CSC4140 Final Project

Computer Graphics

May 23, 2022

Bi-Directional Path Tracer

Student ID: 118010335

Student Name: Wei WU

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

Signature:

1 Overview

In Assignment 6 and 7, we implemented path tracing, allowing us to render photo-realistic images. However, path tracing is not perfect. One significant shortcoming is that when a part of a scene is mainly illuminated by indirect illumination, the result is very dark and noisy. The reason is that in those regions, the paths traced have a small probability to hit the directly illuminated part (i.e. the indirect light sources).

To deal with that, we implemented BDPT (Bidirectional Path Tracing) by shooting half-paths from both the camera and the lights, then connecting them somewhere in between.

2 Implementation

2.1 New Command Line Argument

A new command line argument -w is added. This integer argument specifies the maximum depth of the backward half-paths. When -w flag is not given, the maximum depth is set to the default value -1. When the value of the argument is smaller than or equal to 0, there is no backward path tracing. However, when the argument is strictly smaller than zero, the original path tracing pipeline is used; when the argument is equal to 0, the new bidirectional path tracing pipeline is used. The choice of pipeline should have no effect on the result. This implementation is just for debugging.

Example:

./pathtracer -t 6 -s 16 -l 16 -m 5 -w 5 -r 480 360 ../dae/sky/CBspheres_bdpt.dae -f CB-spheres_bdpt_16_16_5_5.png

2.2 The PathTracer Class

Vector3D estimate_lighting_bdpt(const Ray &r, const SceneObjects::Intersection &isect);

This function estimates the illumination from the backward-traced paths on a particular point specified by a ray and a corresponding intersection.

Vector3D one_bounce_radiance_bdpt(const Ray &r, const SceneObjects::Intersection &isect);

This function estimates the one-bounce radiance on a point in the same way as one_bounce_radiance().

Vector3D at_least_one_bounce_radiance_bdpt(const Ray &r, const SceneObjects::Intersection &isect);

This function estimates the one to n-bounce radiance on a point. There are two ways of estimating indirect illumination. The first is sampling a ray, which is the same as at_least_one_bounce_radiance(). The second is connecting the point to the backward-traced paths and calculating the radiance. We use these two ways to get two different radiance, and then pick the greater one.

Vector3D est_radiance_global_illumination_bdpt(const Ray &r);

This function estimates the global radiance on a point in the same way as est_radiance_global_illumination().

void raytrace_light(const SceneObjects::SceneLight& light, int num_light_samples);

This function shoots a certain number of rays from a light. If a ray has an intersection with the scene, the function creates a BdptPoint structure and adds it into the bdptBuffer.

void raytrace_bdpt_point(const BdptPoint& bdptPoint);

This function shoots one ray from a bdptPoint. If this ray has an intersection with the scene, the function creates another BdptPoint structure and adds it into the bdptBuffer.

void raytrace_pixel_bdpt(size_t x, size_t y);

This function shoots a ray from the camera to a pixel.

Vector3D connect(const Vector3D& pos, const Ray& r, BSDF* bsdf, const BdptPoint& bdptPoint);

This function connects a point on the forward sub-path and a point on the backward sub-path and calculates the output radiance.

2.3 The SceneLight Class

virtual Vector3D sample_L_backward(const Vector3D d, const BVHAccel* bvh, Vector3D* sample, Vector3D* hit_p, double* distToLight, double* pdf, BSDF** bsdf, Vector3D* n) const = 0;

This function shoots a ray from a light. If the ray hits the scene, it updates the sample, pdf, hit_p, distToLight, bsdf, and n pointers and returns the radiance.

2.4 The RaytracedRenderer Class

void start_raytracing();

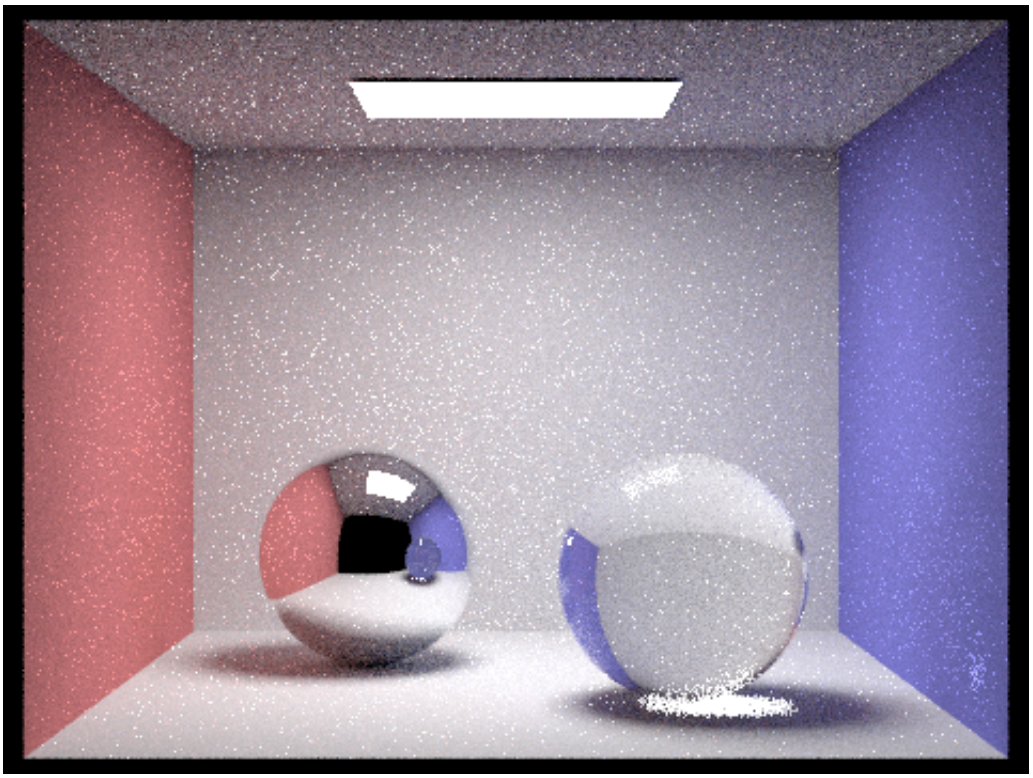
If BDPT is enabled, trace the backward sub-paths before tracing the forward sub-paths.

void raytrace_tile(int tile_x, int tile_y, int tile_w, int tile_h);

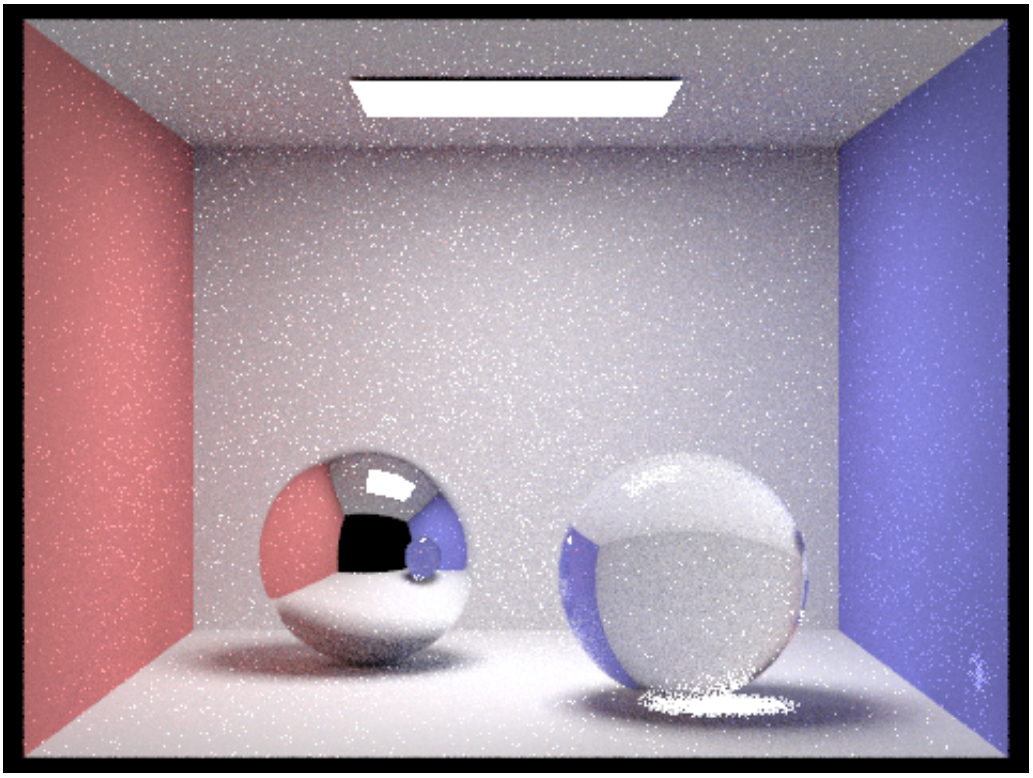
If BDPT is enabled, call pt->raytrace_pixel_bdpt(x, y) to use the BDPT pipeline.

3 Result - The Original CBspheres Scene

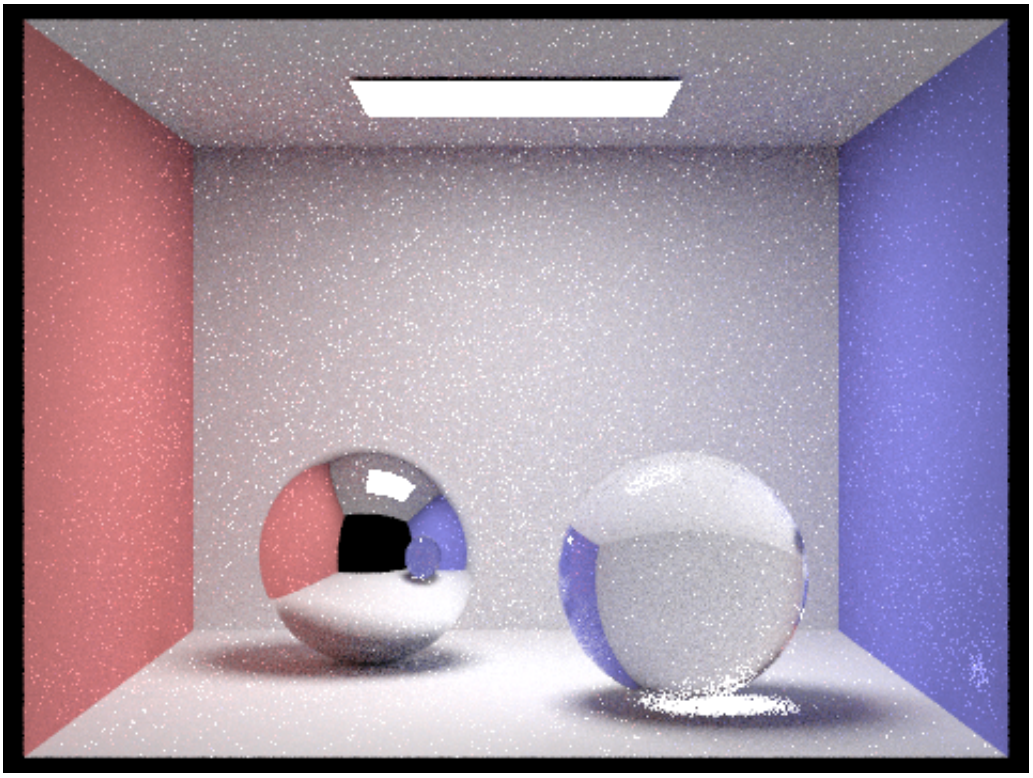
3.1 -s 16 -l 16 -m 5 -w 0



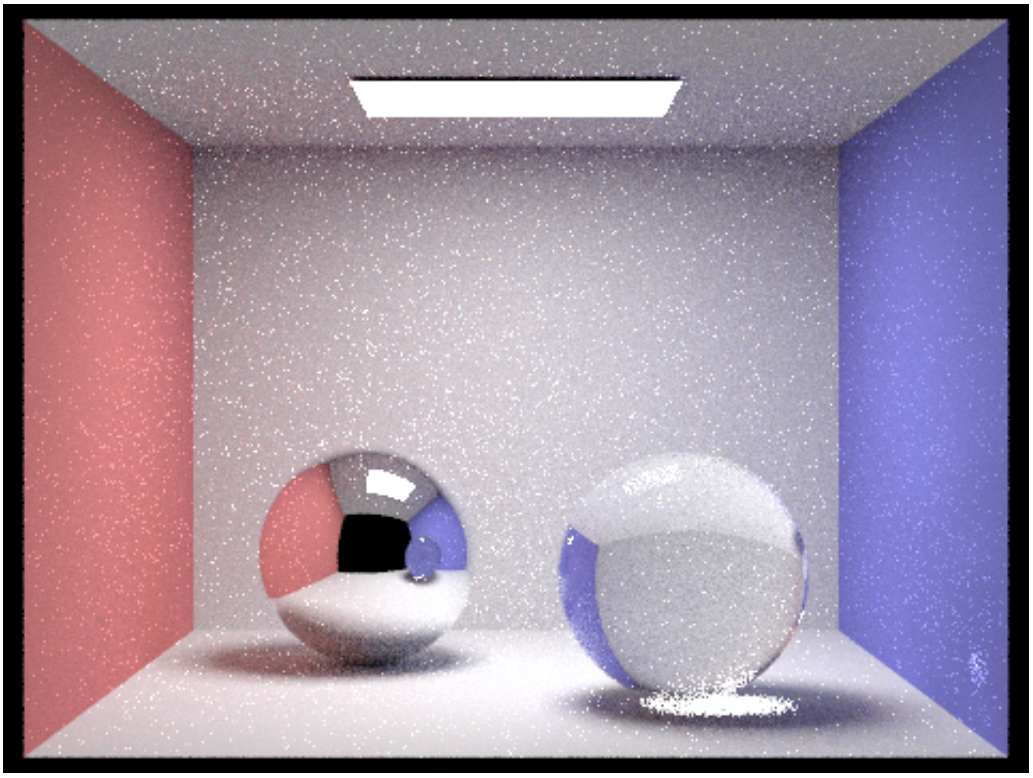
3.2 -s 16 -l 16 -m 5 -w 1



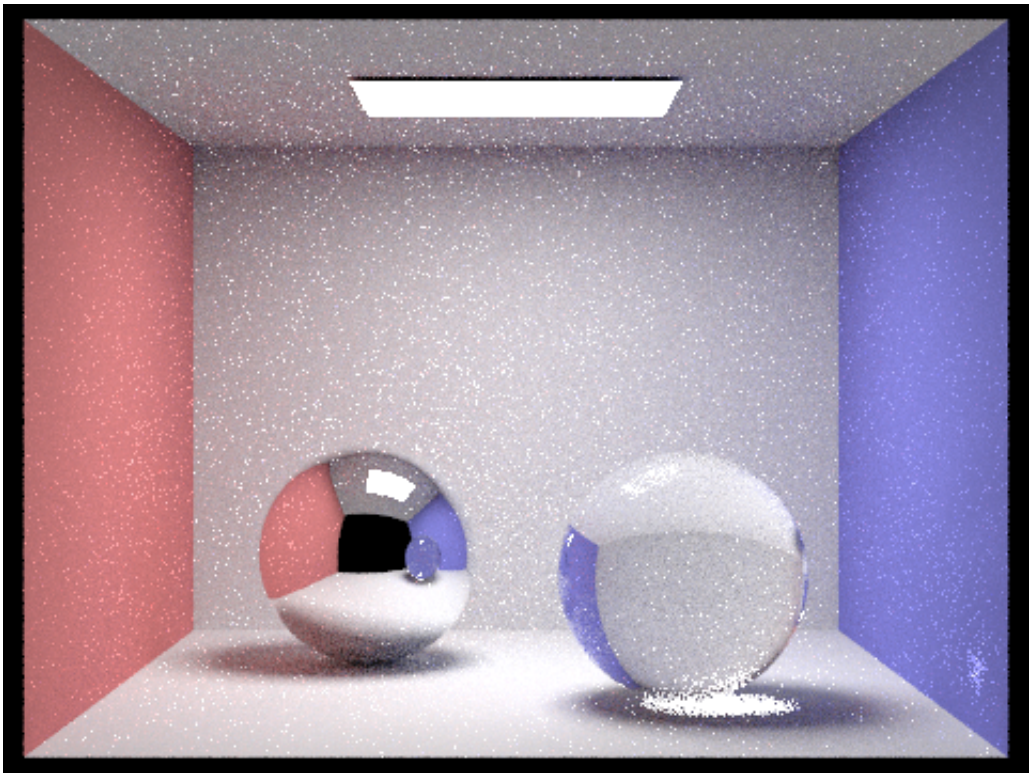
3.3 -s 16 -l 16 -m 5 -w 2



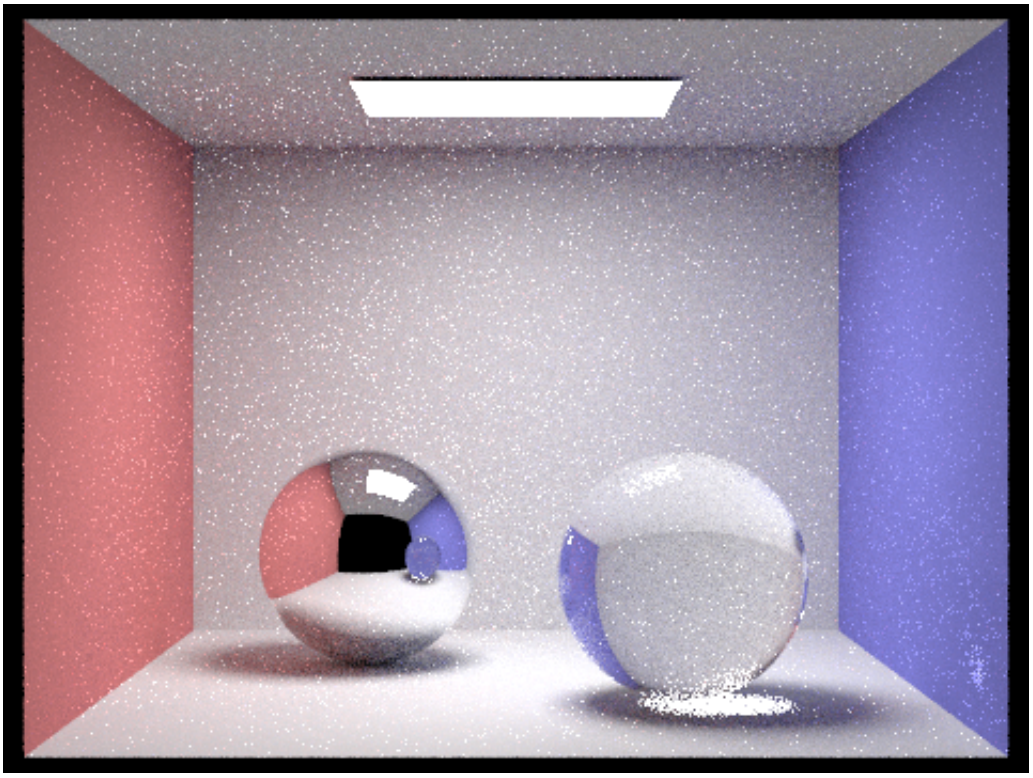
3.4 -s 16 -l 16 -m 5 -w 3



3.5 -s 16 -l 16 -m 5 -w 4



3.6 -s 16 -l 16 -m 5 -w 5

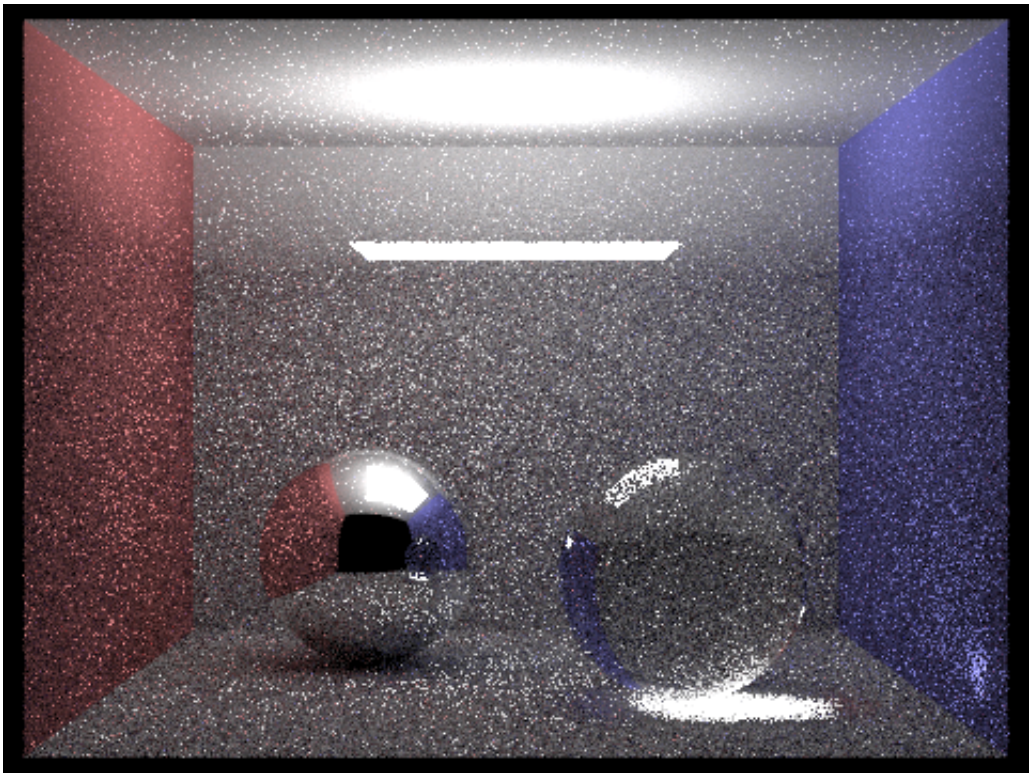


Clearly, on the original CBspheres scene, when BDPT is enabled, the result is much brighter and less noisier.

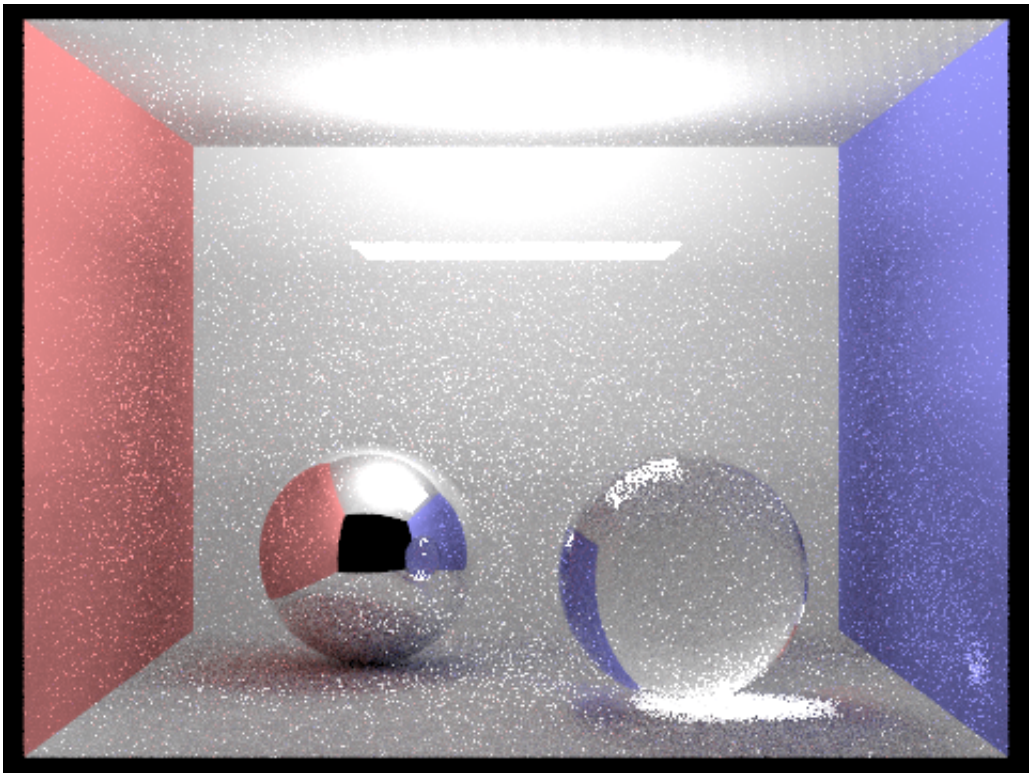
4 Result - The Modified CBspheres Scene

To better demonstrate the effect of bidirectional path tracing, we modified the original CBspheres scene by lowering and inverting the area light.

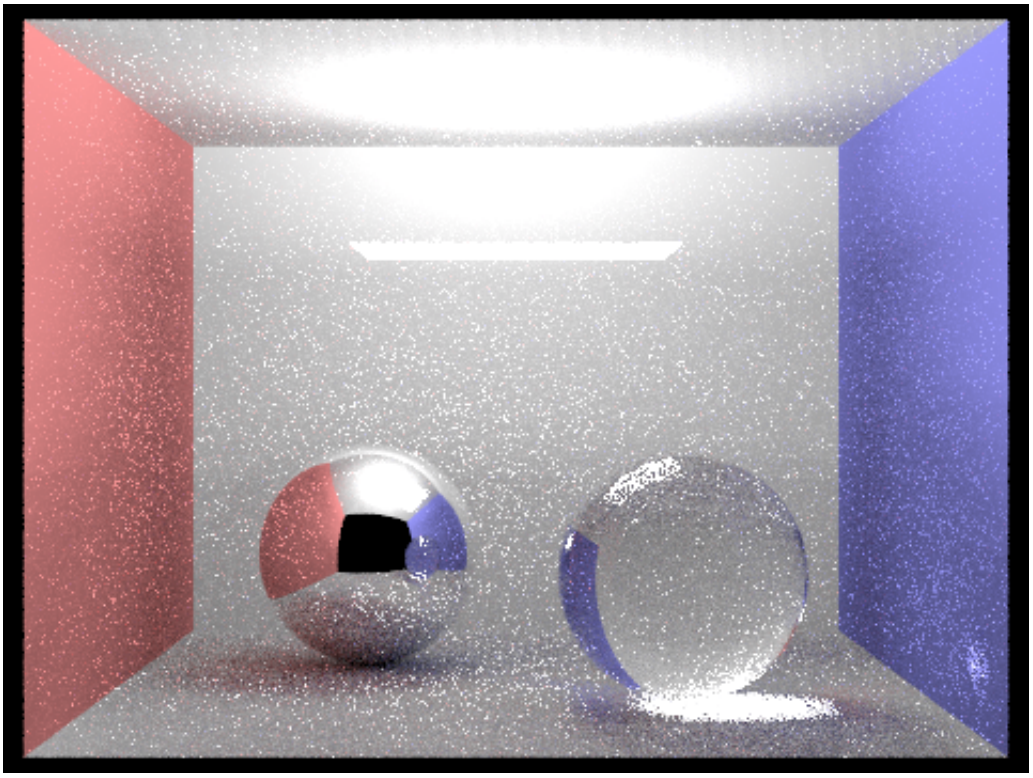
4.1 -s 16 -l 16 -m 5 -w 0



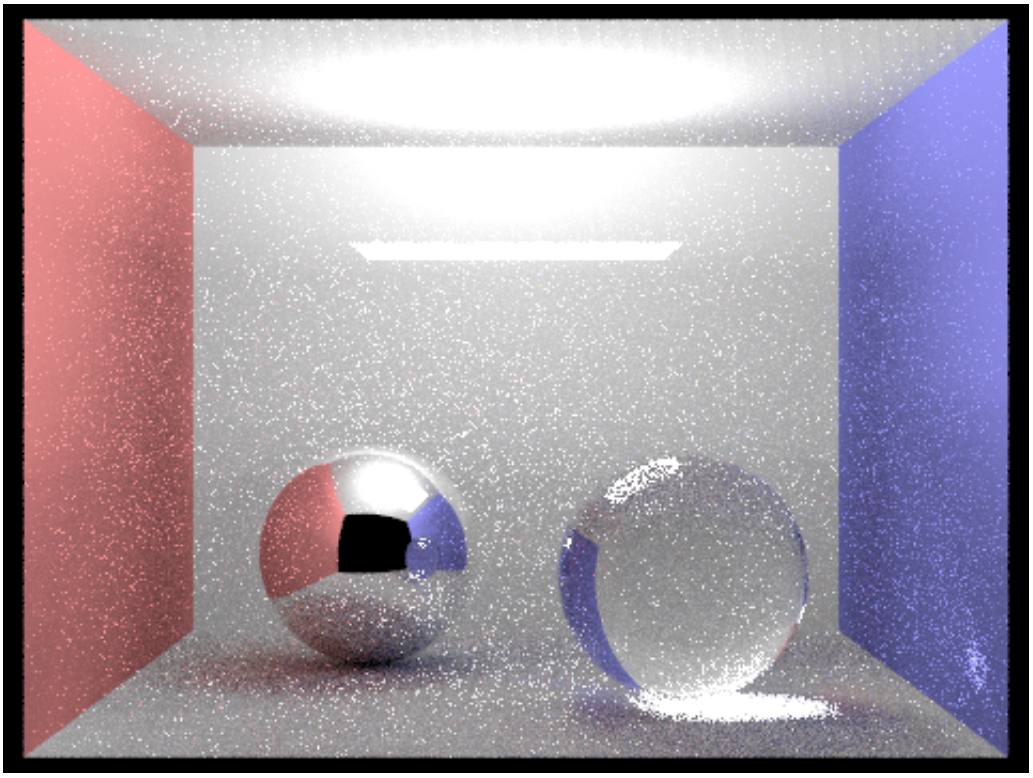
4.2 -s 16 -l 16 -m 5 -w 1



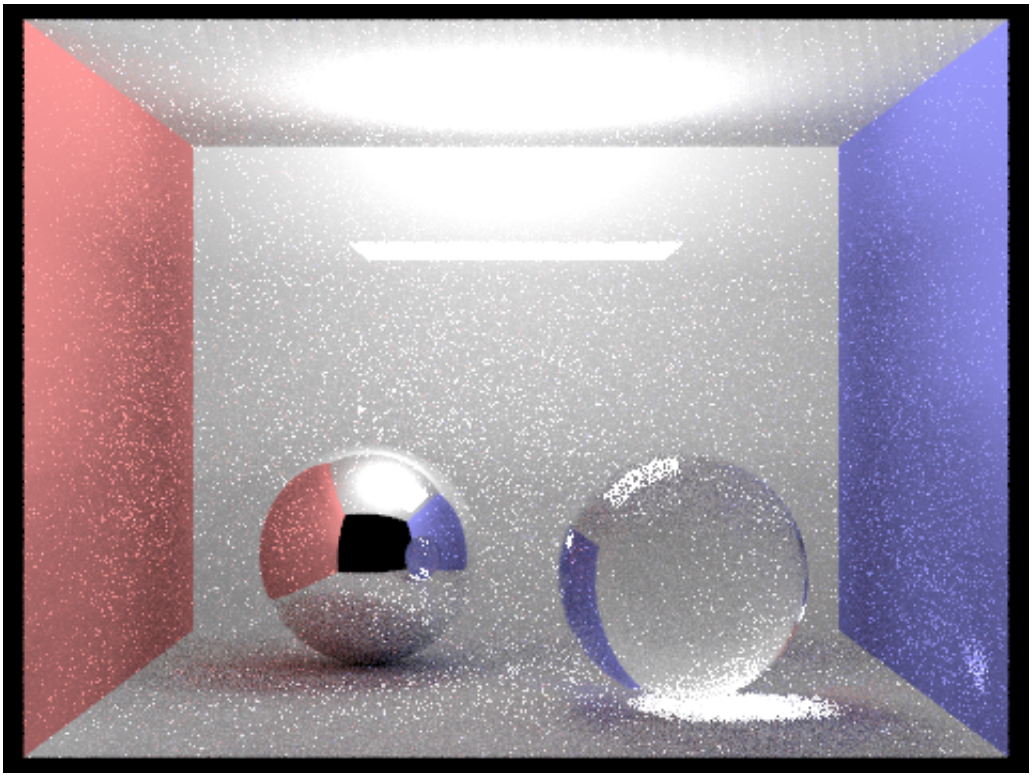
4.3 -s 16 -l 16 -m 5 -w 2



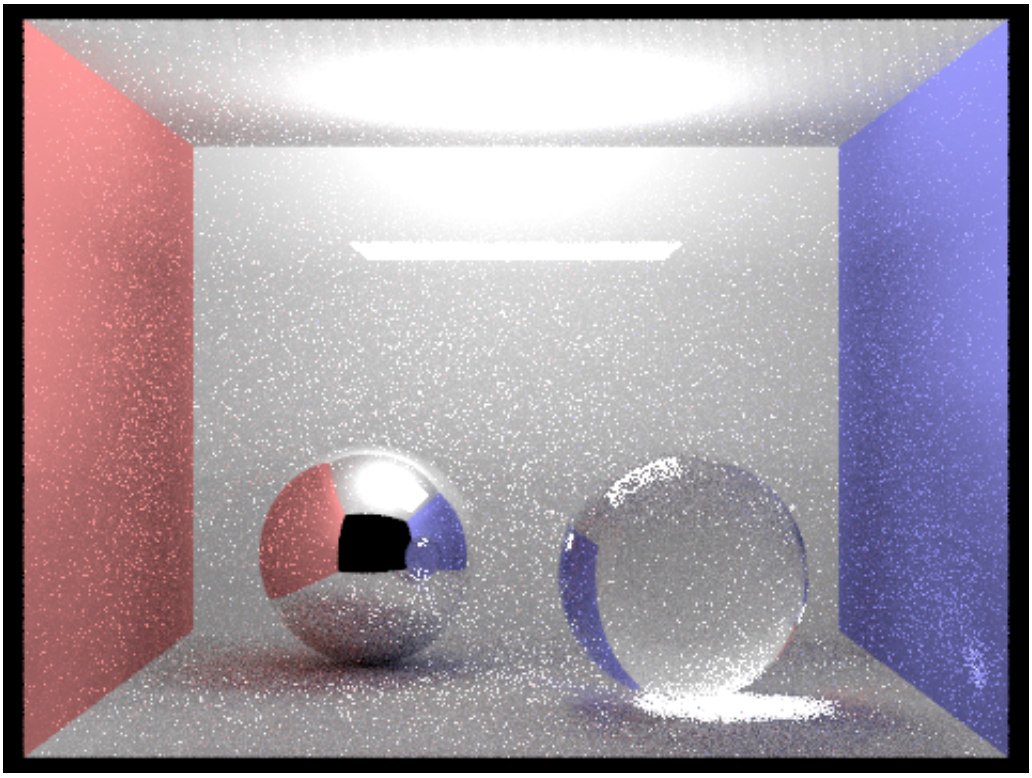
4.4 -s 16 -l 16 -m 5 -w 3



4.5 -s 16 -l 16 -m 5 -w 4



4.6 -s 16 -l 16 -m 5 -w 5



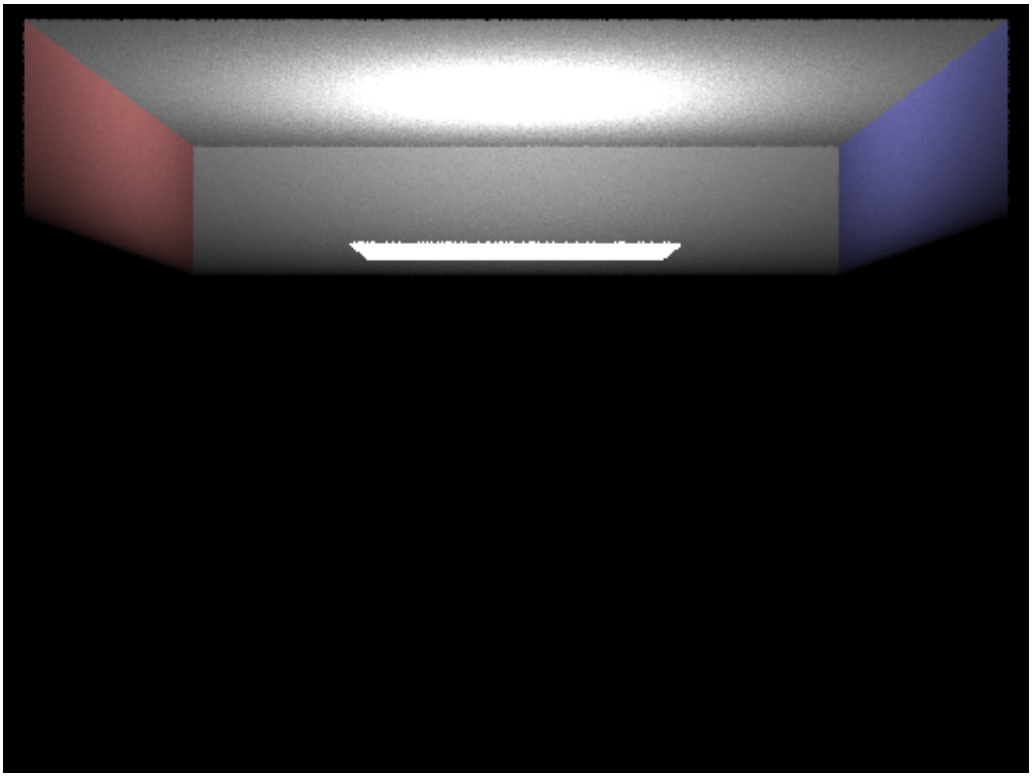
When BDPT is disabled, the lower part of the scene is very dark and noisy. When BDPT is turned on, the result becomes much brighter and less noisier. Note that the upper part of the scene also becomes brighter.

5 Result - The Modified CBspheres_lambertian Scene

To better demonstrate the effect of bidirectional path tracing, we modified the original CB-spheres_lambertian scene by lowering and inverting the area light.

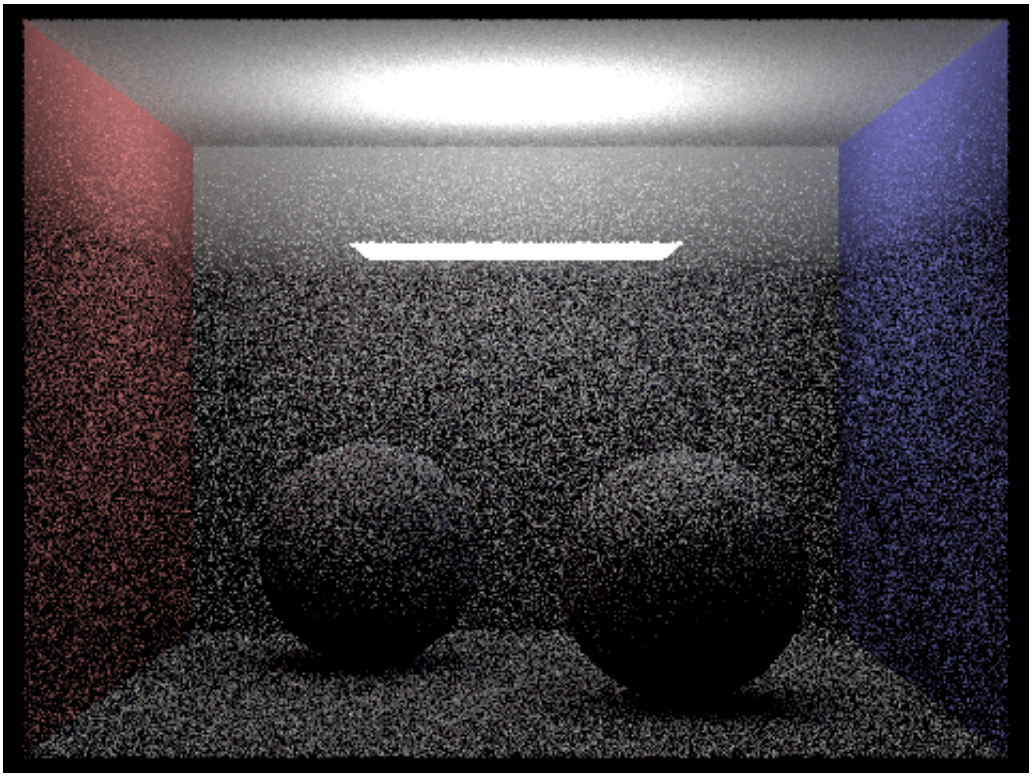
5.1 -s 4 -l 16 -m 0 -w 0

5.2 -s 4 -l 16 -m 1 -w 0

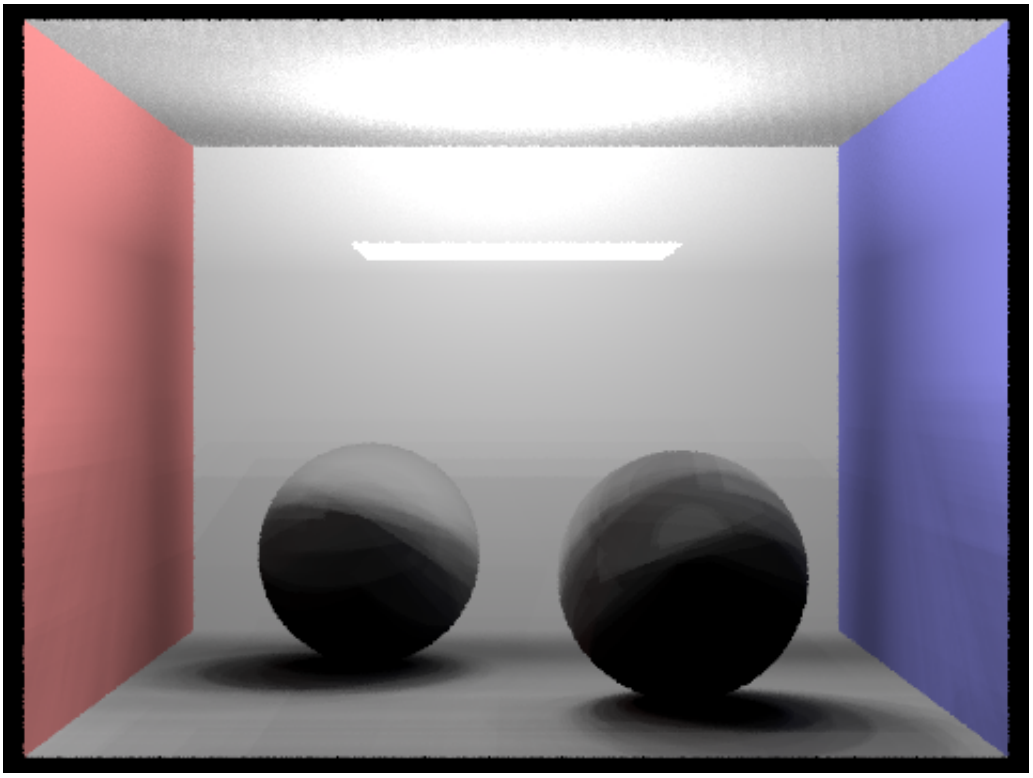


5.3 -s 4 -l 16 -m 0 -w 1

5.4 -s 4 -l 16 -m 2 -w 0

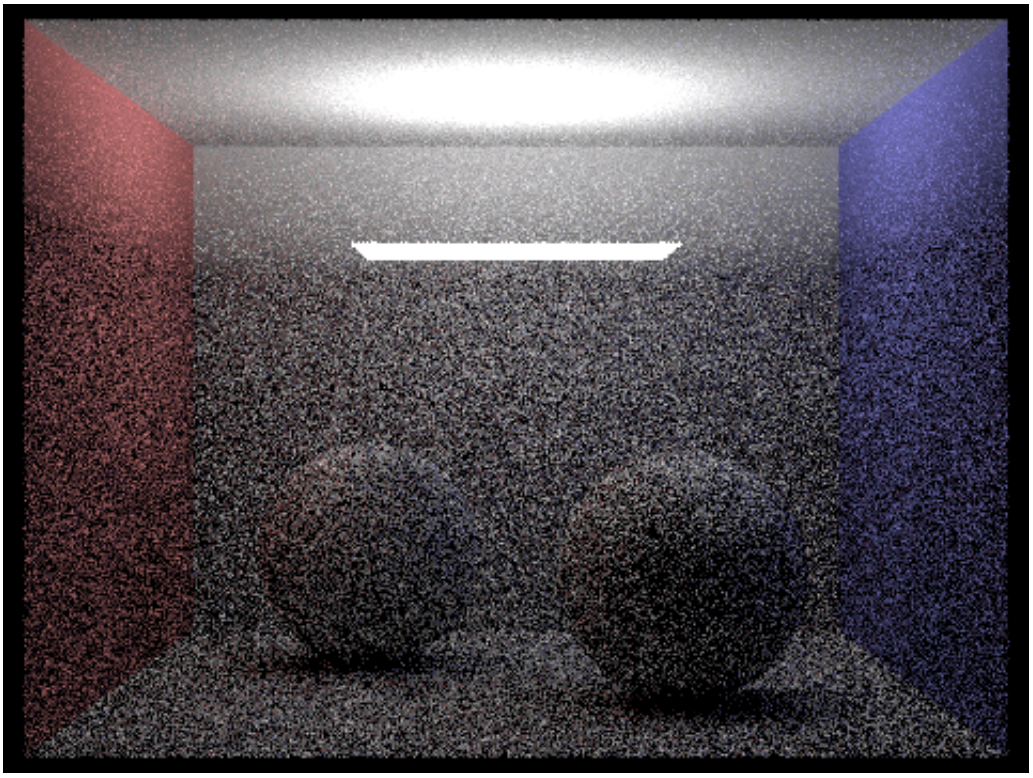


5.5 -s 4 -l 16 -m 1 -w 1

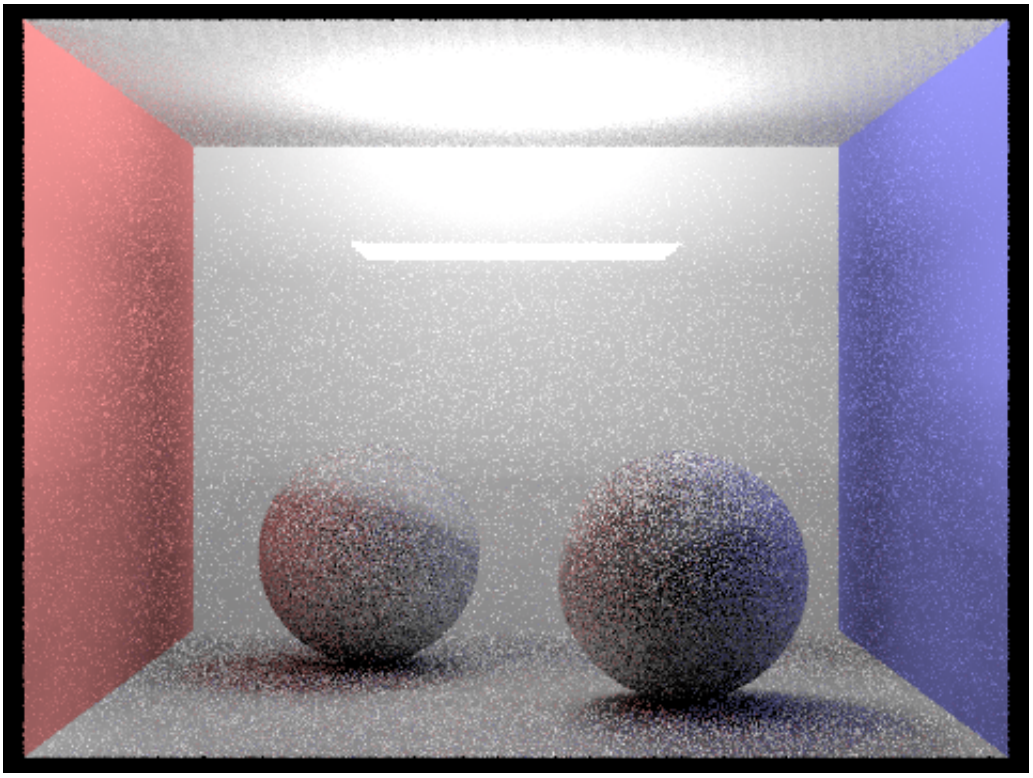


5.6 -s 4 -l 16 -m 0 -w 2

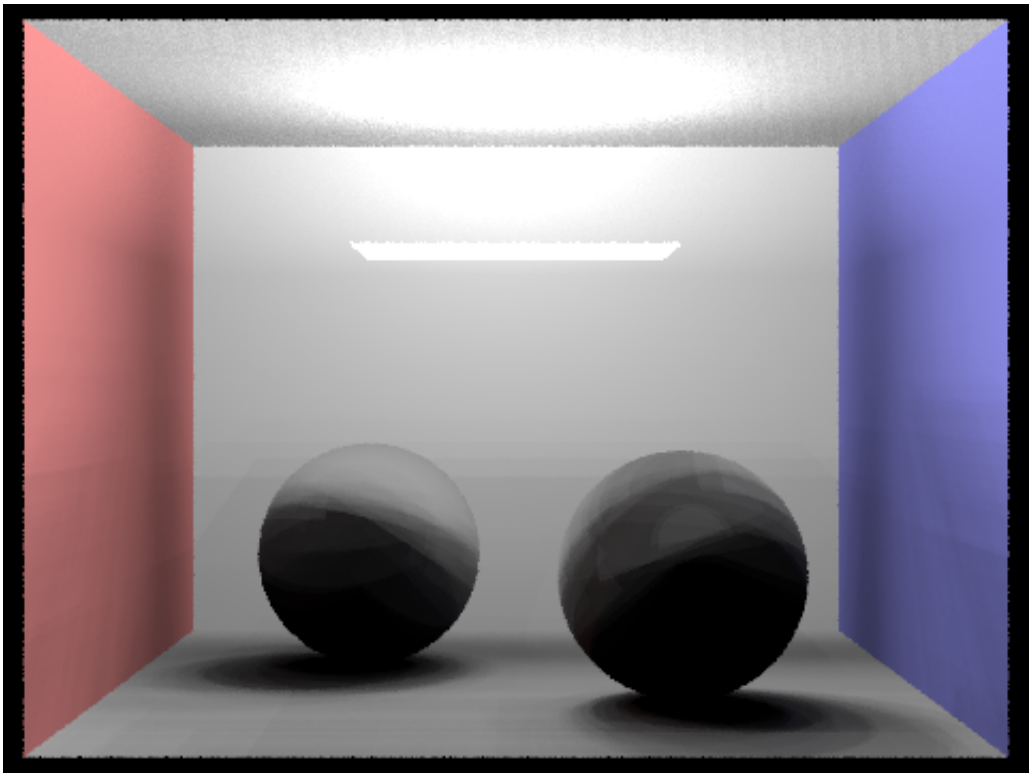
5.7 -s 4 -l 16 -m 3 -w 0



5.8 -s 4 -l 16 -m 2 -w 1

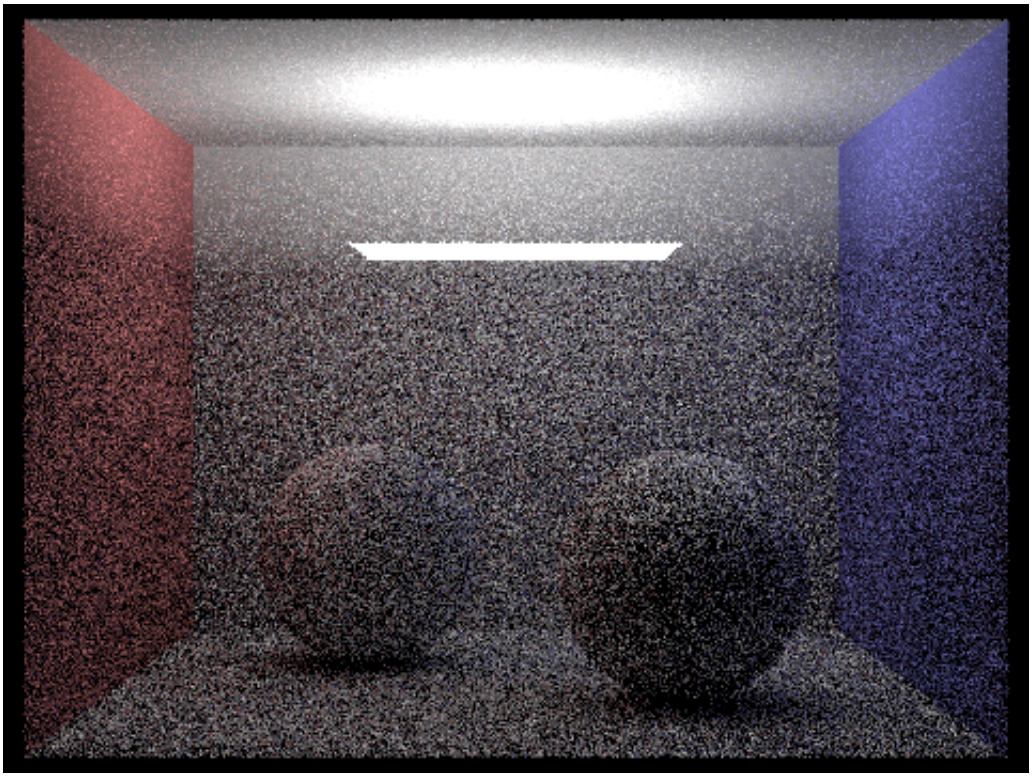


5.9 -s 4 -l 16 -m 1 -w 2

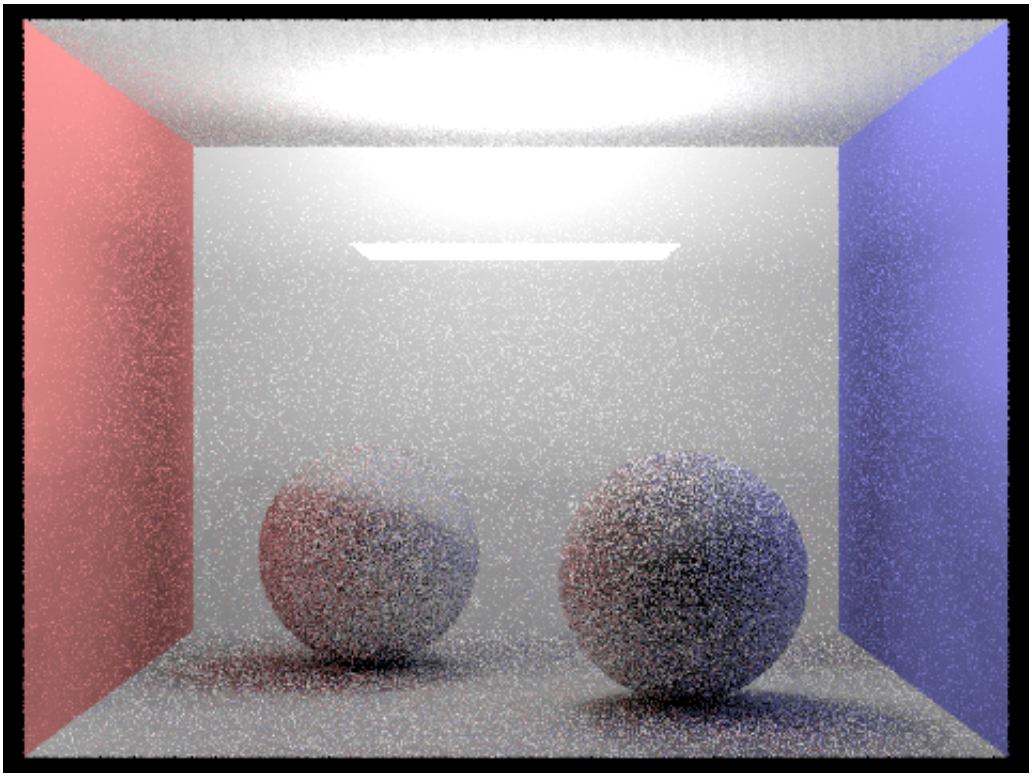


5.10 -s 4 -l 16 -m 0 -w 3

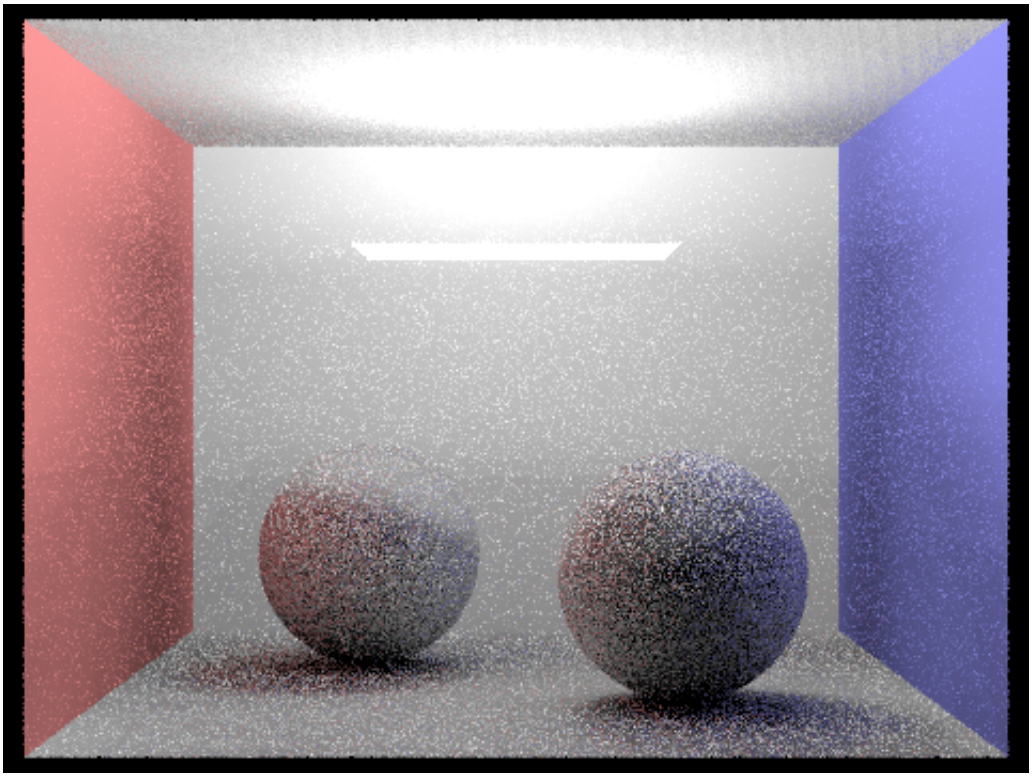
5.11 -s 4 -l 16 -m 4 -w 0



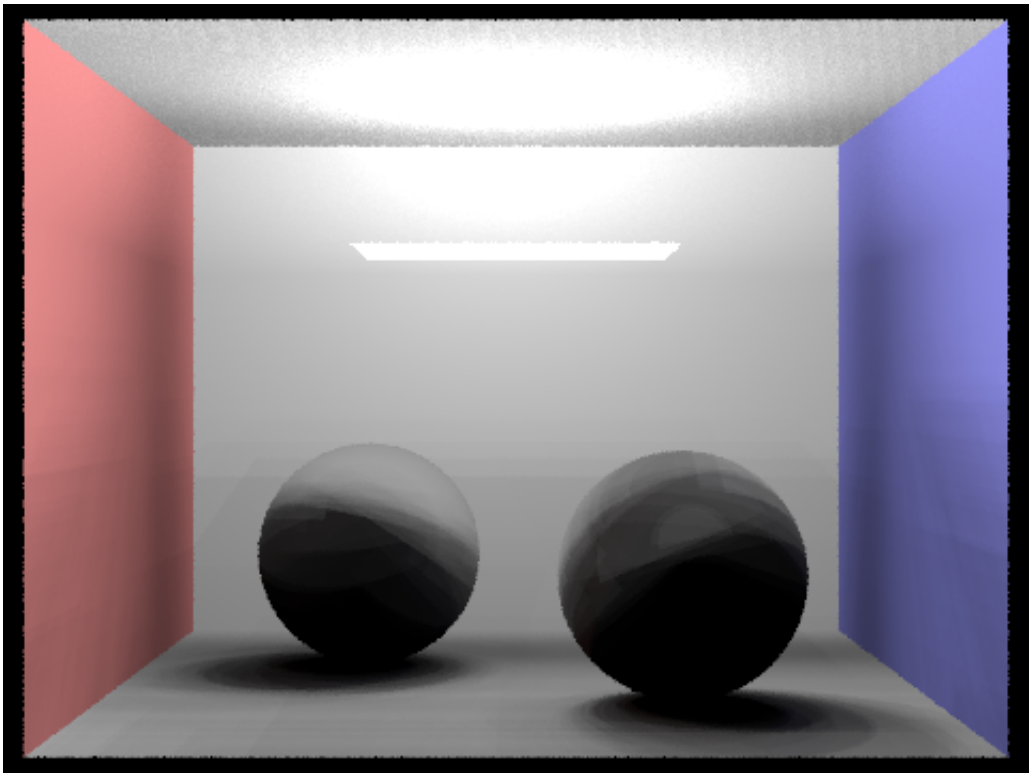
5.12 -s 4 -l 16 -m 3 -w 1



5.13 -s 4 -l 16 -m 2 -w 2

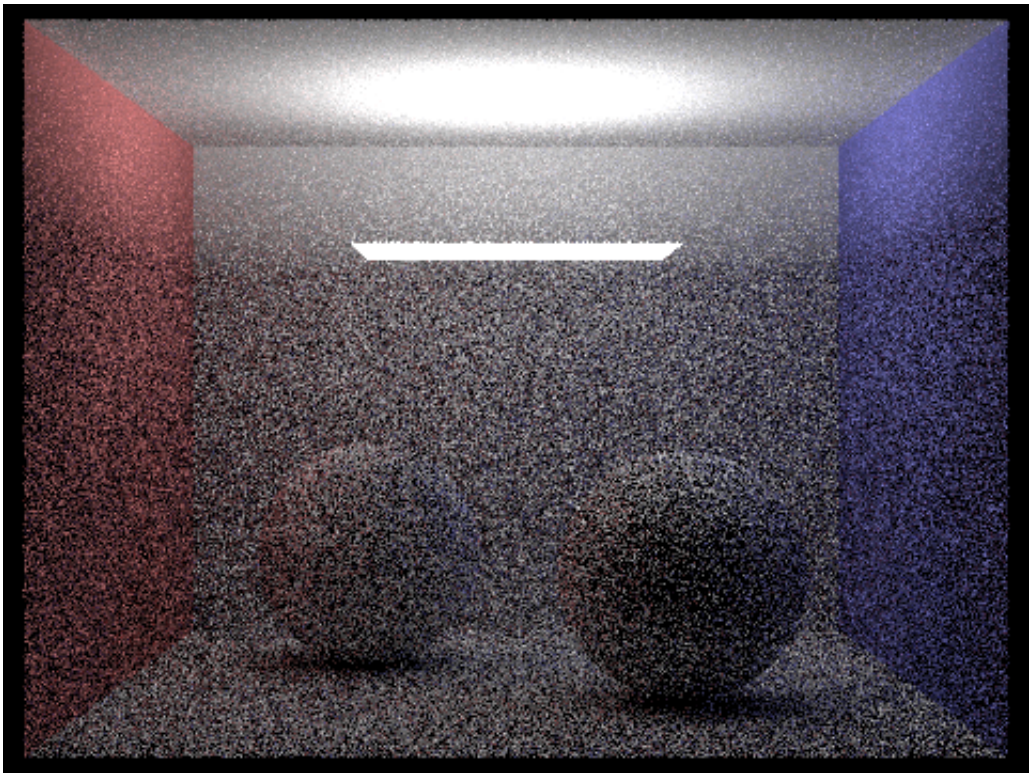


5.14 -s 4 -l 16 -m 1 -w 3

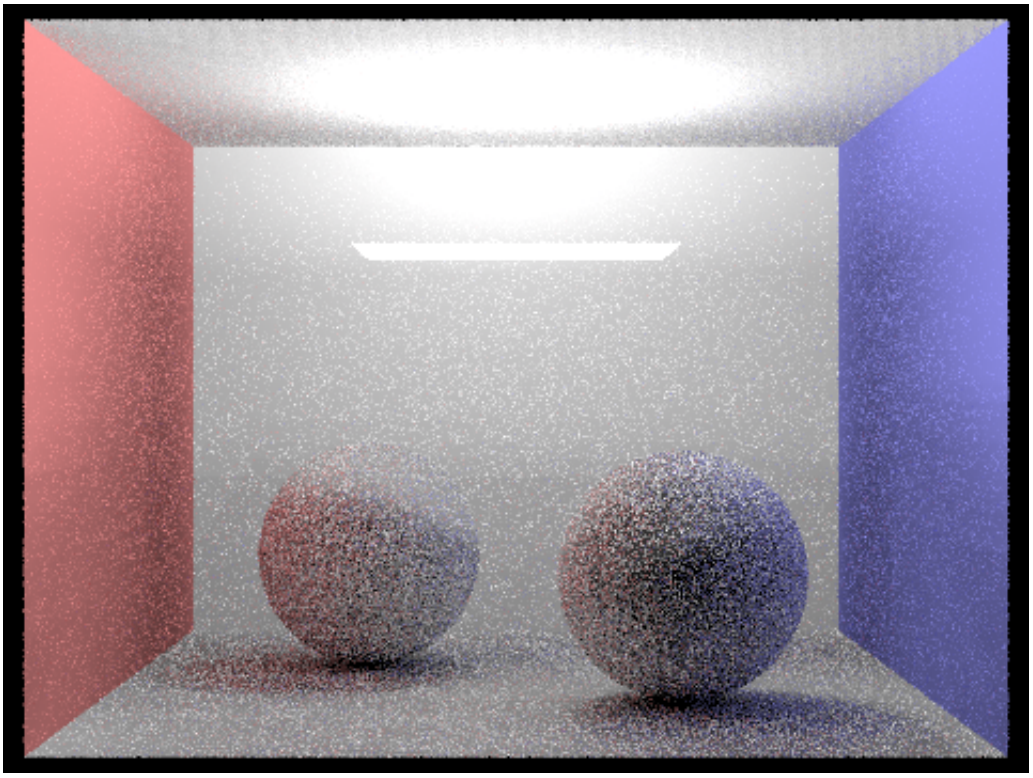


5.15 -s 4 -l 16 -m 0 -w 4

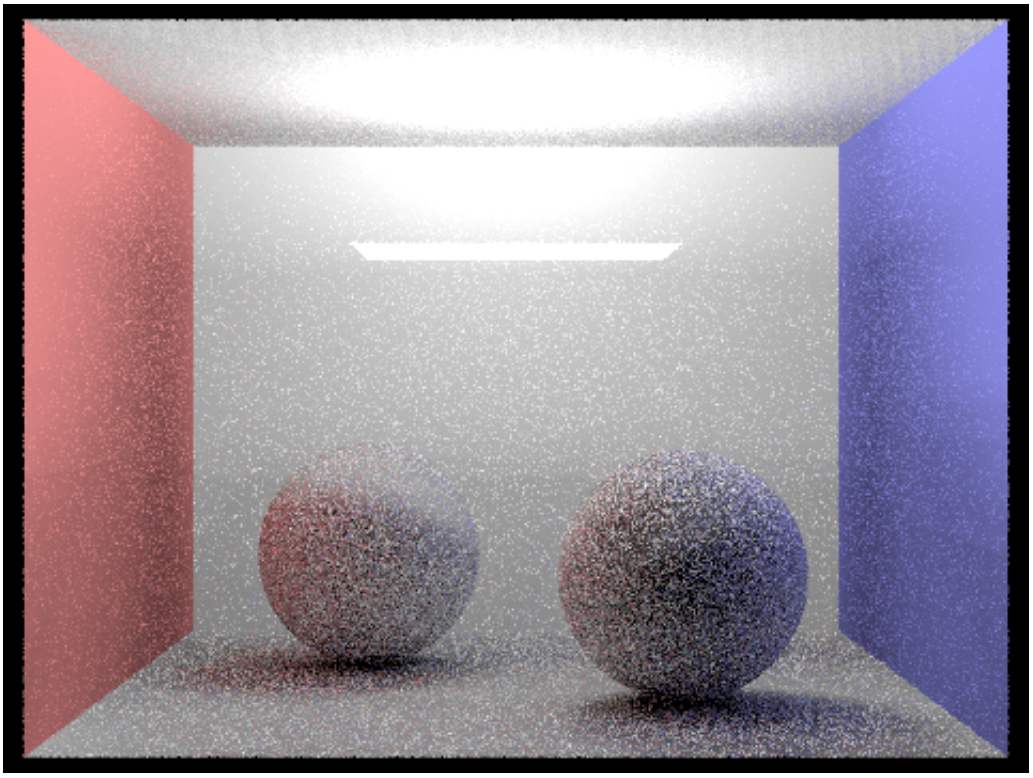
5.16 -s 4 -l 16 -m 5 -w 0



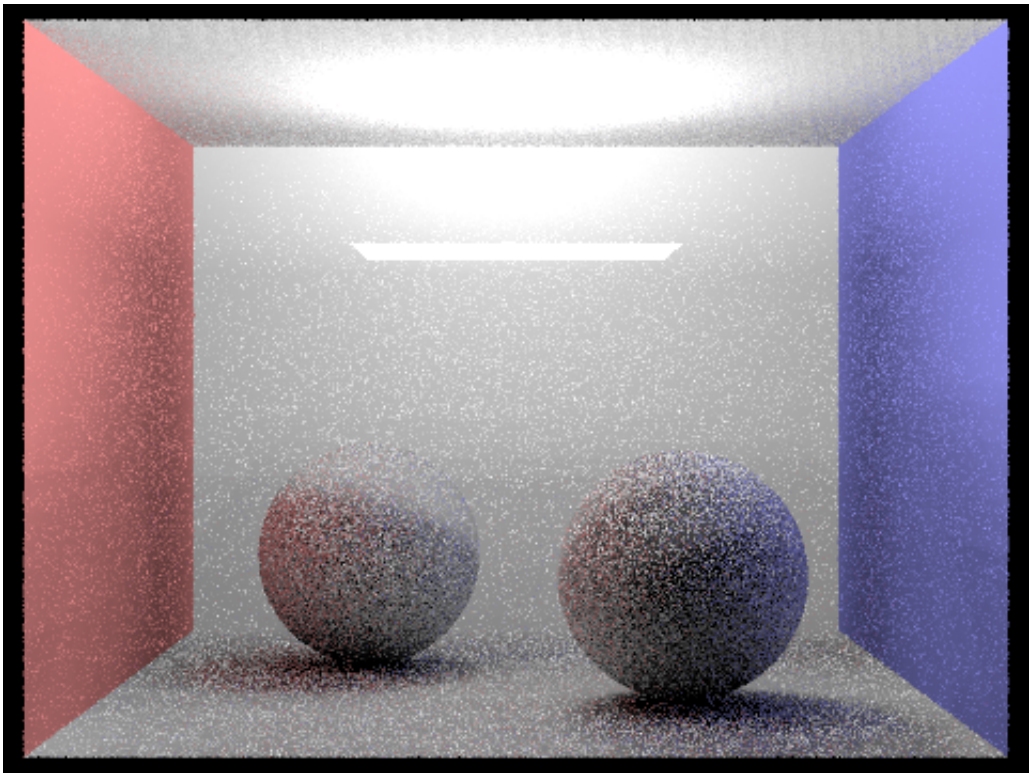
5.17 -s 4 -l 16 -m 4 -w 1



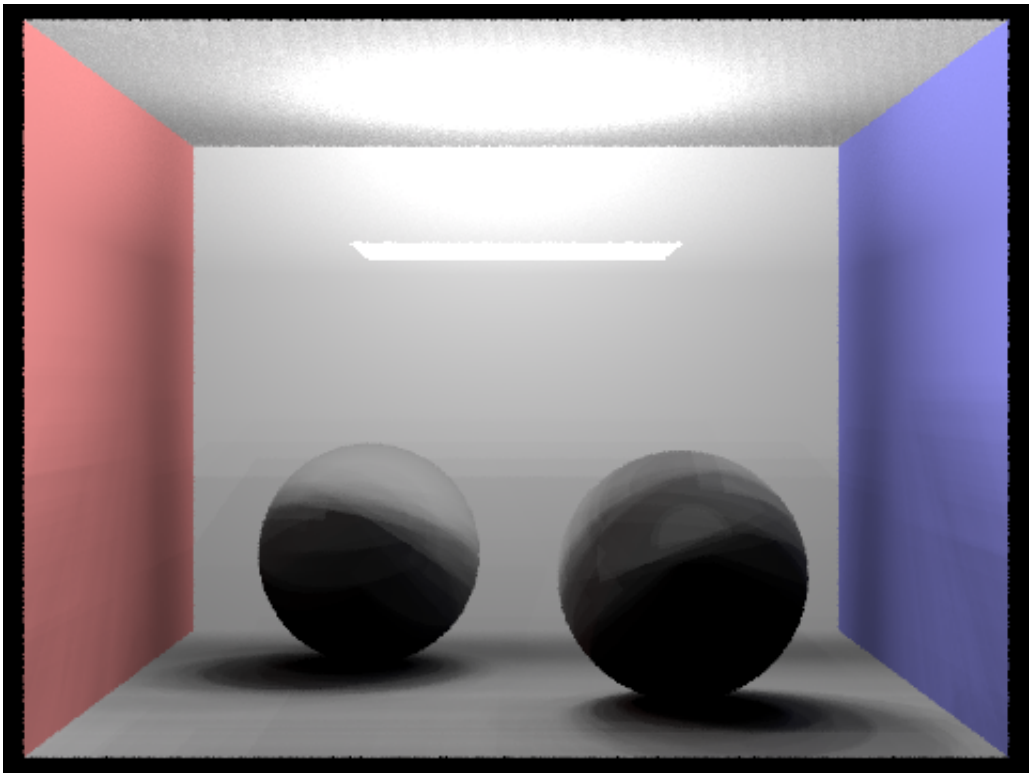
5.18 -s 4 -l 16 -m 3 -w 2



5.19 -s 4 -l 16 -m 2 -w 3

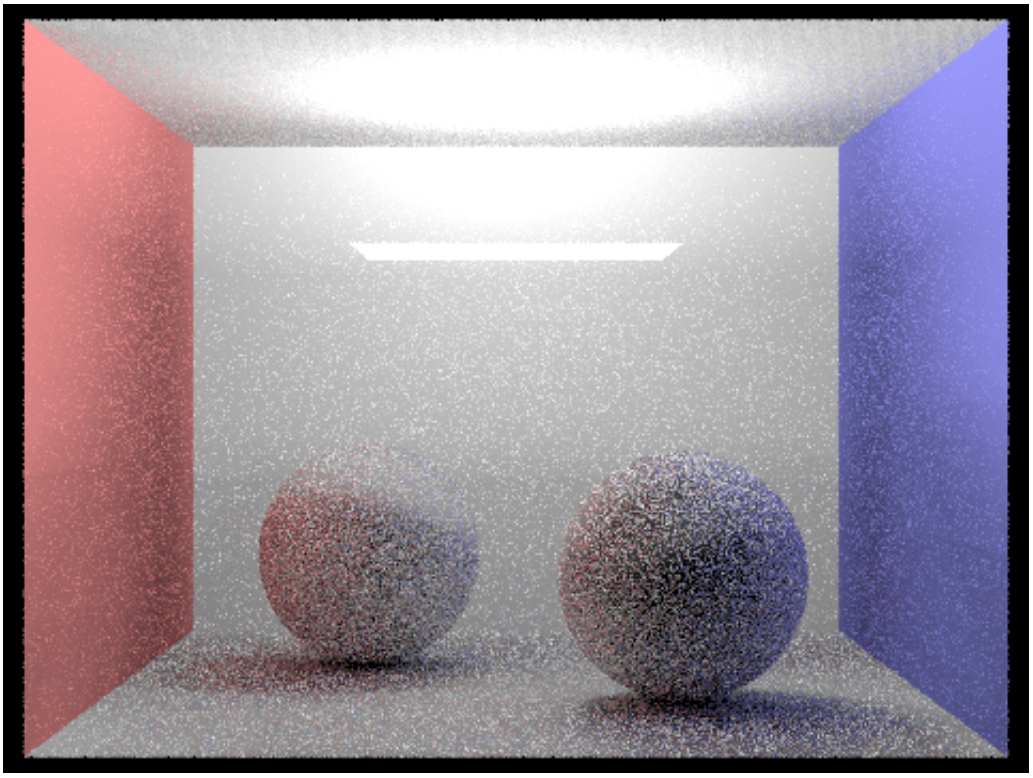


5.20 -s 4 -l 16 -m 1 -w 4

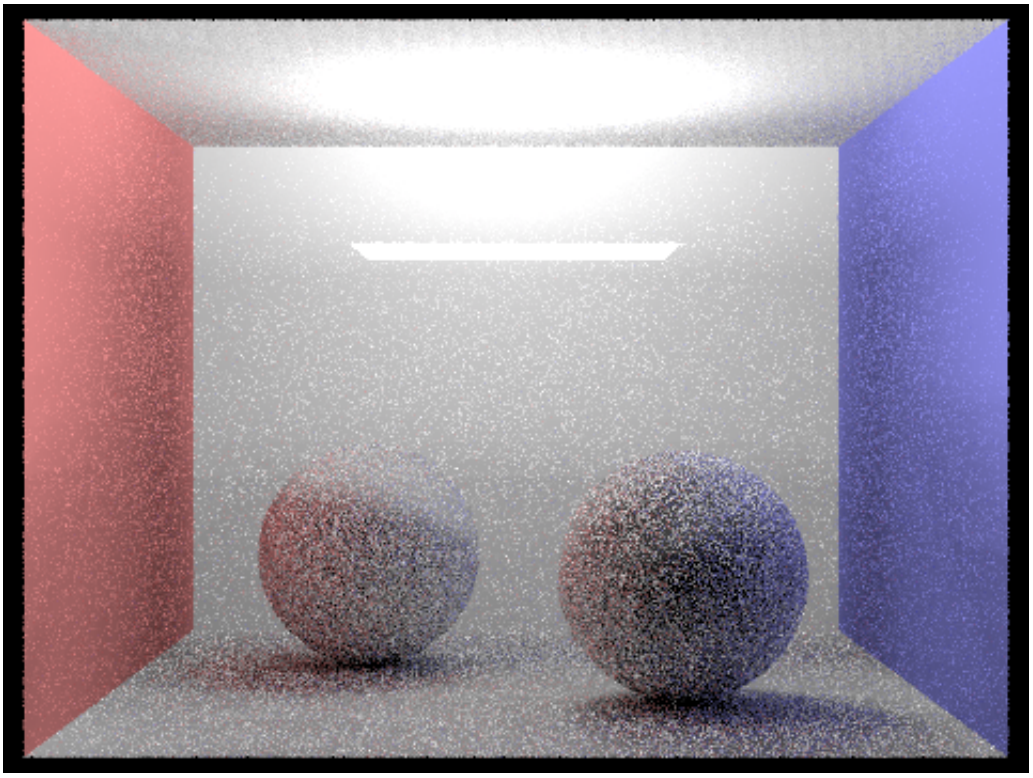


5.21 -s 4 -l 16 -m 0 -w 5

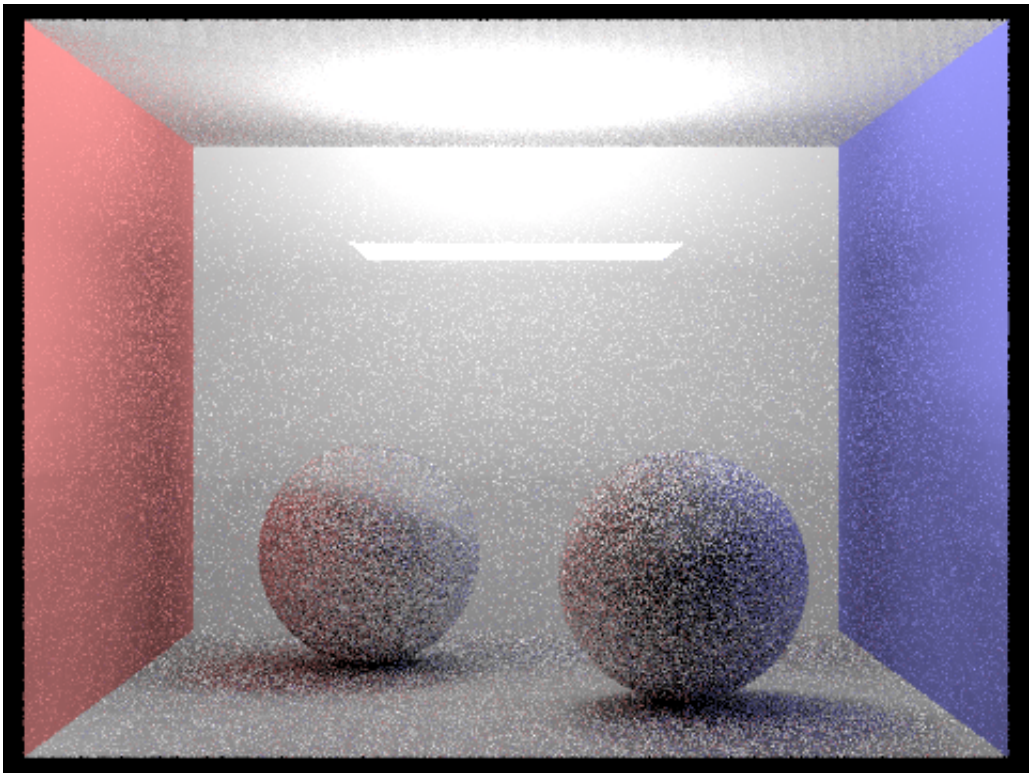
5.22 -s 4 -l 16 -m 5 -w 1



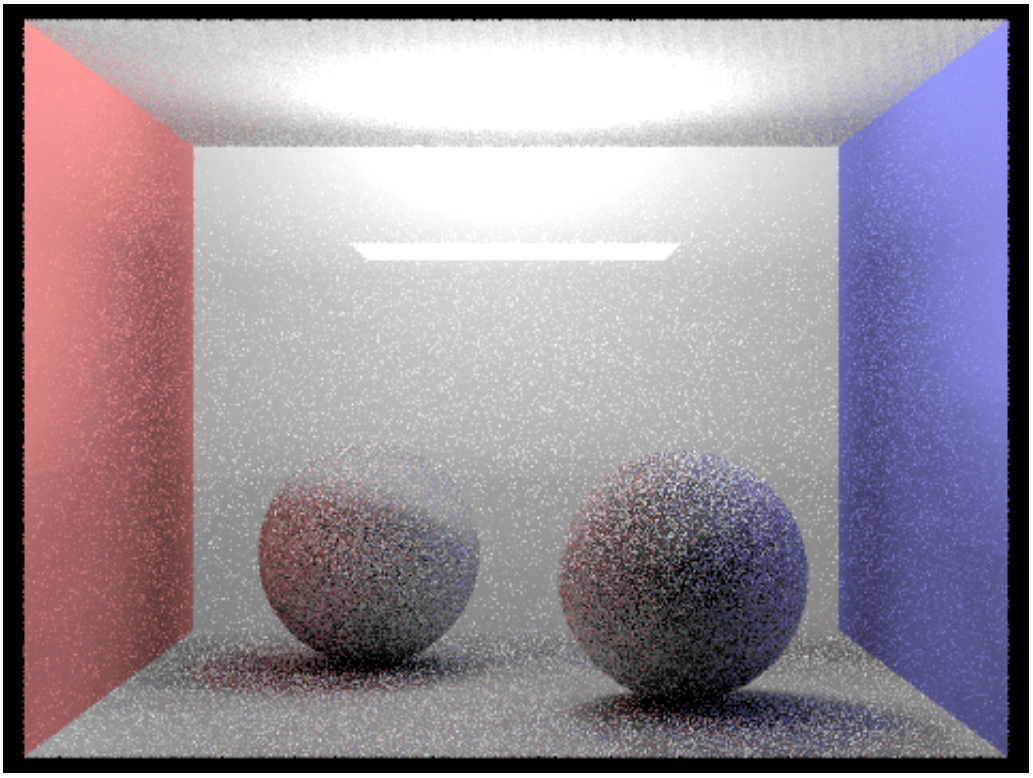
5.23 -s 4 -l 16 -m 4 -w 2



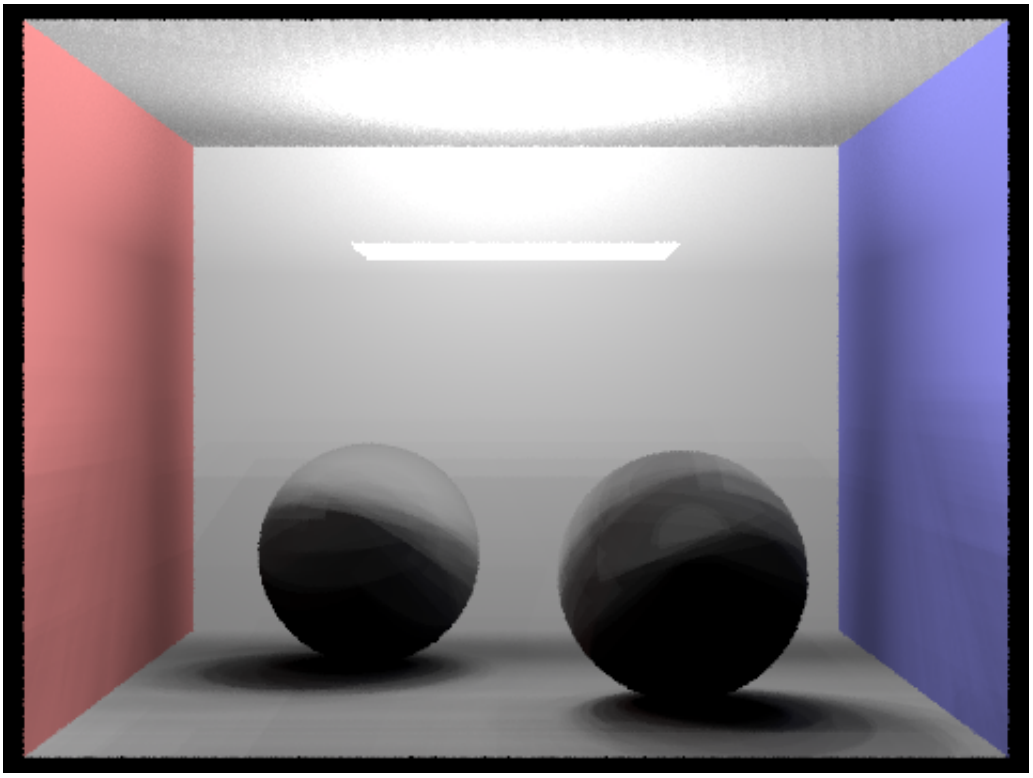
5.24 -s 4 -l 16 -m 3 -w 3



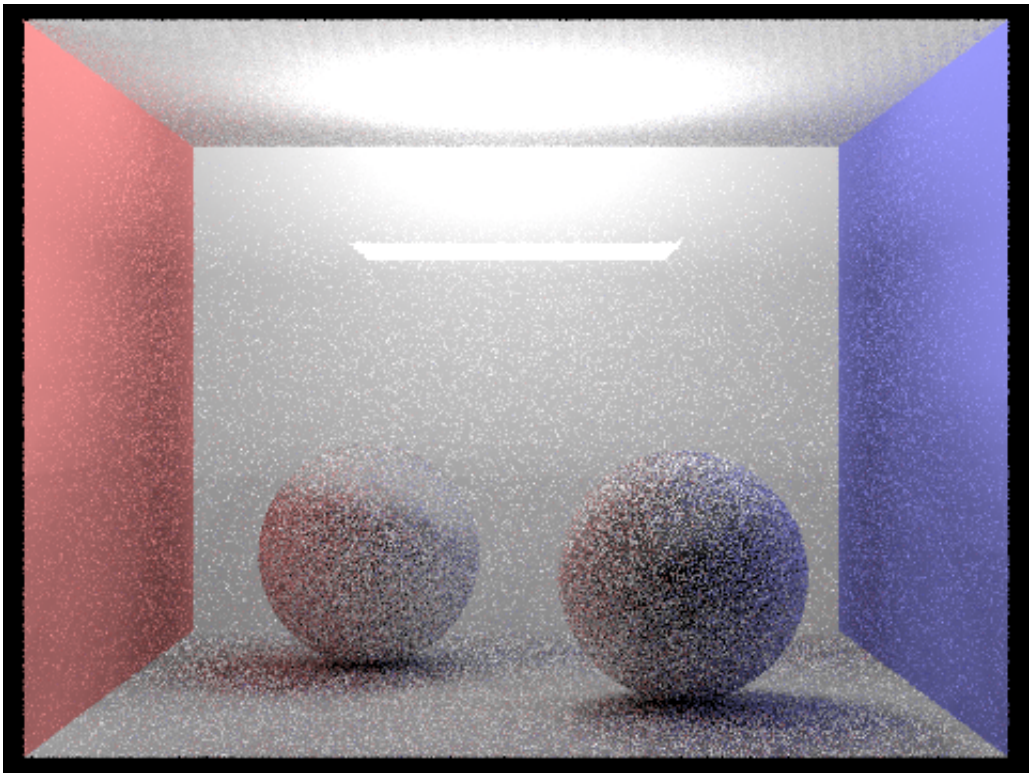
5.25 -s 4 -l 16 -m 2 -w 4



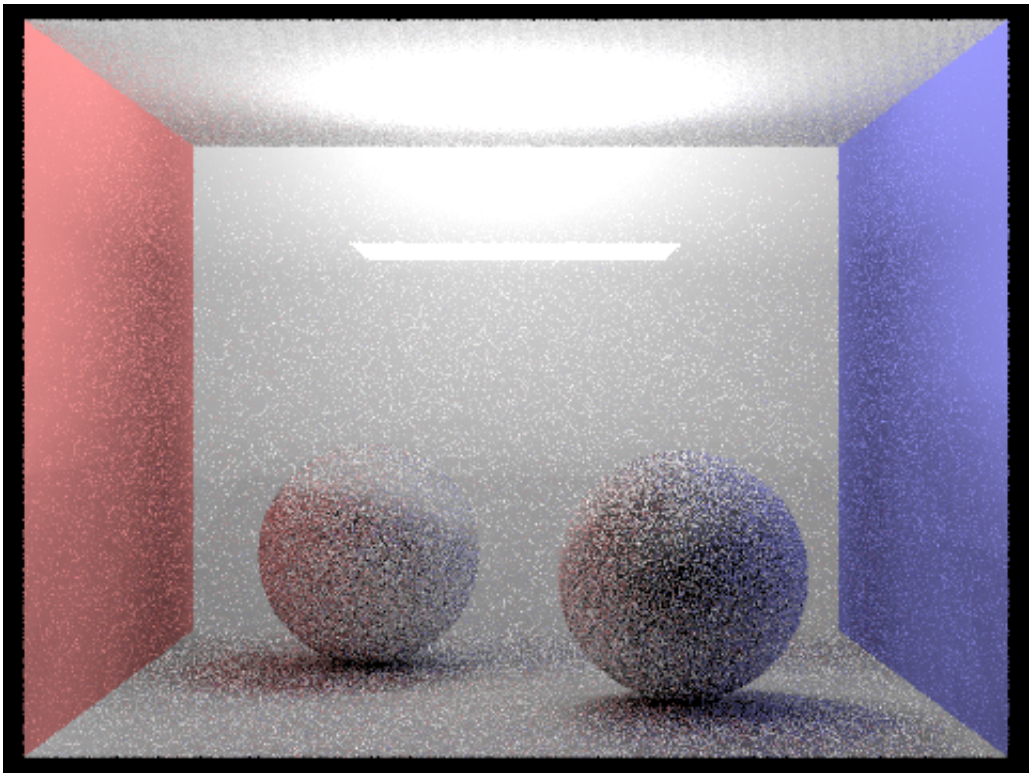
5.26 -s 4 -l 16 -m 1 -w 5



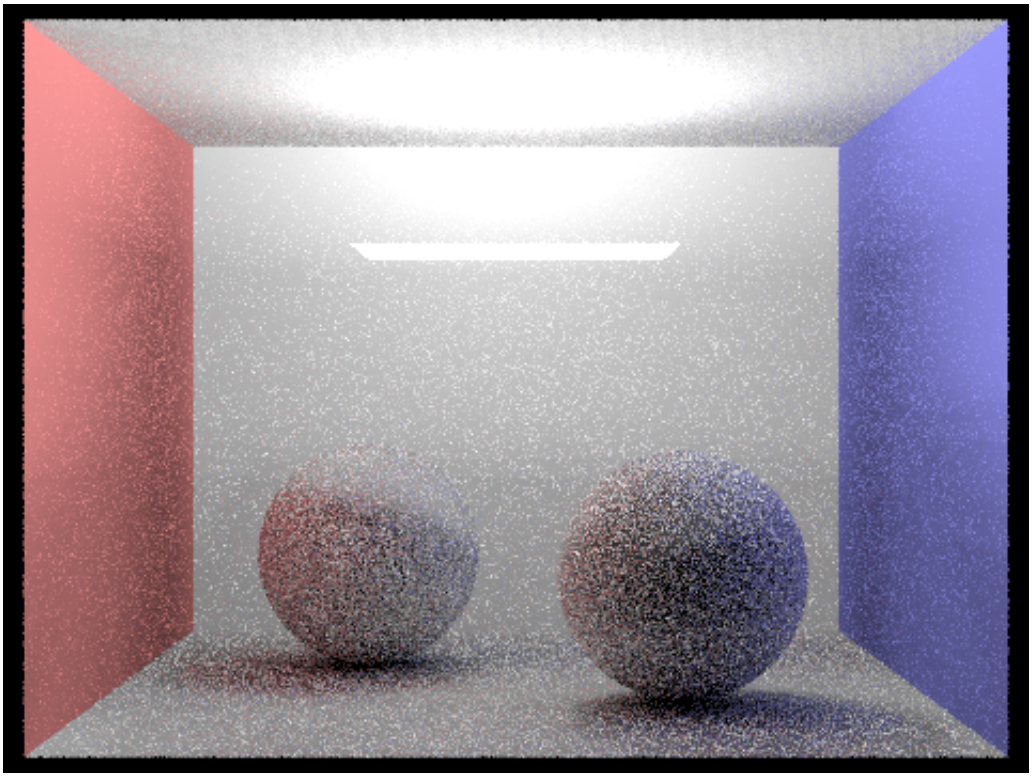
5.27 -s 4 -l 16 -m 5 -w 2



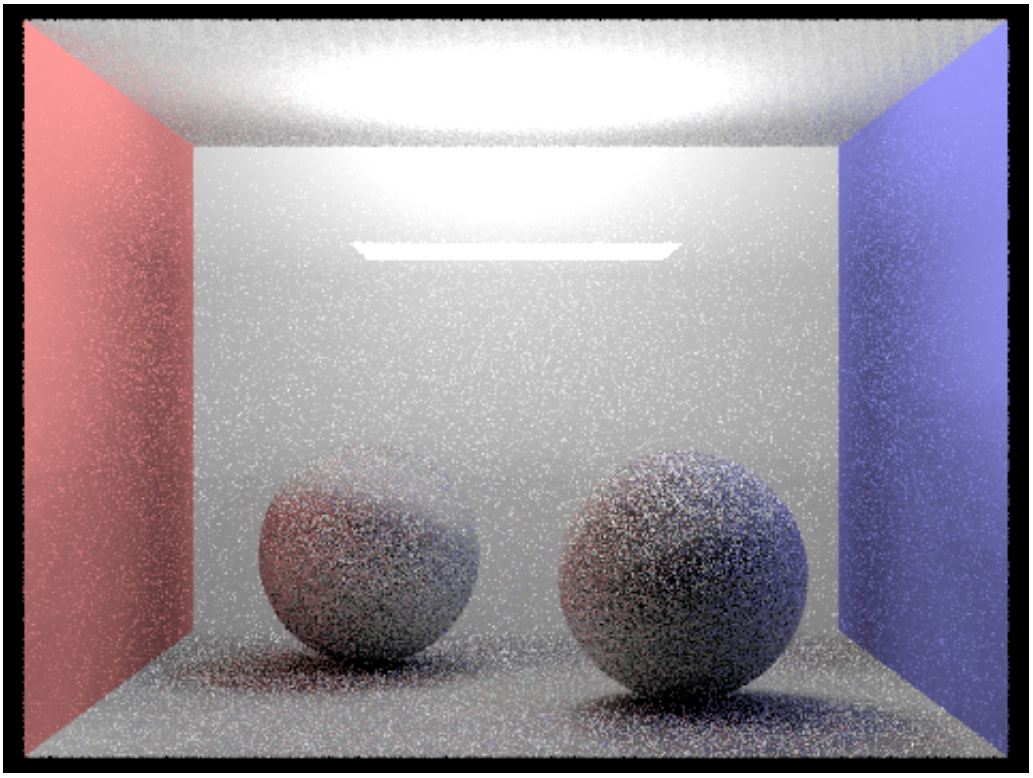
5.28 -s 4 -l 16 -m 4 -w 3



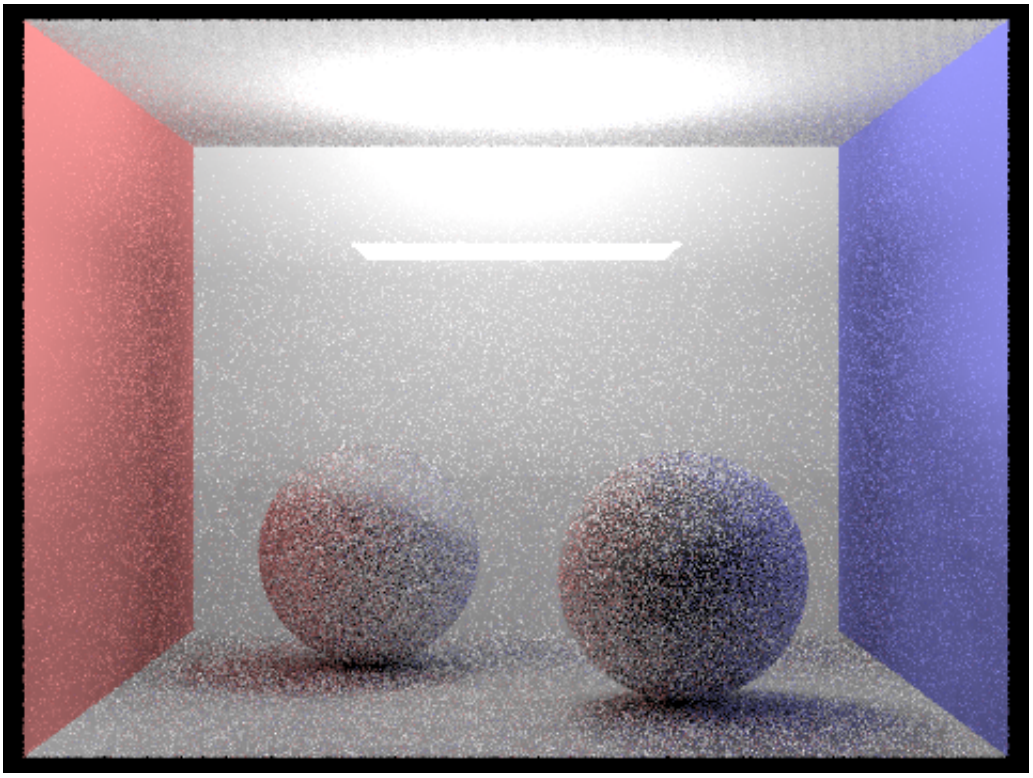
5.29 -s 4 -l 16 -m 3 -w 4



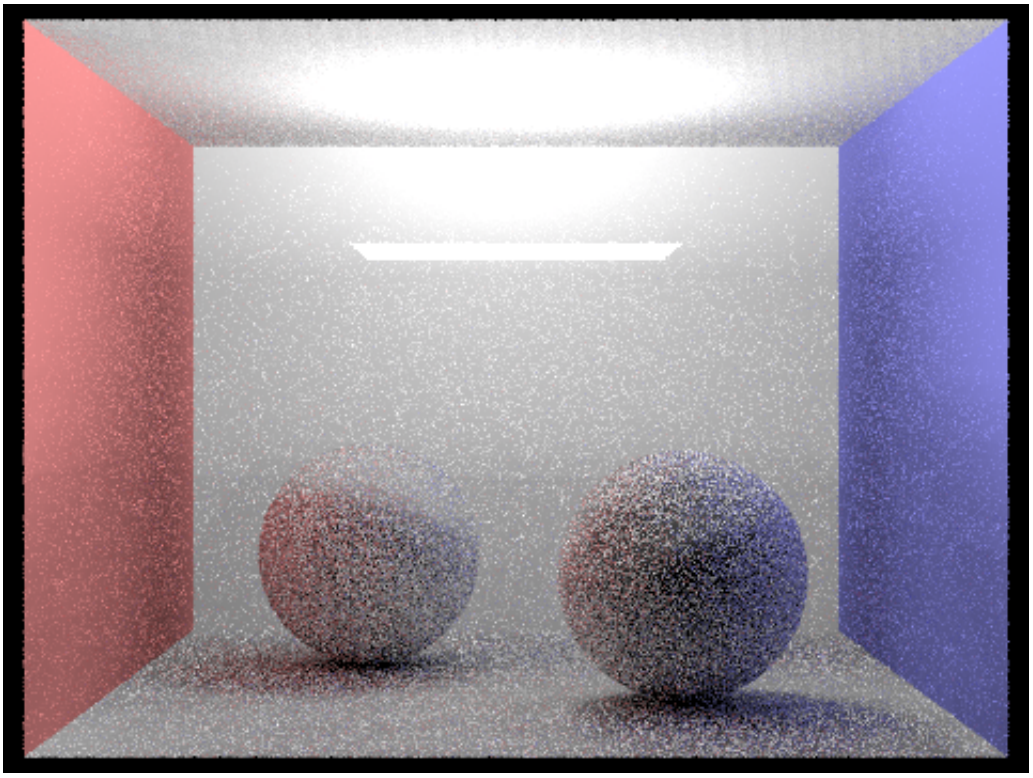
5.30 -s 4 -l 16 -m 2 -w 5



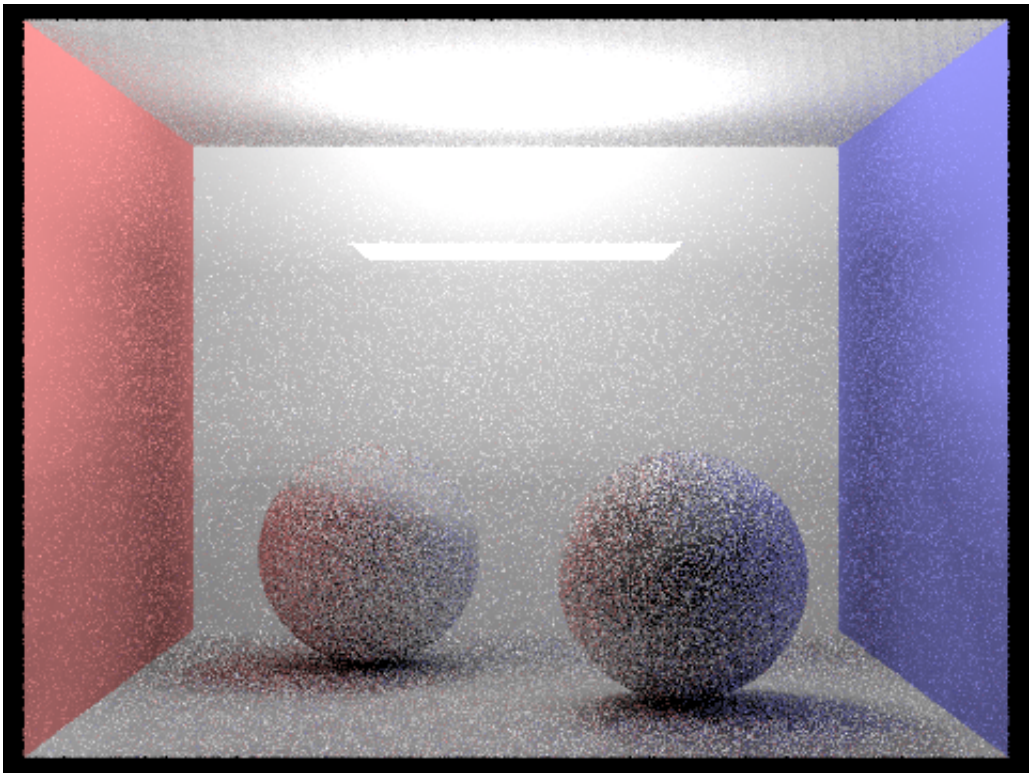
5.31 -s 4 -l 16 -m 5 -w 3



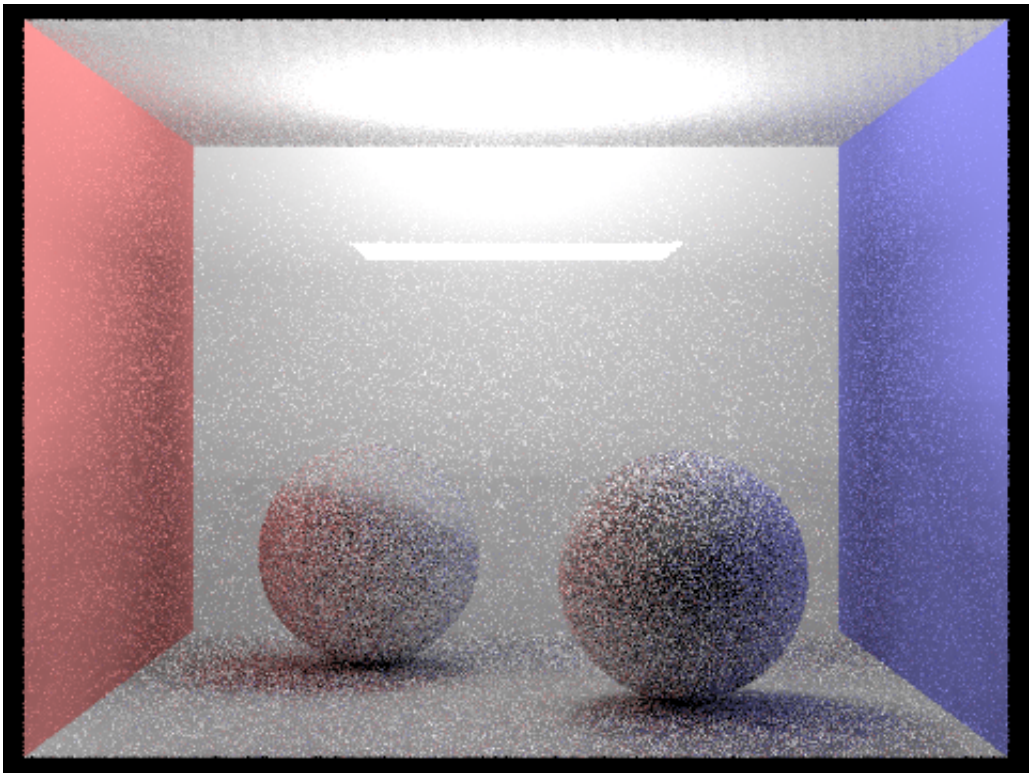
5.32 -s 4 -l 16 -m 4 -w 4



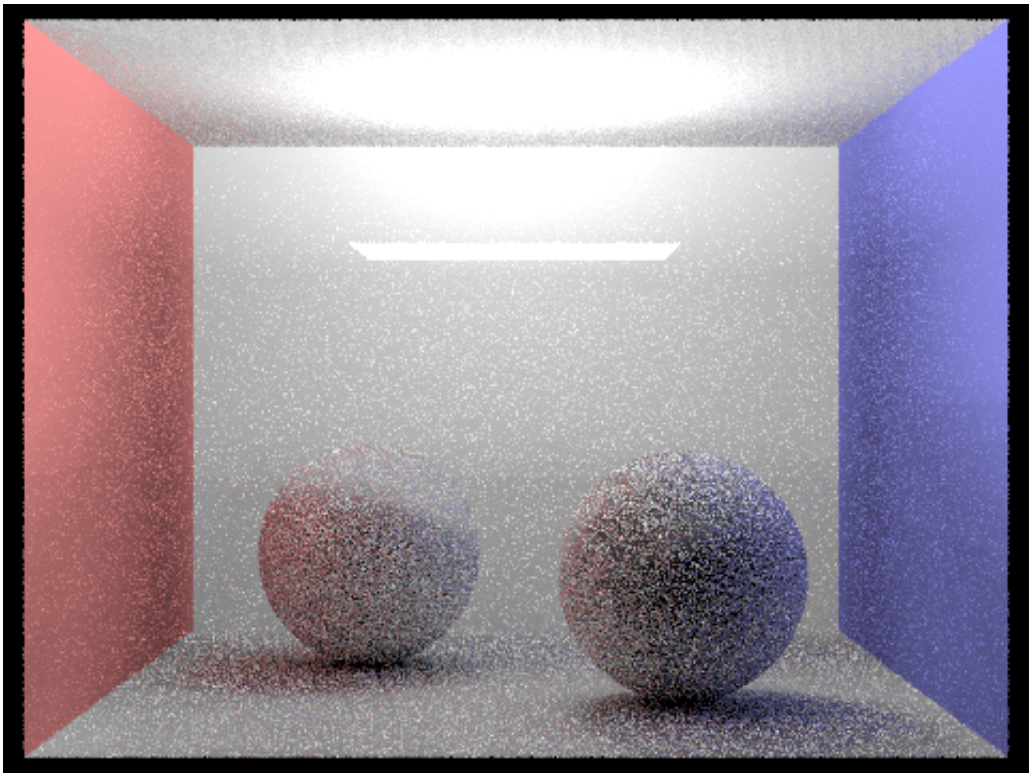
5.33 -s 4 -l 16 -m 3 -w 5



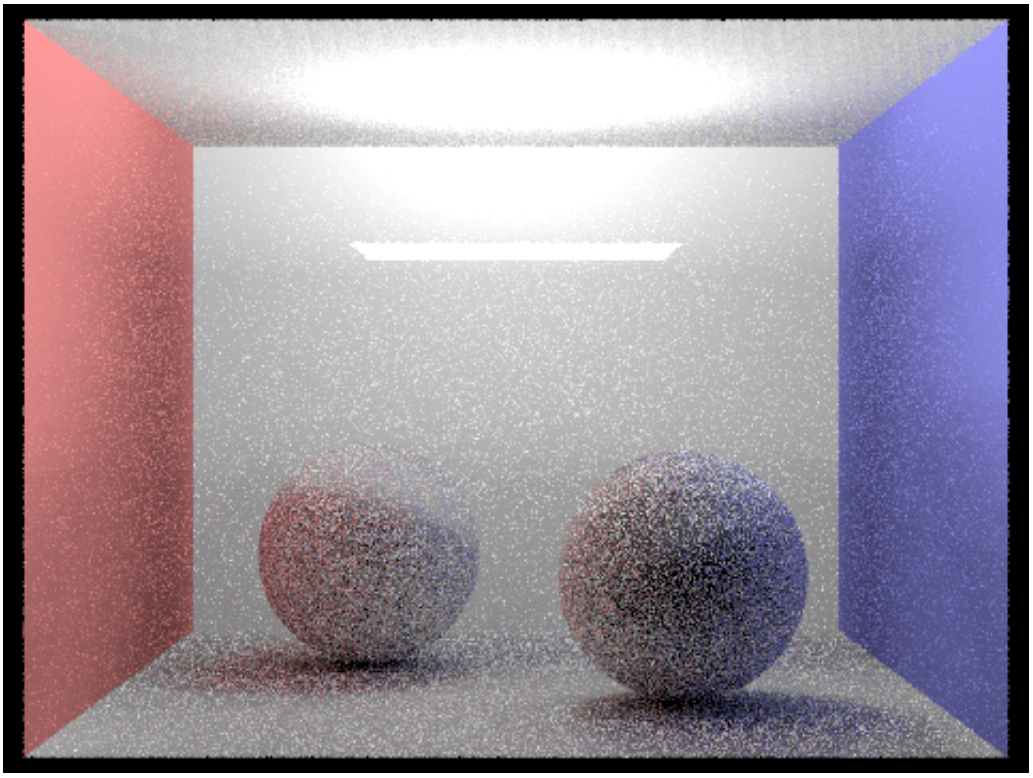
5.34 -s 4 -l 16 -m 5 -w 4



5.35 -s 4 -l 16 -m 4 -w 5



5.36 -s 4 -l 16 -m 5 -w 5



When BDPT is disabled, the lower part of the scene (which is indirectly illuminated) is very dark and noisy. When BDPT is enabled, both the lower part and the upper part become much brighter and less noisier.

When the maximum depth of forward sub-path (-m flag) is 1 and the maximum depth of backward sub-path is greater than 0, the lower part of the image is completely illuminated by connecting the forward sub-paths and backward sub-paths. Note that the lower part is noise-free. Through these images, we can know how backward path tracing contributes to the BDPT results. Also, we can see that the spheres are mottled, which is caused by insufficient light sampling in backward path tracing. When the maximum depth of forward path is increased, the lower part becomes brighter but noisier.

6 Limitations & Further Improvements

This project only implements a simple BDPT algorithm due to the time limit. To render less noisy and more accurate results, Multiple Importance Sampling (MIS) method can be implemented. There are a few MIS heuristics, such as the balance heuristic, the cutoff heuristic, and the power heuristic. By using MIS, the variance can be effectively reduced.

This project only implements CPU rendering due to the time and device limit. When BDPT is enabled, the rendering process can take tens of minutes on a Intel Core i7 CPU. GPU (e.g. CUDA) can significantly accelerate the rendering process through high degree of parallelization.

7 Credits

Special thanks to Mr. Tengfei Wang (118010307) for providing a way to sample the lights. To be specific, how to sample a direction, how to sample a position (for area lights), and how to calculate the probability density functions.