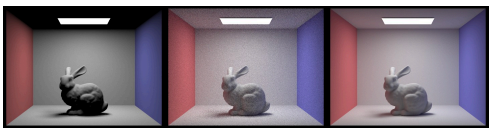
CS184/284A Spring 2025 Homework 3 Write-Up

Names: Owen Zou

Link to webpage: (TODO) cal-cs184-student.github.io/hw-webpages-orz/ Link to GitHub repository: (TODO) github.com/cal-cs184-student/hw-webpages-orz



dreamcore

Overview

In this homework, we implemented the ray tracing, BVH tree to accelerate the rendering of ray tracing, and several illuminations (direct illumination and global illumination) to shade the images. It formulated the visualized light to controllable mathematics variables, help us to do research with them in a more efficient and scientific way.

Part 1: Ray Generation and Scene Intersection

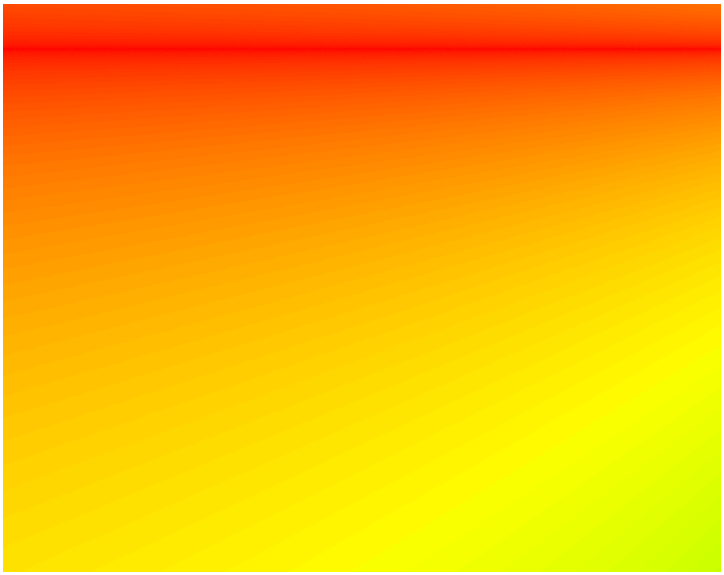
What ray generation function do is to transform the noramlized x and y on image space to the position on the camera space, and return the ray in world space by times c2w matrix to ray's direction. The next step is to write the raytrace_pixel. We use the gridSampler to get the sample. Then, normalize (x + sample[0]) and (y + sample[1]). Generate Ray on that position and estimate the radiance of this ray, add to vector sum. After a num_samples times loop, we normalize the sum by num_samples and update the sample buffer. Inside estimation of radiance process, we will check whether the intersection with primitives occurs. When the intersection takes place, the intersection data like intersection's time, normal surface, bsdf and intersected primitives should be updated correctly. Besides, the ray's max_t will also be updated. Our intersection functions here make effect on detecting whether the intersection occurs and update the intersection instance.

For triangle intersection algorithm, I use the Moller-Trumbore algorithm to calculate the time of intersection.

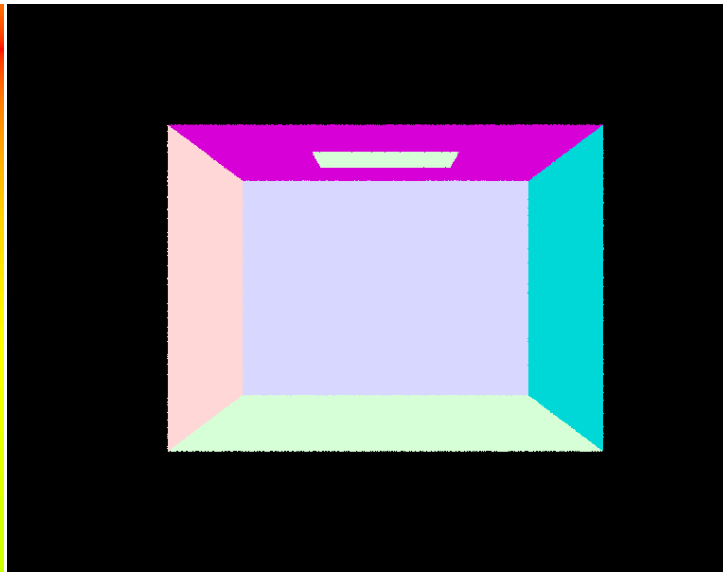
- $O + tD = (1 - b1 - b2)P0 + b1P1 + b2P2$
- we should get [t b1 b2] by calculating

$$1/S.E[S2.E2S1.SS2.D]$$

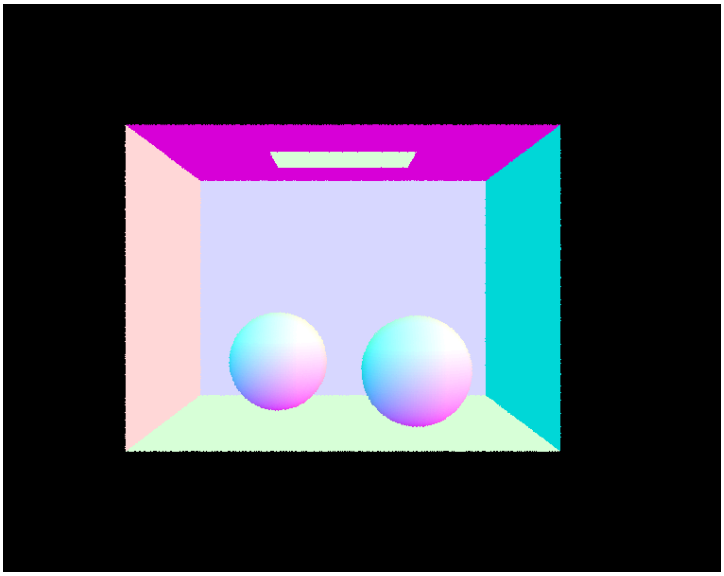
If b1 is less than 0 or b1 is larger than 1, intersection does not take place. If b2 is less than 0 or b2 + b1 is larger than 1, intersection does not take place. If t is not in the range of min_t and max_t, intersection does not take place. If all tests pass, set intersection's time to t, set the normal surface to (1 - b1 - b2) * n1 + b1 * n2 + b2 * n3. Set intesection's primitive and bsdf as the primitive itself and its bsdf



banana



CBempty after triangle intersection



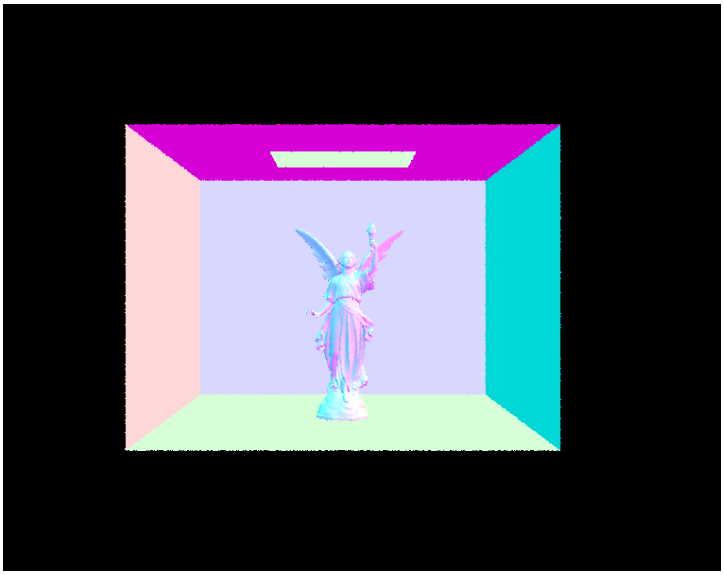
CBspheres after sphere intersection

Part 2: Bounding Volume Hierarchy

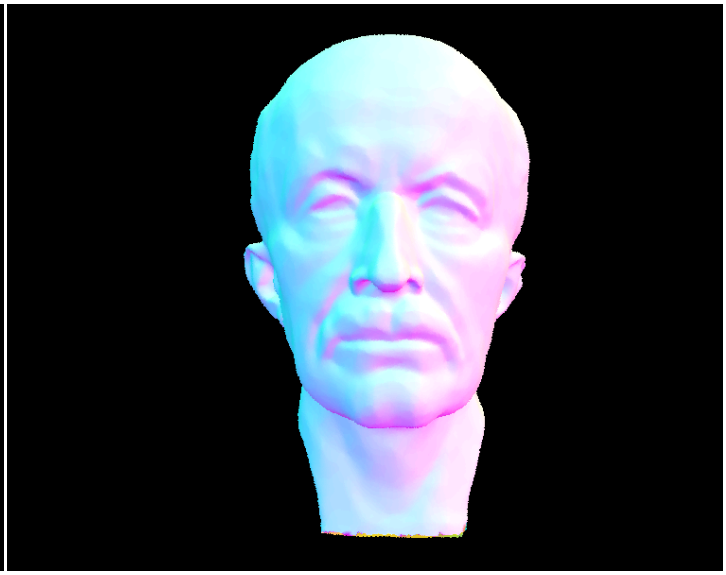
We construct BVH recursively. Base case: If the number of primitives is less than or equal to max_leaf_size, create a leaf node that stores all those primitives. Then we will splitting the start and the end to find the middle. We Calculate the centroid bounding box of all primitives, find the axis with the largest extent, sort primitives along this axis based on their centroid positions, split the primitives in

half to get the mid. Recursive step: Recursively apply the same process to the two halves , (start, mid) as left, (mid, end) as right, until the base case is reached.

The herustic I use here is the median split. I choose the axis with the largest extent in the centroid bounding box and sort primitives along this axis. Finally I Split at the median. The largest extent axis typically represents the dimension with the greatest variation in primitive positions. Splitting along this axis helps ensure that primitives are well-distributed between the two child nodes.



CBlucy



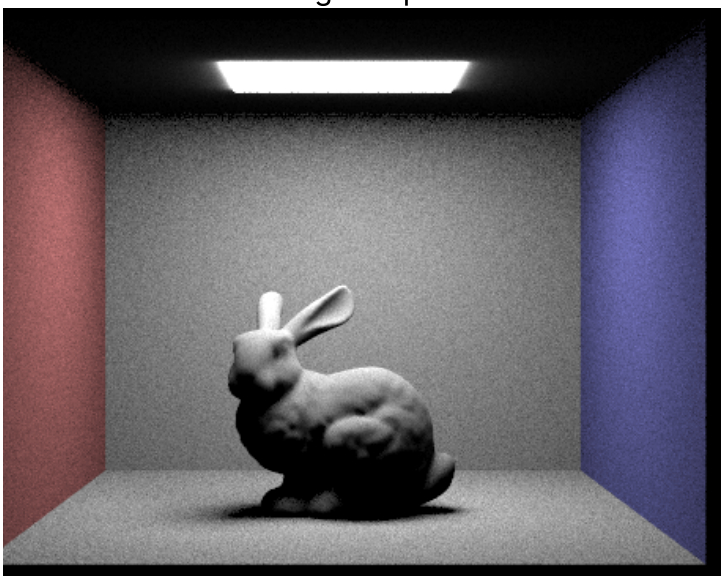
maxplanck

Without BVH, redndering cow need 57s on my local computer. With BVH, it needs only 0.4s. The cow scene exhibited an 142.5 times speedup when using the BVH structure. The magnitude of this speedup aligns with expectations for moderately complex geometries, where simple ray-primitive testing scales linearly with primitive count (O(n)), while BVH-accelerated rendering approaches logarithmic time complexity (O(log n)). For scenes with even greater geometric complexity, the performance gap would likely widen further, so it is mandatory to make BVH accelerationfor for rendering applications.

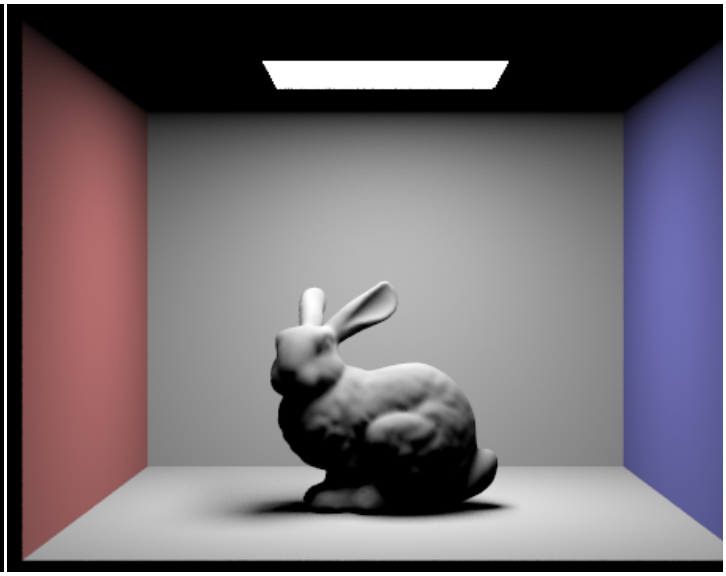
Part 3: Direct Illumination

The direct illuminations consist of two parts: zero bounce radiance and one bounce radiance. The zero bounce radiance returns emissions of the bsdf directly. For one bounce radiance, we have two sampling methods: uniform hemisphere sampling and light importance sampling. Uniform hemisphere sampling is the method we sampled every direction above the surface uniformly. After choosing a direction, we trace a small “shadow ray” into the scene. If it hits an emissive object, we collect that object's emission. Then we use the Monte Carlo nomalization to estimate the radiance out. For lighting importance method, we specifically sample directions that point toward the light sources. light→sample_L(...): returns

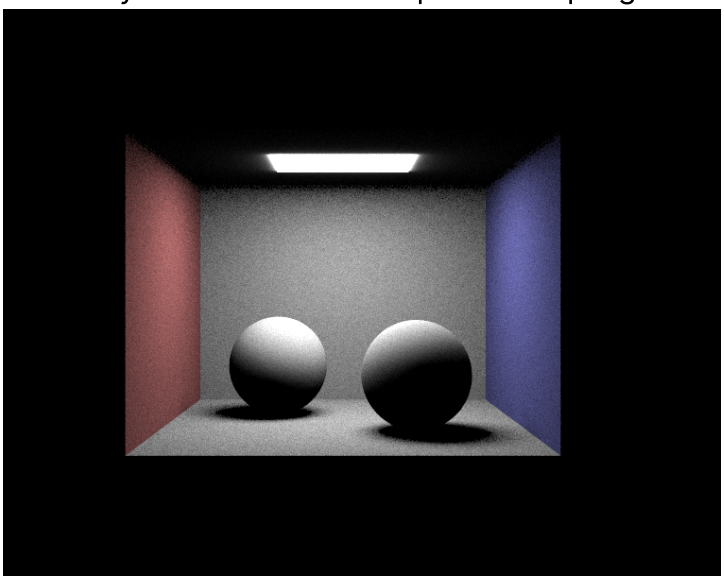
the radiance arriving from the sampled point on the light, the direction wi and distance to that point, and the PDF of having chosen that point on the light's surface. We again check whether something blocks the sampled direction to the light. If no intersection, add that light's contribution. The method to estimate the radiance is also monte carlo estimation. If direct_hemisphere_sample is true, we use uniform hemisphere sample method. Otherwise, we use light importance method.



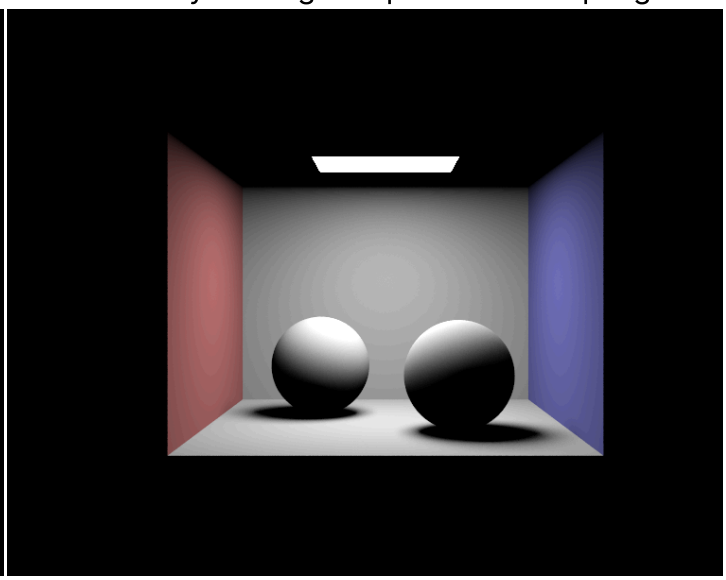
bunny with uniform hemisphere sampling



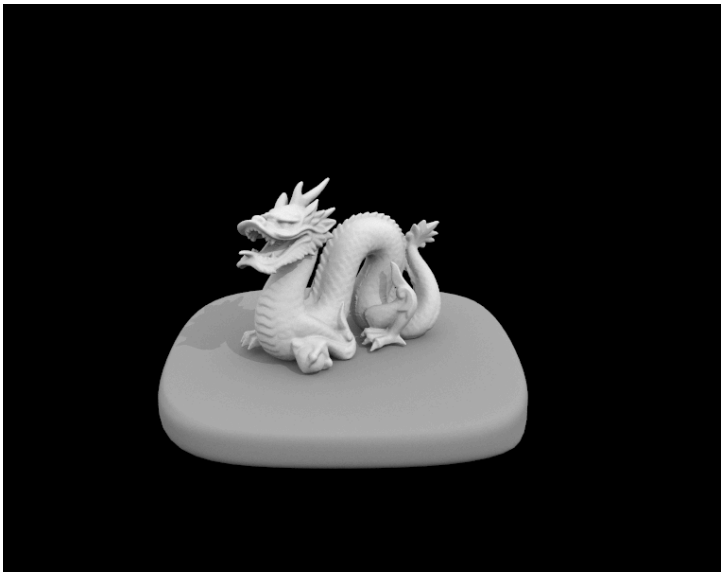
bunny with light importance sampling



sphere with uniform hemisphere sampling

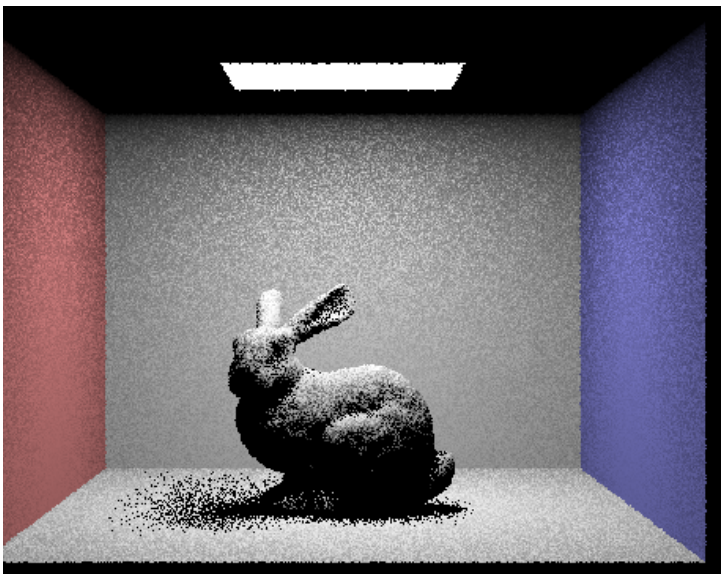


Sphere with light importance sampling

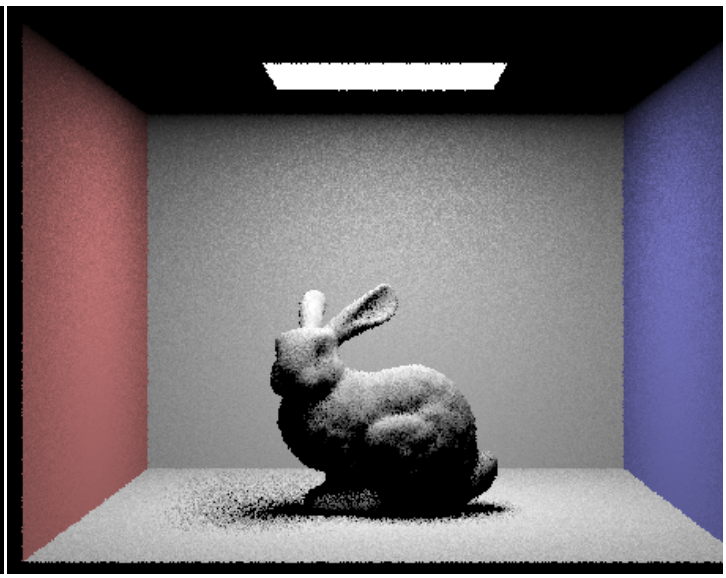


dragon with direct light

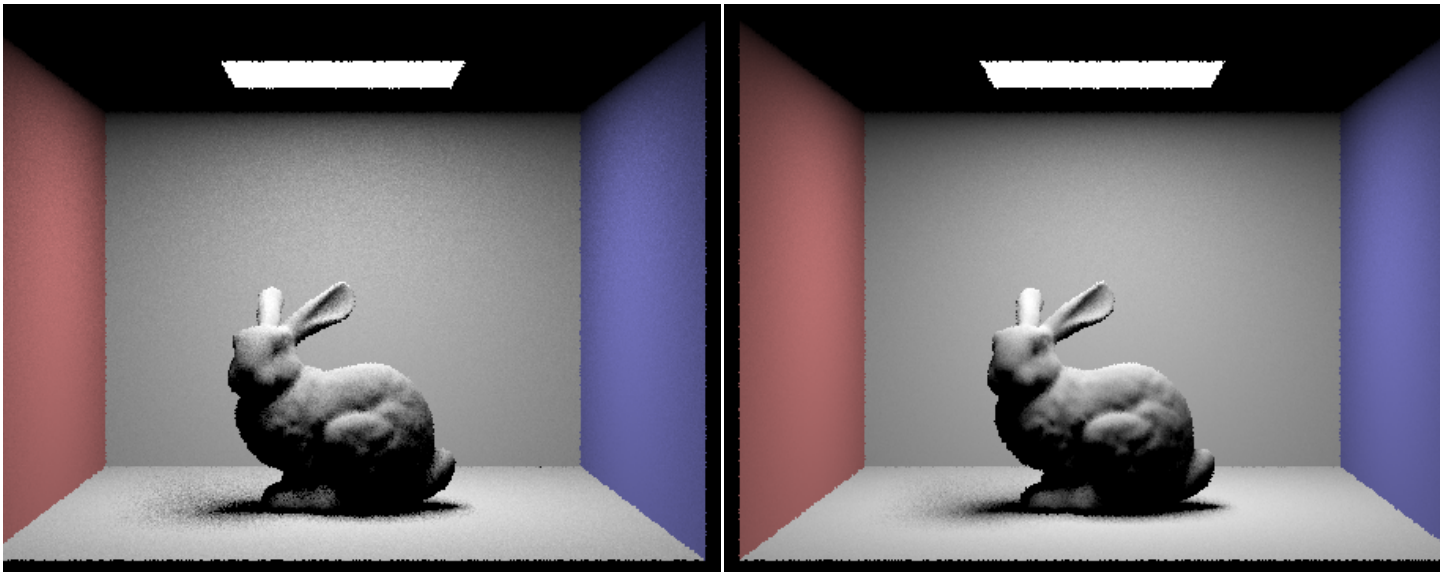
Different light rays with the 1 sample per pixel : At 1 light sample, the shadows have high variance—lots of flicker, show strong noise. At 4 light samples, the noise is reduced compared to 1 sample, though shadow edges can still show some grain. At 16 light samples, the shadows look much smoother and the noise is far less than before. Finally, at 64 light samples, the soft shadows are very clean, with any residual noise being subtle enough. It is not difficult to discover that the more rays we use for rendering, the less noise and smooth images we will get.



1 ray



4 rays



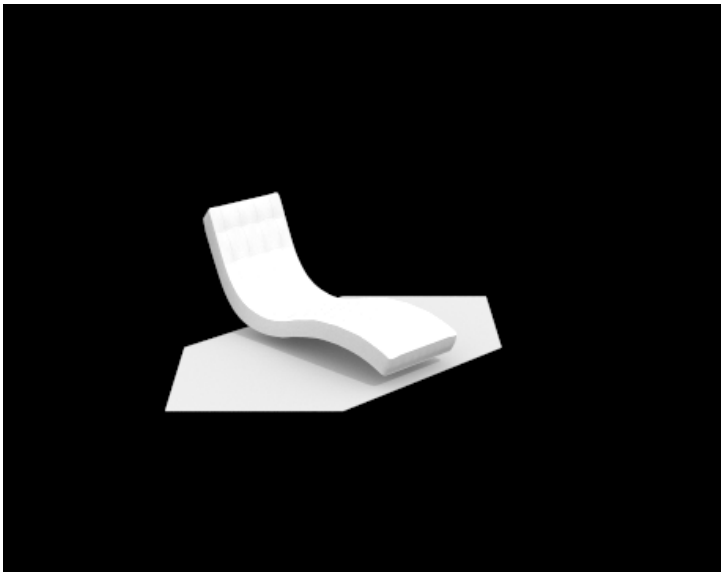
16 rays 64 rays

The comparison between uniform hemisphere sampling and light importance sampling reveals significant differences in noise levels and rendering quality. The image rendered using uniform hemisphere sampling, exhibits a high level of noise, especially in shadowed areas, as the sampling is distributed evenly over the hemisphere rather than focusing on the light source. This results in inefficient sampling where many rays contribute little to the final image. In contrast, the image rendered using light importance sampling, has significantly reduced noise and a much smoother appearance. This is because the sampling prioritizes directions toward the light source, leading to better convergence and more accurate shading.

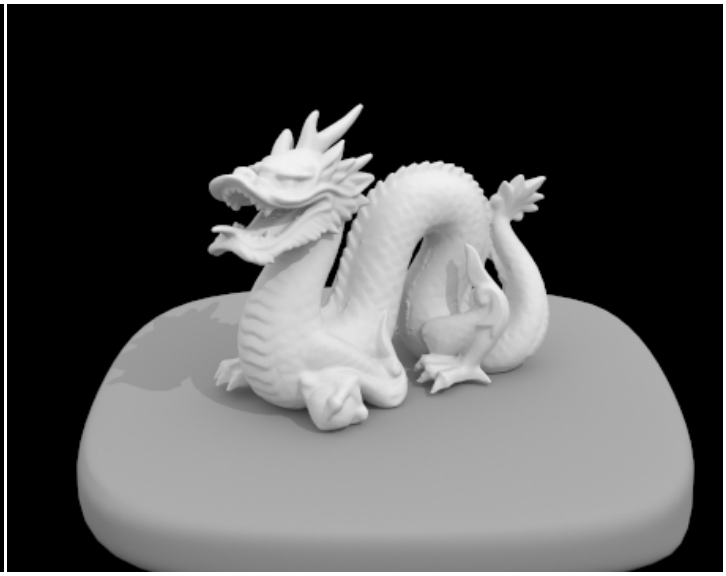
Part 4: Global Illumination

To estimate the global illumination, we should complete the function at_least_one_bounce. This function calculates indirect lighting (global illumination) using recursive ray tracing. At each intersection, it computes direct lighting (one_bounce_radiance) and recursively samples indirect lighting by tracing additional rays into the scene. For the base case, It dds direct lighting to output if either: It's the first bounce (r.depth == 1), or isAccumBounces is true (accumulating multiple bounce contributions explicitly). If an intersection occurs, recursively calls at_least_one_bounce_radiance to get indirect radiance from further bounces. add direct and indirect radiance together to get the final result.

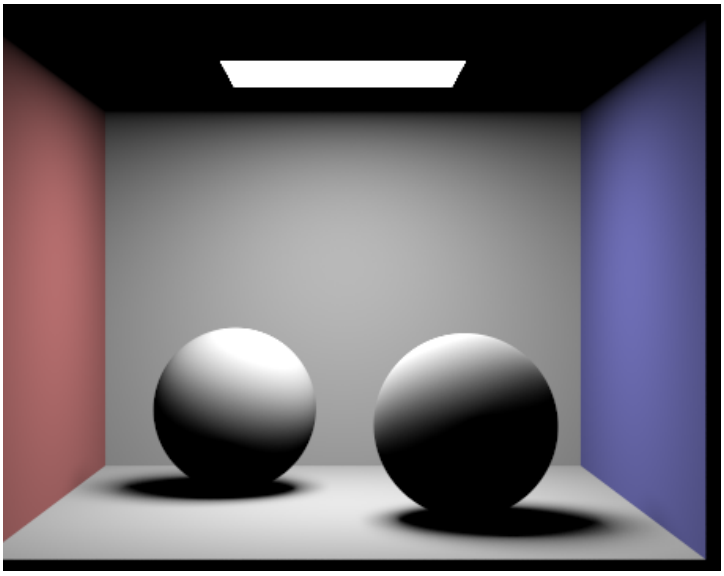
Russian Roulette implementation: we set a termination probability. The recursive step has 1 - termination probability to execute and the result of indirect light will be scaled with 1 - termination probability.



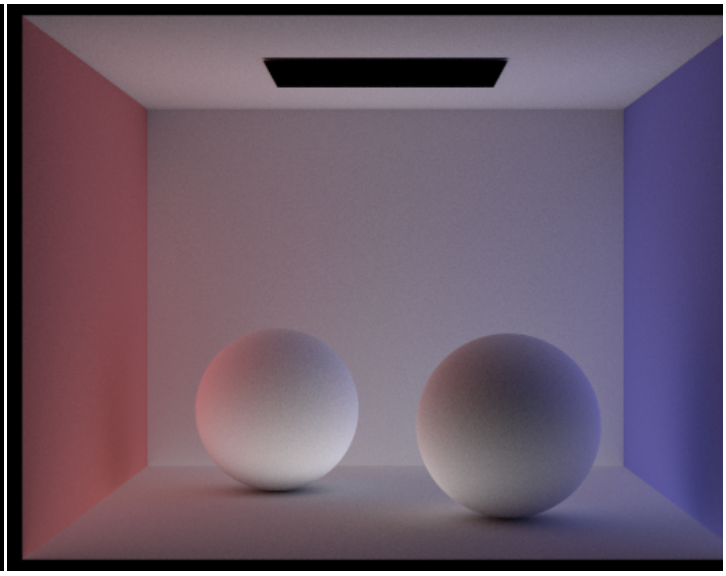
bench with global light 1024 samples per pixel



dragon with global light 1024 samples per pixel



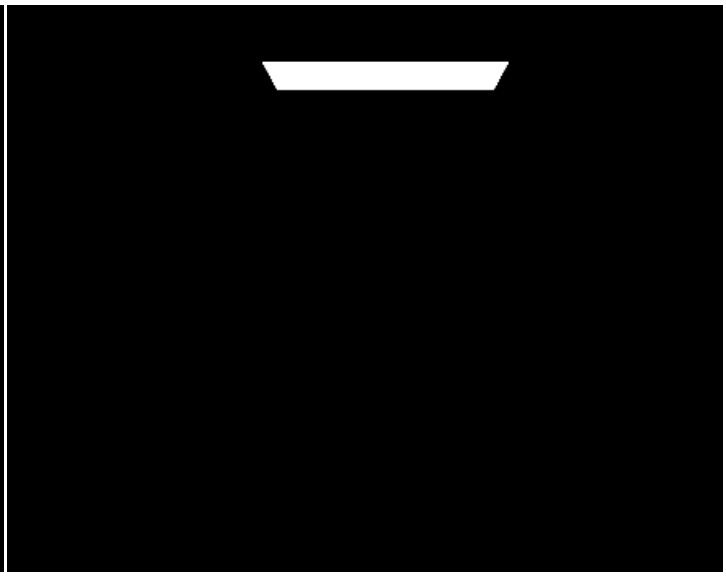
spheres with only direct light



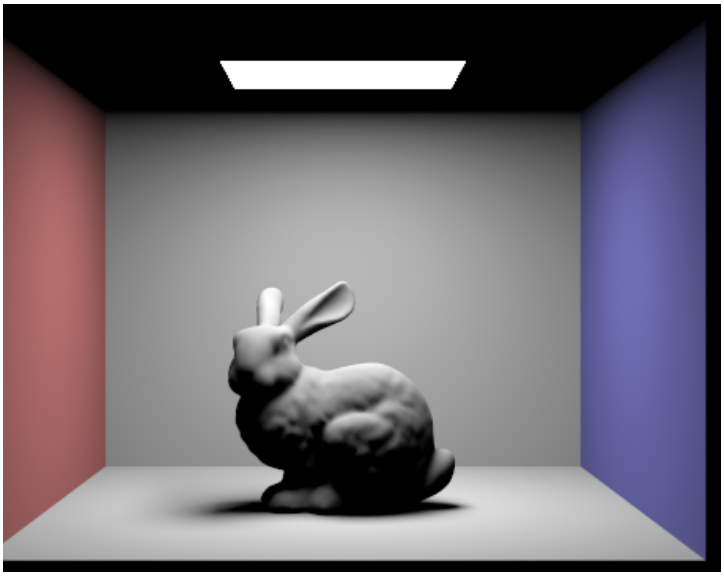
spheres with only indirect light

Above two images are images with only direct light and indirect light. The only direct light images has well-defined shadows, strong lightning, and no color bleeding. The indirect version has no direct shadows and significant color bleeding from the red and blue walls onto the spheres and the floor.

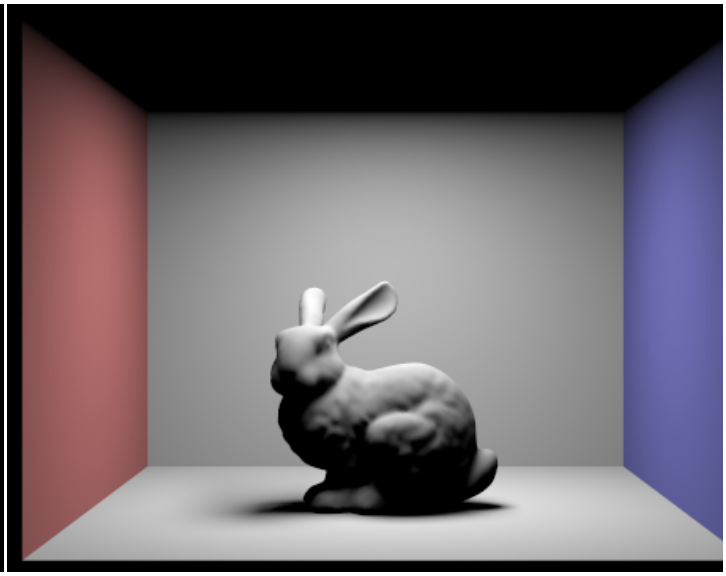
Accumulated, m = 0



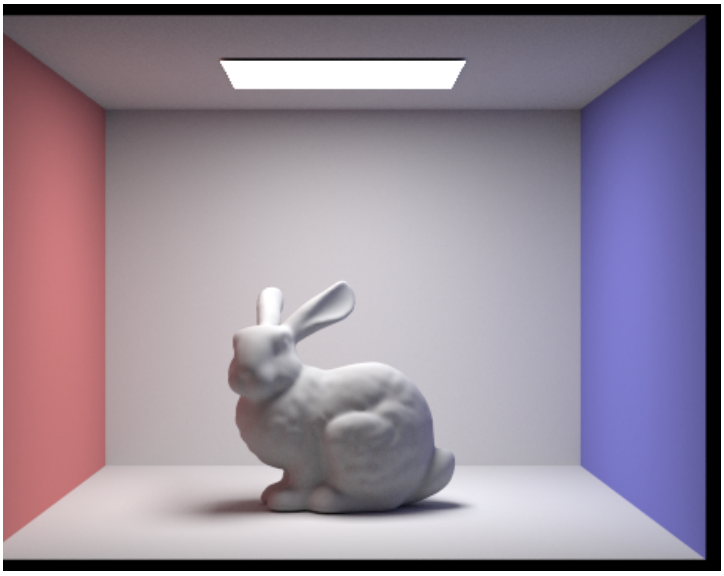
Unaccumulated, m = 0



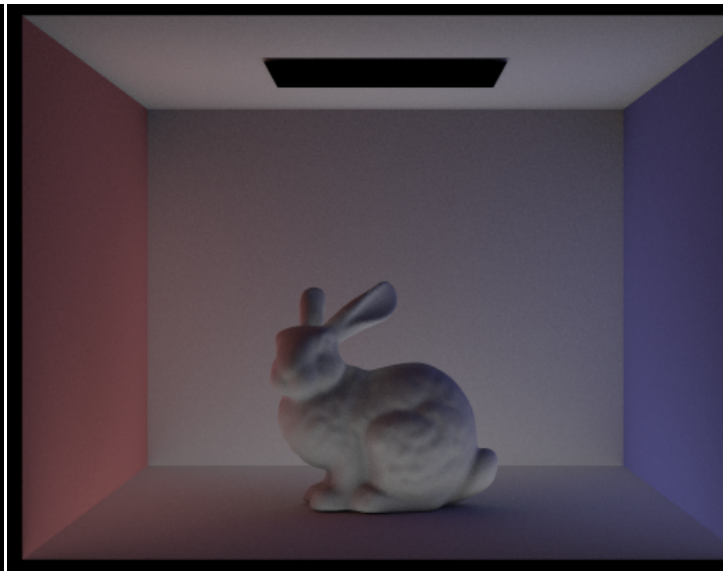
Accumulated, m = 1



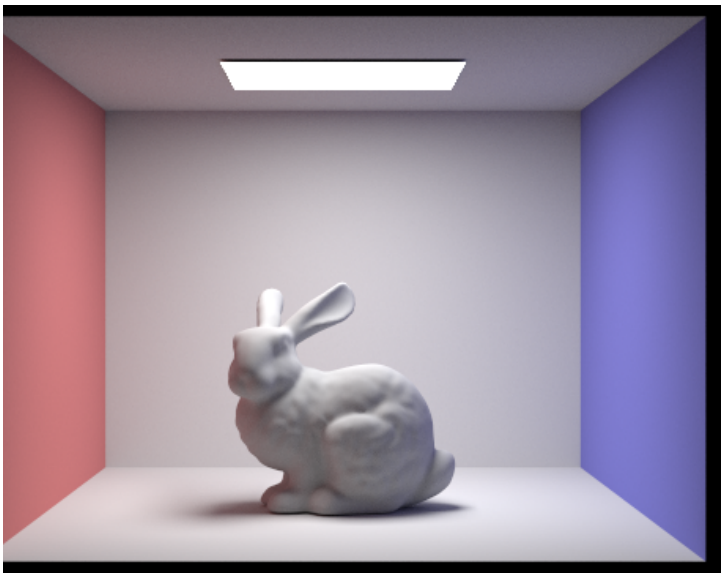
Unaccumulated, m = 1



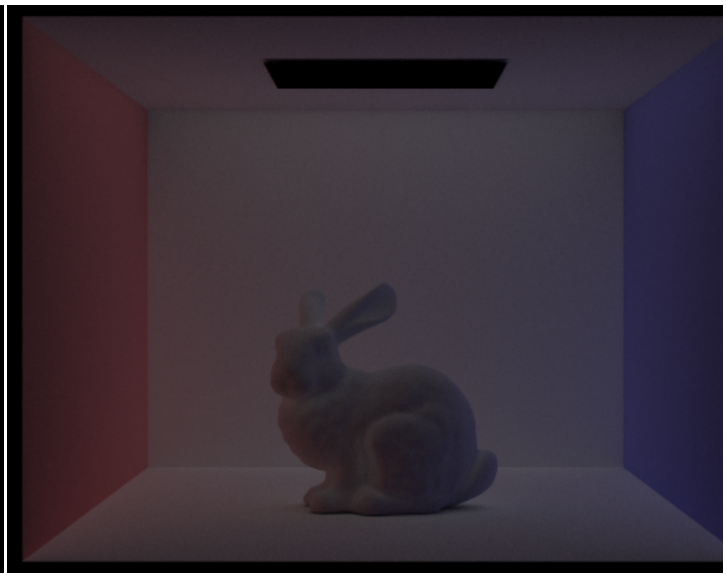
Accumulated, m = 2



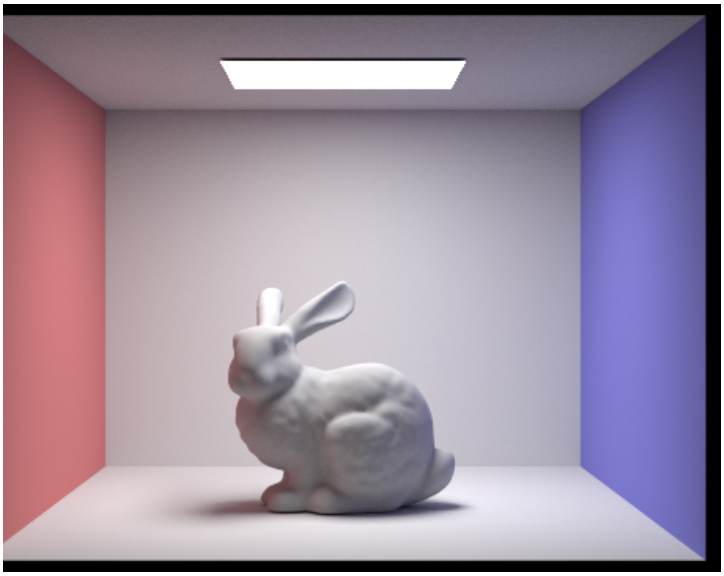
Unaccumulated, m = 2



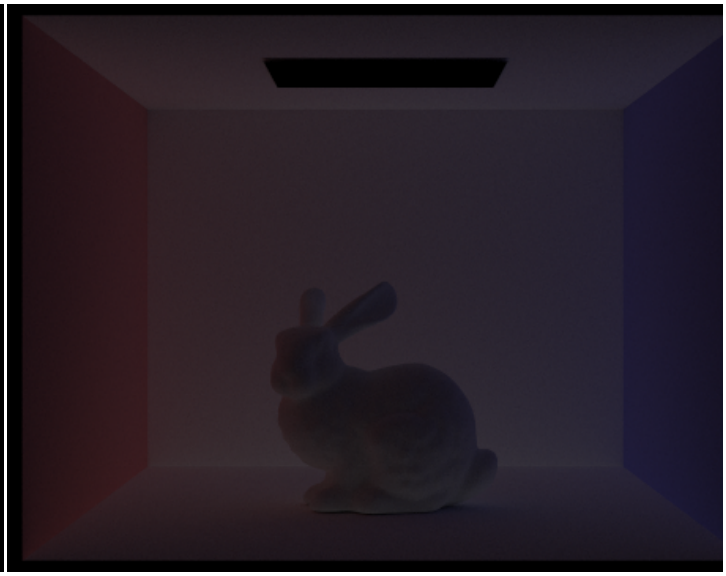
Accumulated, m = 3



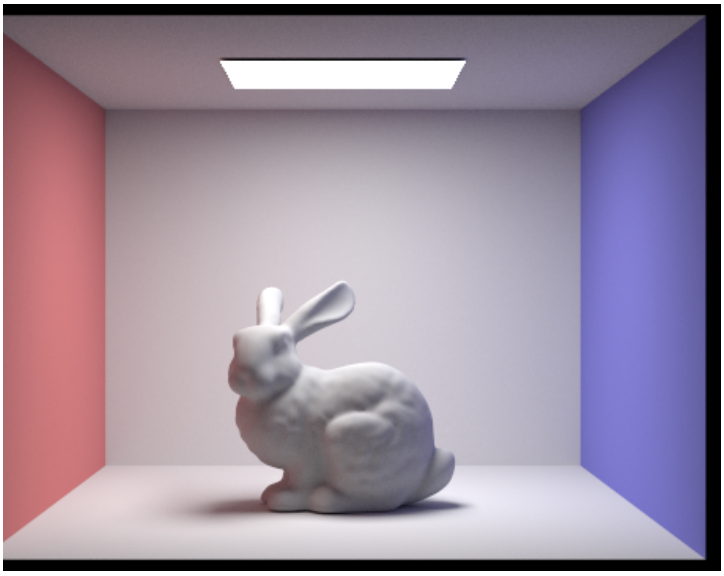
Unaccumulated, m = 3



Accumulated, m = 4



Unaccumulated, m = 4



Accumulated, m = 5

Unaccumulated, m = 5

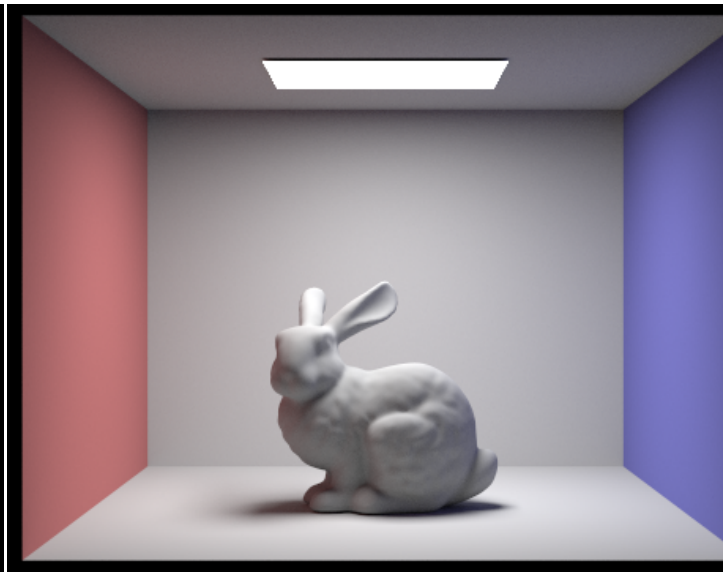
At the second bounce, the light from ceiling reflects off the floor and walls. The red and blue walls cause the subtle color breeding on bunny. For the third bounce, the brightness decreases and the light becomes more soft. More color bleeding occurs onto the bunny to make it more colorful.

Raterization majorly relies on direct illumination. It could not utilize indirect illumination to make the color breeding occur and the atmosphere of image to become more real and soft.

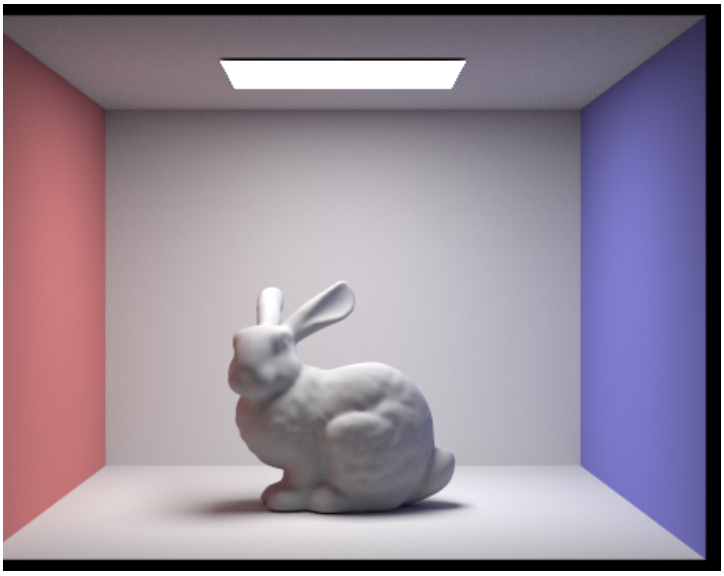
As for comparision of accumulated and unaccumulated bunny, the accumulated bunny become much more brighter and more realistic with the increase of max_ray_depth. The unaccumulated bunny become much more darker. Although more color bleeding occurs, it is too dark to observe that.

Russian Roulette bunny:

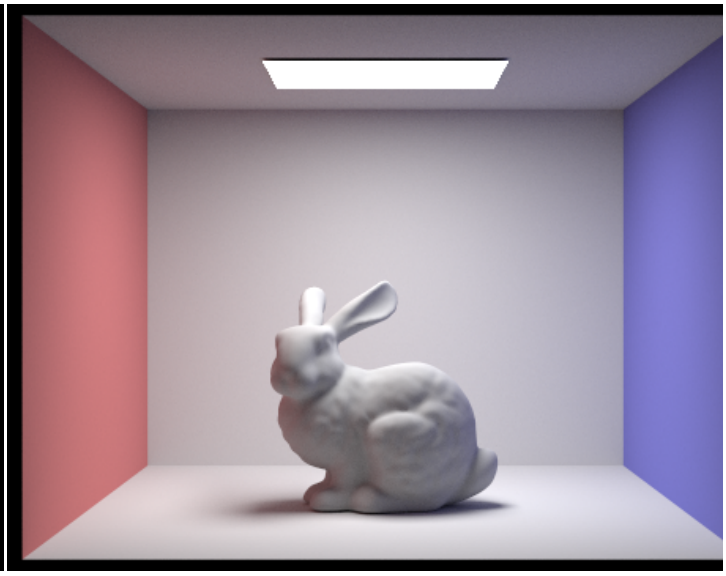
Russian Roulette with m = 0



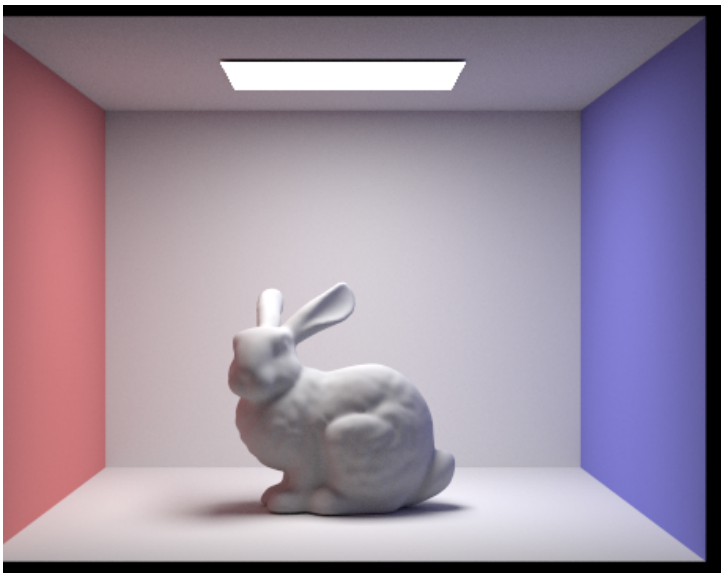
Russian Roulette with m = 1



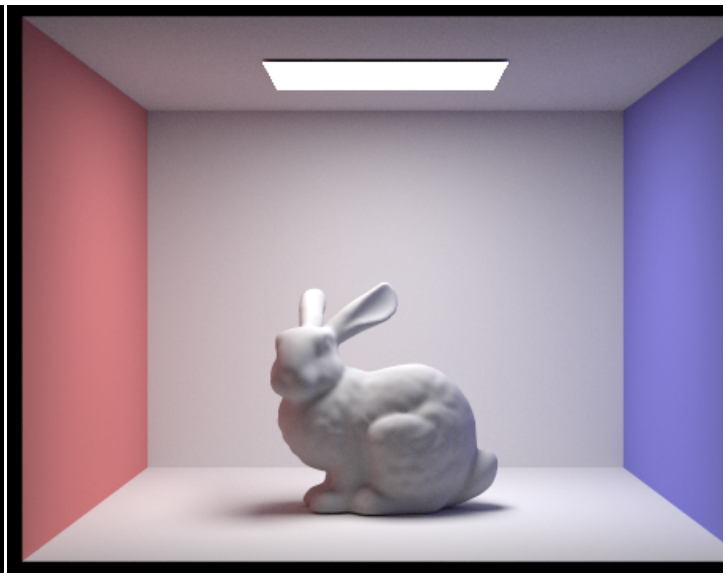
Russian Roulette with m = 2



Russian Roulette with m = 3

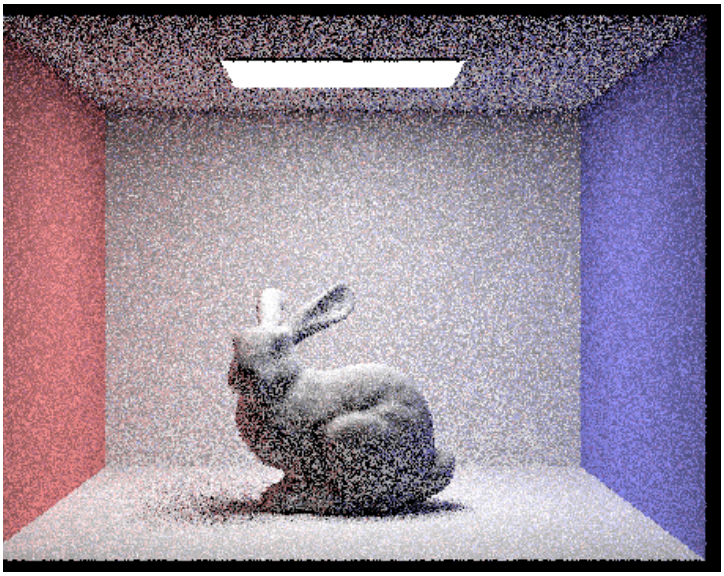


Russian Roulette with m = 4

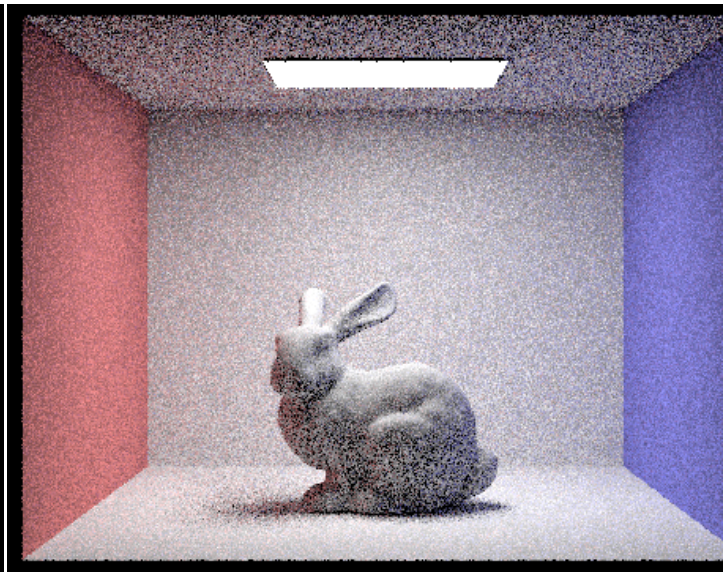


Russian Roulette with m = 100

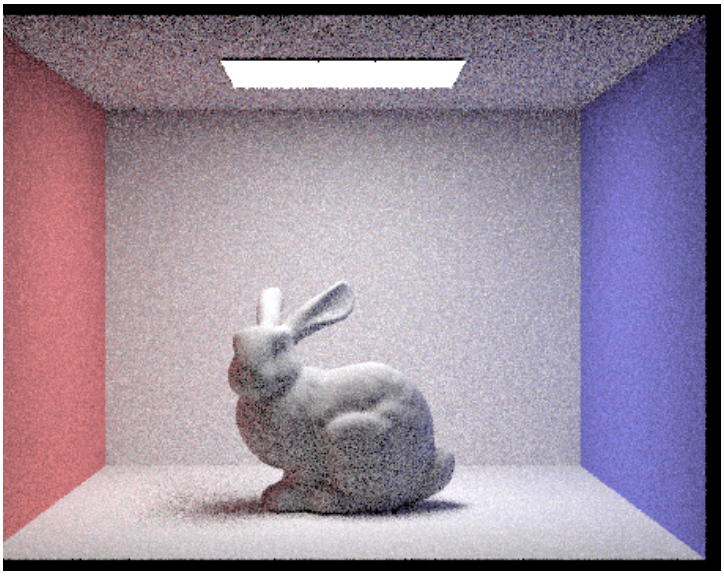
bunny with different sample rate.



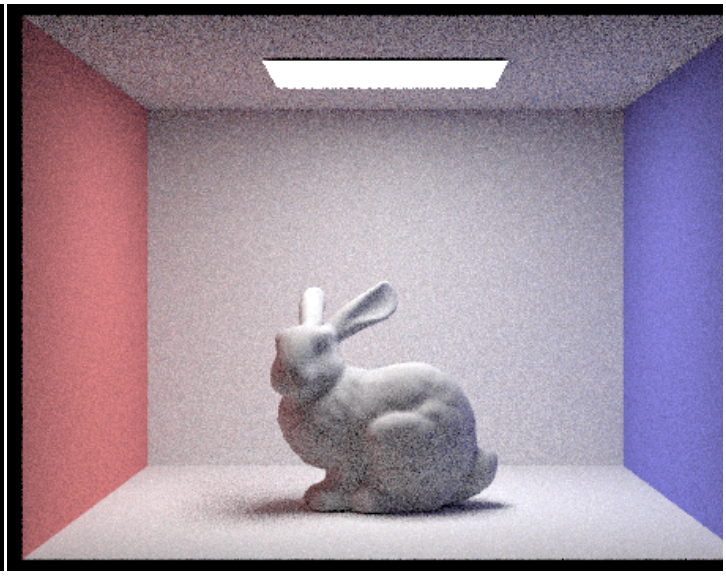
4 light rays with 1 sample rate



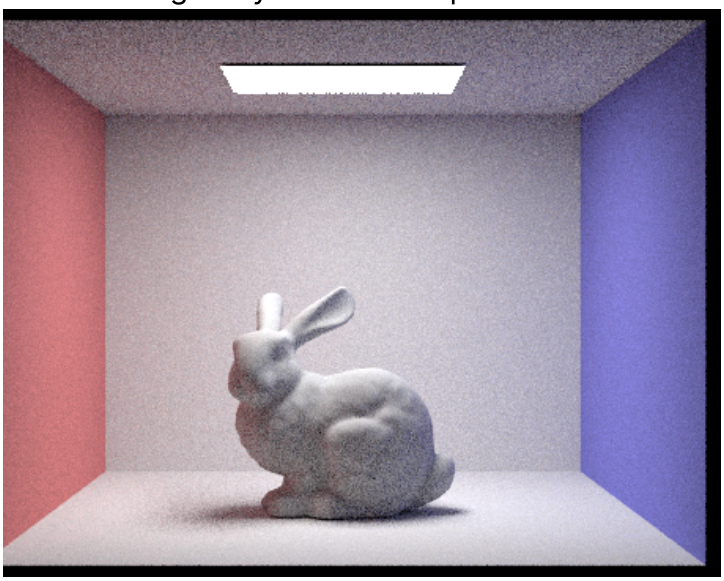
4 light rays with 2 sample rate



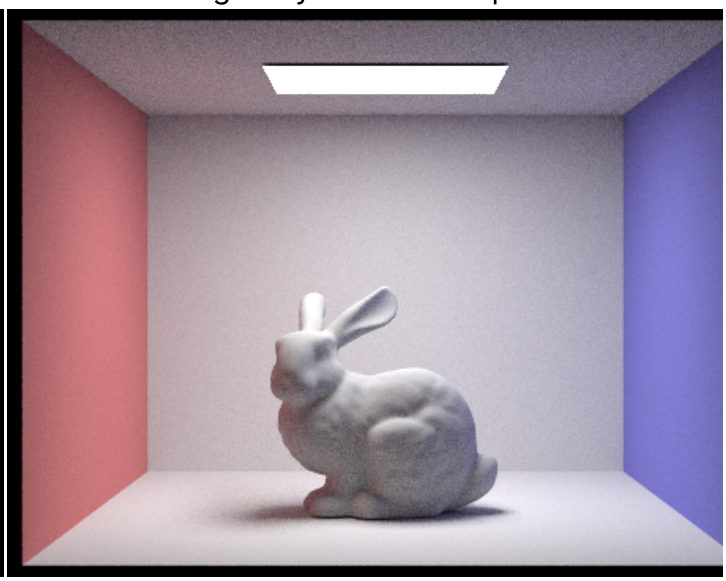
4 light rays with 4 sample rate



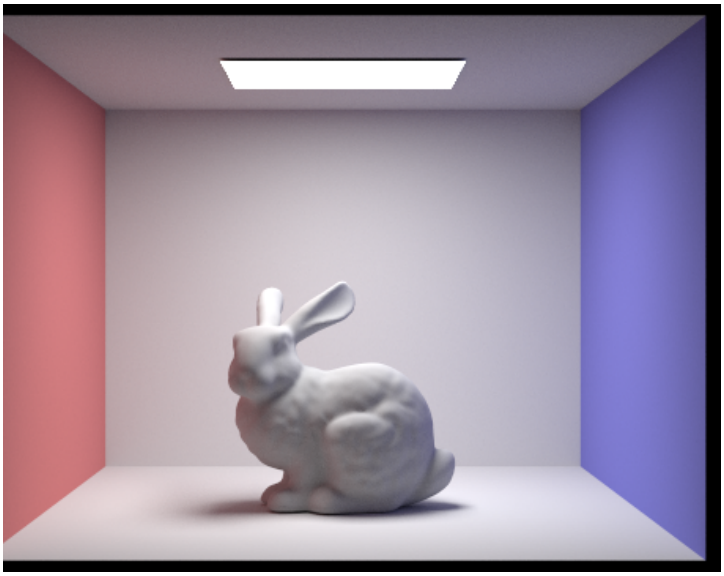
4 light rays with 8 sample rate



4 light rays with 16 sample rate



4 light rays with 64 sample rate



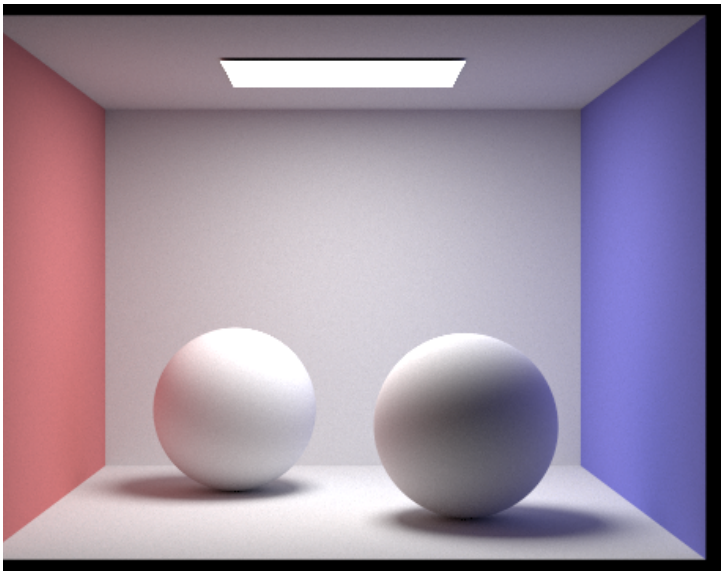
4 light rays with 1024 sample rate

At low sampling rates, the image exhibits heavy grain and noise. At high sampling rate, the noise is virtually eliminated, and the image achieves a realistic quality with soft shadows and well-distributed indirect illumination.

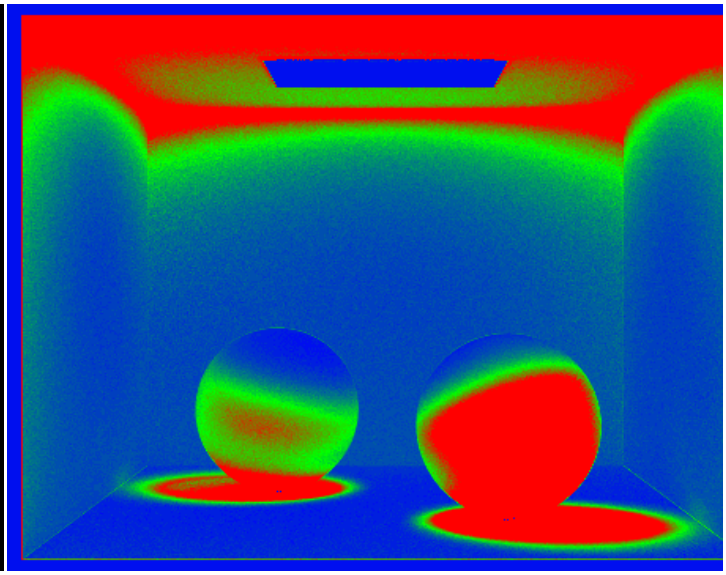
Part 5: Adaptive Sampling

Adaptive sampling is a technique in path tracing used to optimize rendering by dynamically adjusting the number of samples per pixel based on variance. Instead of using a fixed number of samples for every pixel, adaptive sampling evaluates the noise level of a pixel and decides whether additional samples are necessary.

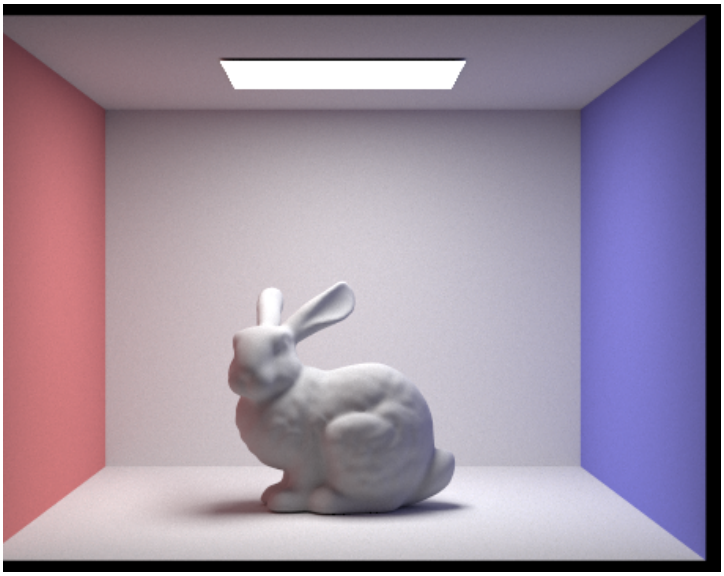
Implementation: I add new variables s1, s2, var. The illumination value of the estimated radiance is extracted and accumulated into s1 and s2 at the end of each loop. Every samplesPerBatch iterations, the variance is computed and The stopping condition checks whether the confidence interval is within maxTolerance of the current mean. If this condition is met, the loop terminates early, saving computation.

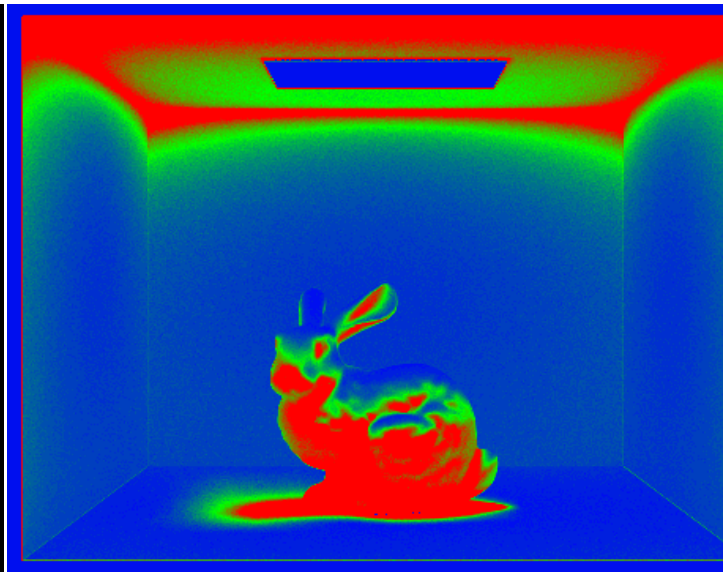


pheres 2048 sample rate with adaptive sampling



spheres'sample rate image

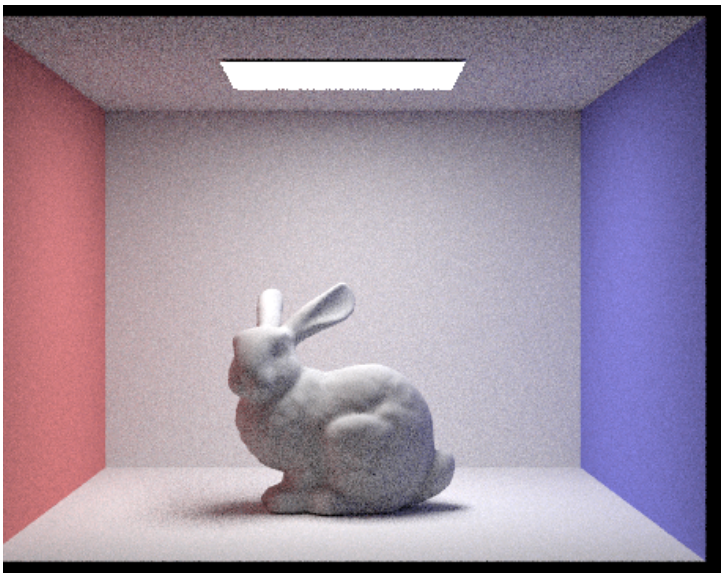

bunny 2048 sample rate with adaptive sampling

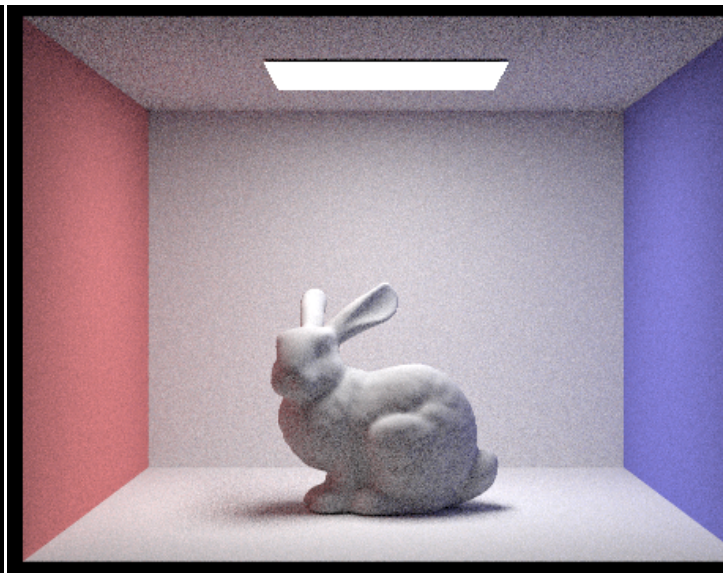

bunny's sample rate

(Optional) Part 6: Extra Credit Opportunities

Challenge level 1: jittered sampling.

Choose a number of samples that is a perfect square, call this n2 Divide each pixel into an n x n grid.Randomly place a single sample within each cell of this grid. Each pixel is said to be stratified, and the domain is the n x n grid. Each cell is called a stratum, where strata are independent , but together cover the entire domain. To implement this, I set new variables called j = sqrt(num_samples). Then I divide the random sample with j. Next steps is to calculate the grids position. I use the floor of i divided by j as the numerator and j as the denomenator to represent grids' y position, then I add 1 mod 8 to x_position numerator. This ensures that every sample will be distributed in nxn grids and each grids have one.


4 rays, 16 sample rate with random sampling


4 rays, 16 sample rate with jittered sampling

The left image (random sampling) exhibits a more uneven noise pattern, with some areas appearing grainier. The

right image (jittered sampling) has a more uniform noise distribution, reducing the variance in the noise and making the overall rendering appear smoother. As the evidence, you can see the lights of these two images. The left image's light has uneven noisy edge, but the right image's light is smoother.