, we can compute the indirect lighting recursively by first sampling the direct lighting bouncing into the outgoing direction from some other intersection using previously defined methods, and then making an

3 Part 3: Direct Illumination 20 / 20

✓ + **2 pts** Walk through uniform hemisphere sampling

✓ + **3 pts** Walk through light importance sampling

✓ + **2 pts** Correct screenshots of uniform hemisphere sampling (direct lighting only)

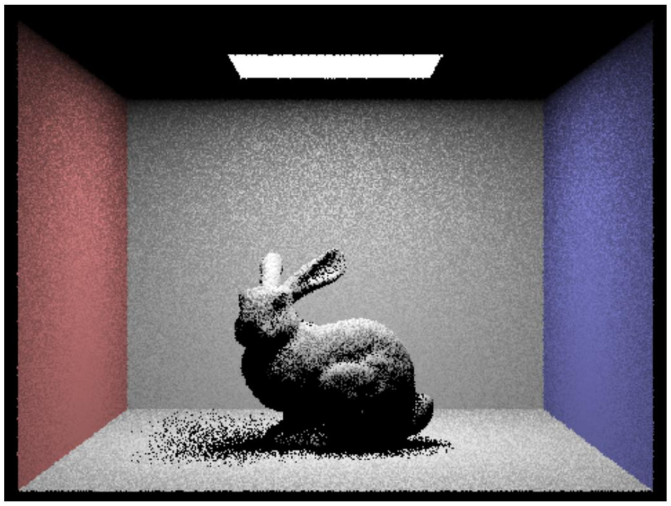
✓ + **3 pts** Correct screenshots of light importance sampling (direct lighting only)

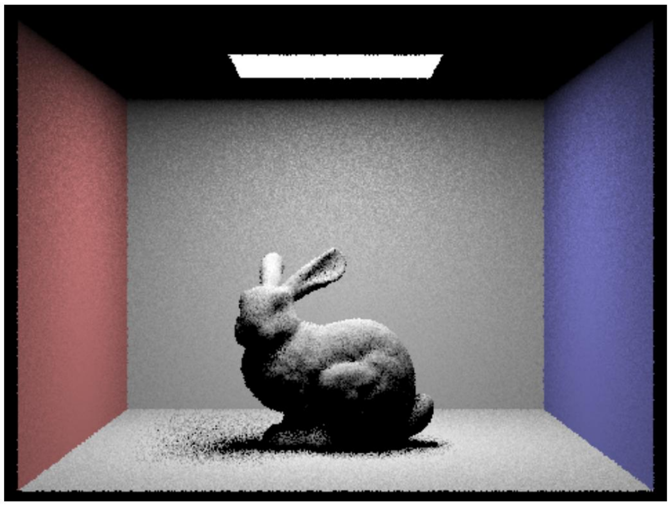
✓ + **3 pts** Compare the results between using uniform hemisphere sampling and light importance sampling (screenshots)

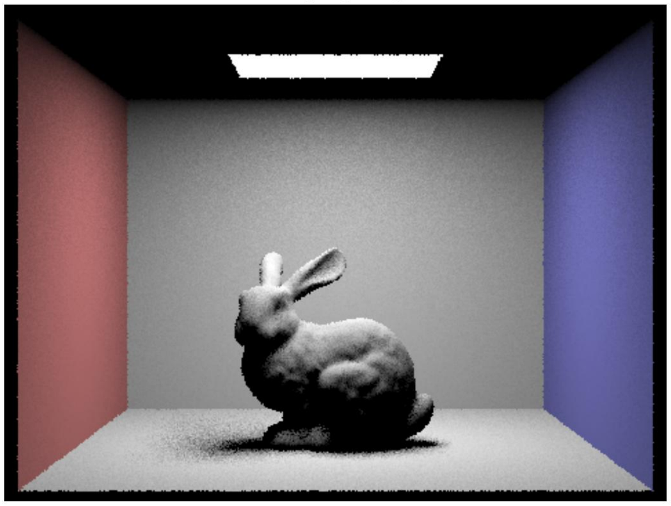
✓ + **2 pts** One paragraph analysis

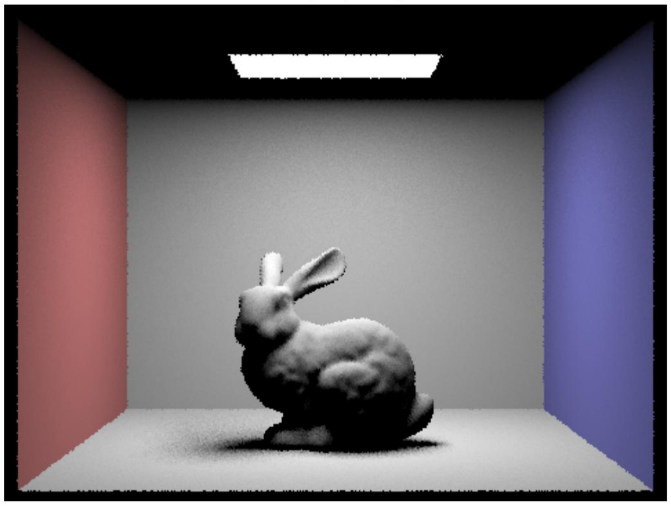
✓ + **5 pts** Focus on one particular scene (with at least one area light) and compare the noise levels in soft shadows when rendering with 1, 4, 16, and 64 light rays (the -l flag) and 1 sample per pixel (the -s flag) using light importance sampling

+ **0 pts** No answer

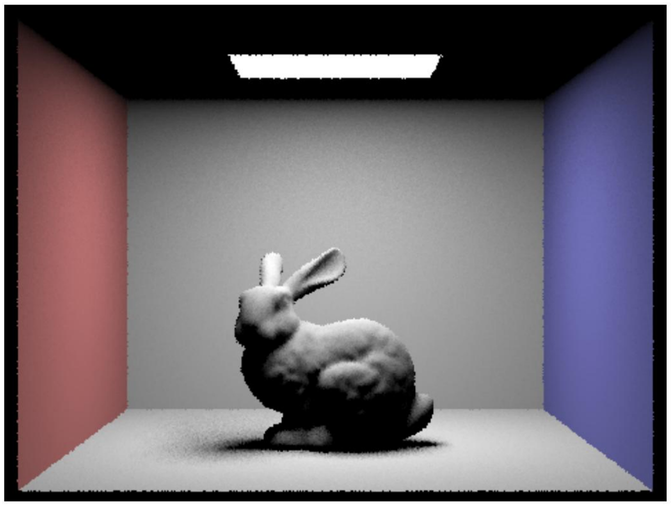

1 sample per light

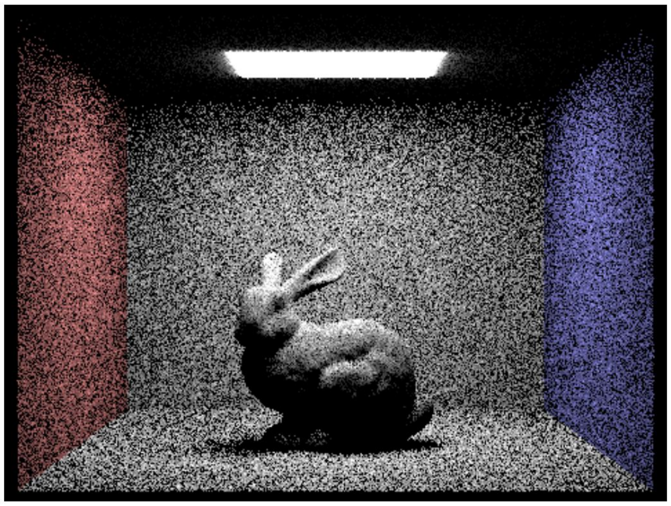

4 sample per light


16 sample per light


64 sample per light

In contrast with Uniform Hemisphere Sampling, Importance Sampling over Lights clearly does a much better job in reducing noise. This reflects the fact that the probability distribution over sampling directions in Importance Sampling closely matches the amplitude of the function whose integral we intend to estimate, which in this case is the (properly weighted) emitted radiance from light sources. This is known to lead to lower variance of the estimator. Intuitively, this is also clear as a large fraction of rays traced by the Uniform Hemisphere Sampling do not hit any light source and therefore leaves dark points throughout the image.

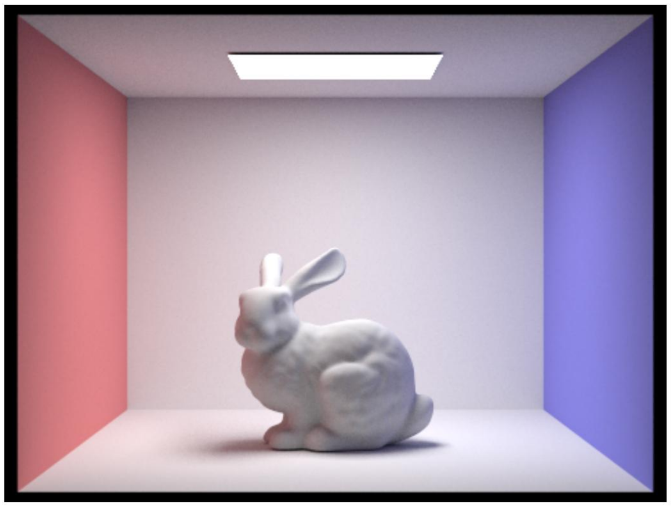

Importance sampling with 64 sample per light

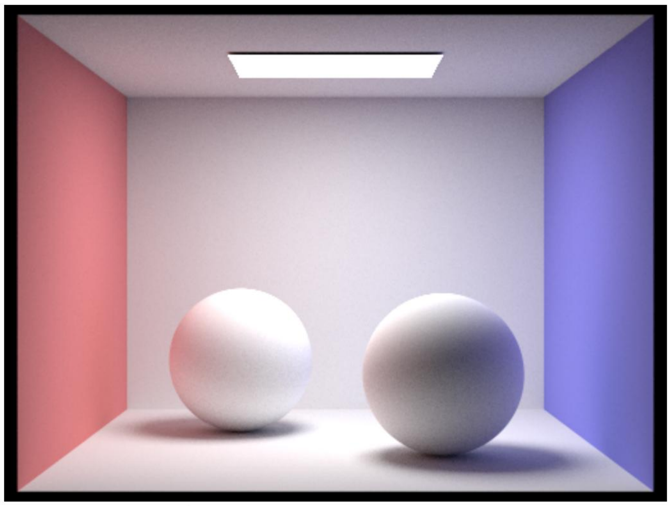

Uniform sampling with 64 sample per light

Part 4: Direct Illumination

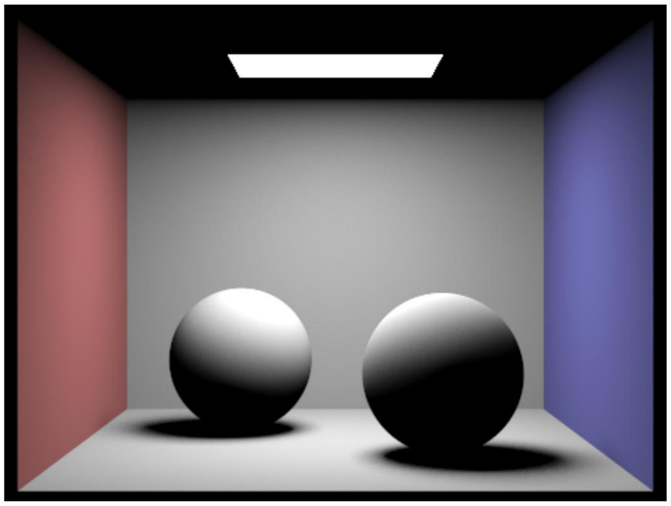
By definition, the indirect lighting on an object is the result of applying the one-bounce operator on the all lighting but emission in the scene. Therefore, we can compute the indirect lighting recursively by first sampling the direct lighting bouncing into the outgoing direction from some other intersection using previously defined methods, and then making an recursive call to sample indirect lighting bouncing into the outgoing direction from some other intersection. For the evaluation of this infinite-dimensional path integral of radiance along possible light rays to terminate, we use Russian Roulette to implement probabilistic termination at a probability of 0.65 per call. We also record the current depth of a ray in order to terminate when its value exceeds the cut-off given by max_ray_depth.

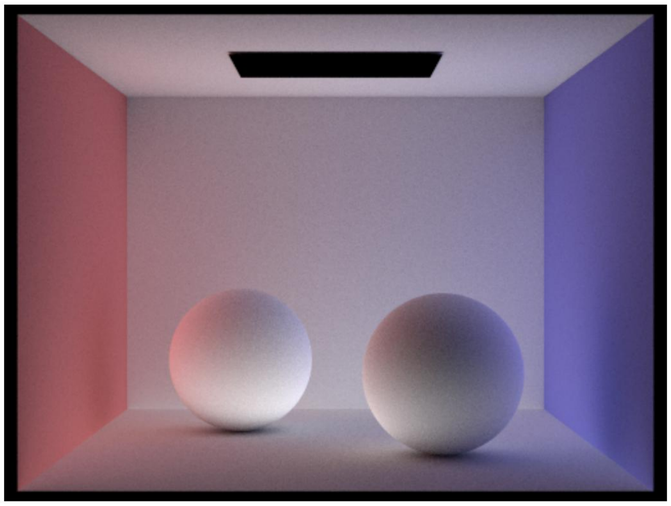
Some images with global illumination with 1024 samples per pixel and max_ray_depth = 5


Bunny with global illumination

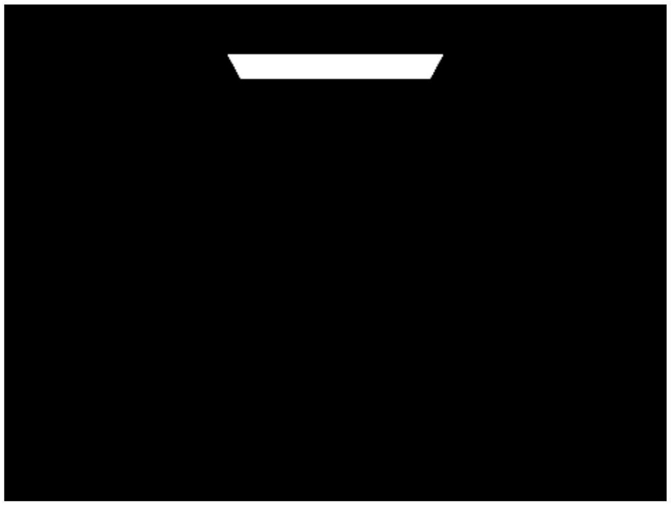

Spheres with global illumination

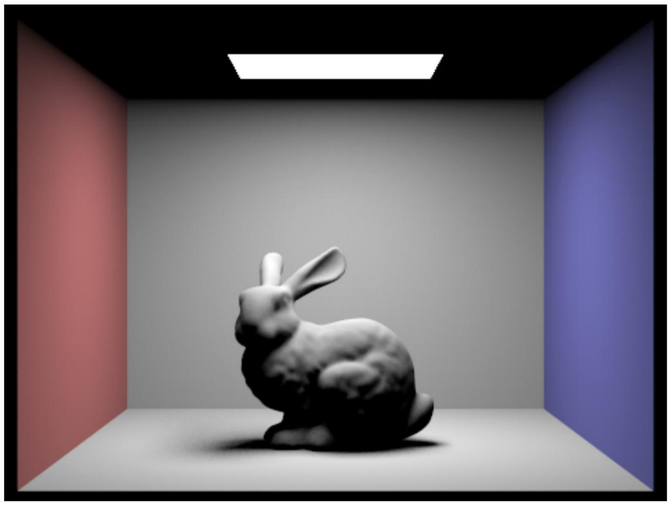
Direct v.s. Indirect illumination with 1024 samples per pixel and max_ray_depth = 5

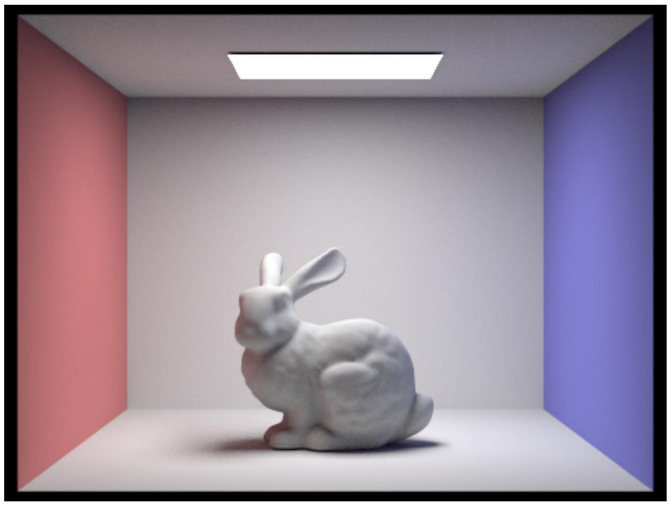

Spheres with direct illumination only

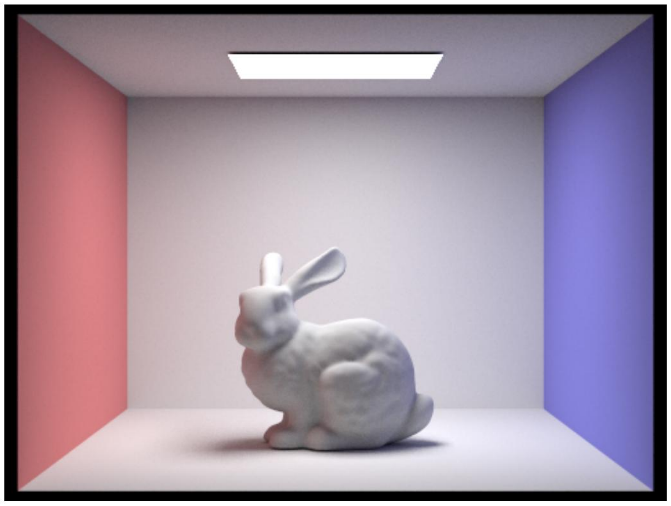

Spheres with indirect illumination only

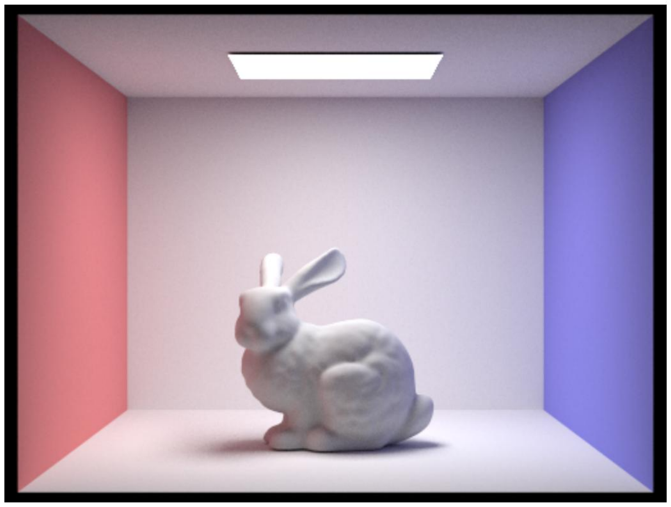
The effect of different max_ray_depth with 1024 samples per pixel


max_ray_depth = 0

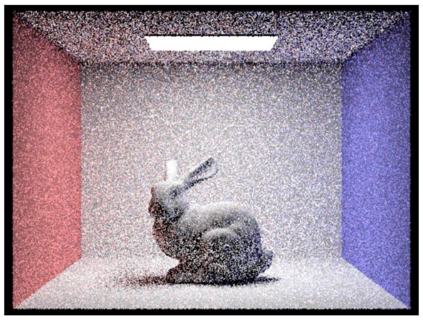

max_ray_depth = 1

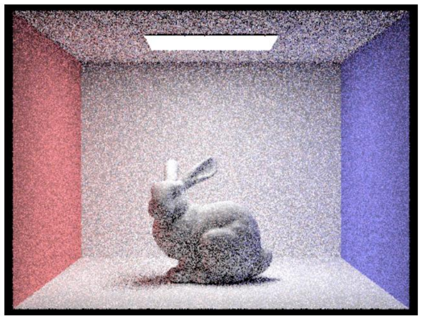

max_ray_depth = 2

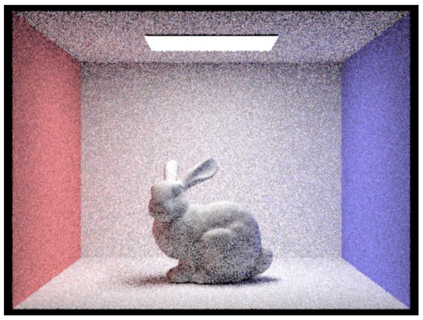

max_ray_depth = 3

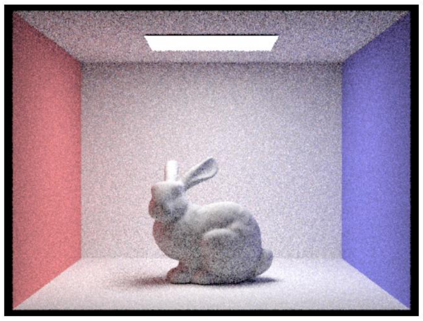

max_ray_depth = 100

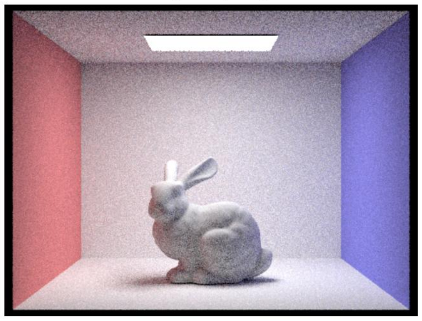
The effect of different sample-per-pixel-rates with 4 light rays

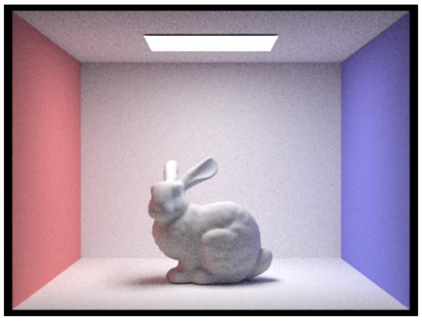

1 samples per pixel


2 samples per pixel


4 samples per pixel


8 samples per pixel


16 samples per pixel


64 samples per pixel

4 Part 4: Global Illumination 18 / 20

✓ + **10 pts** **Included Sample Renders and Comparisons**

✓ + **10 pts** **Correct Explanation**

✓ - **2 pts** **Explanations Insufficient / Overly Simplistic**

- **2 pts** Missing max-ray-depth images / images incorrect

- **2 pts** Missing sample-per-pixel comparison images / images incorrect

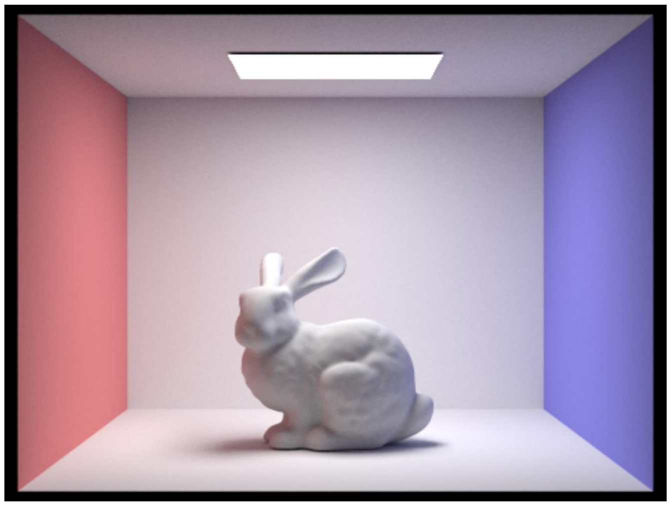
- **1 pts** Missing indirect, direct illumination comparison images / images incorrect

- **0.5 pts** Small error (e.g. Overhead Light present in Indirect Calculation)

+ **0 pts** Blank

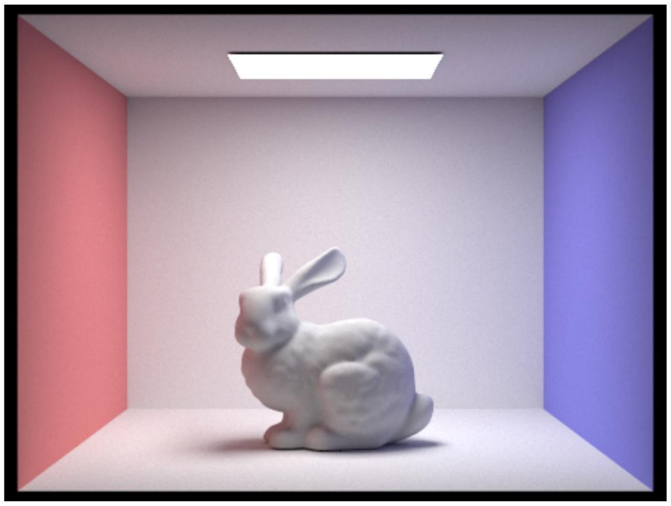
- **1 pts** image(s) too bright / dark

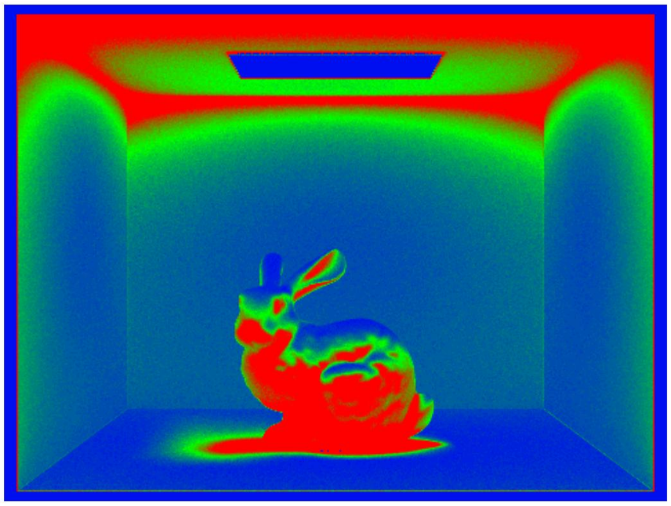
Could discuss more details about implementation and details about different max_ray_depth / sample - per - pixel.


1024 samples per pixel

Part 5: Adaptive Sampling

I implemented adaptive sampling method to concentrate the samples in pixels with higher noise. To do that, for every samplesPerBatch number of samples generated, I check the value of the confidence interval I for the current pixel. If it is smaller than the command-line specified value, I terminate the sampling for this pixel. However, to avoid accidental convergence, I require each pixel must be sampled at least samplesPerBatch number of times. Below is an instance of the image generated by the adaptive sampling and the associated distribution of the number of samples generated. The image is rendered with 2048 smaples per pixel, 1 sample per light, and max ray depth of 5.


Bunny image rendered via adaptive sampling


Distribution of sample rates