Rendering Final Report

**Abstract**

We dived into the world of photon mapping and compared the original version with several refined methods, including Sped-up Photon Map (SPM+), Progressive Photon Map (PPM) and Stochastic Photon Map (SPPM).

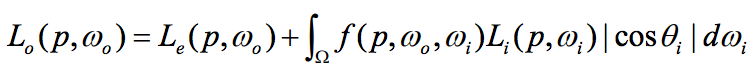
We implemented the SPM+ based on “Faster Photon Map Global Illumination” by *Per H. Christensen*and PPM by modifying the source code of SPPM in pbrtv.3.

**Scene Creation**

Our scene is a Cornell Box with a glass and a mirror sphere, which is created by stuffing the two spheres in *spheres-differentials-texfilt.pbrt* into Cornell box. Note that we had to change the radius and the position of the spheres or they would appear too small or at weird positions.

**Algorithm**

* SPM+

****In traditional Photon mapping, during rendering, when doing final gathering, it needs to compute “irradiance”, noted with red rectangle by referring to a certain amount of photons near the hit point. However, this consumes lots of time, since the hit point differs every time, necessitating computation of the red part. And indeed, this is the most time spent in photon mapping, compared with photon shooting and kd-tree construction.

The paper by *Christensen* observes that though the intersected points differ, they are actually pretty close, meaning the re-computing every time is a little bit overkill. Hence, the algorithm proposed to pre-compute irradiance at some positions. And when irradiance is needed at a certain point, the value of the closest pre-computed position is used.

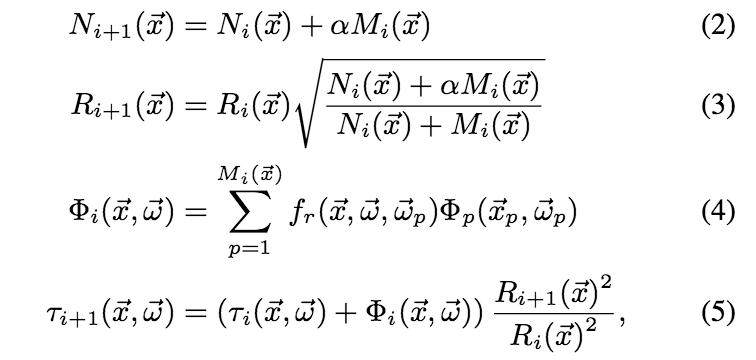
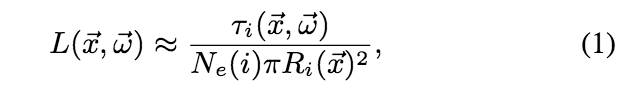
And concerning position selection, where photons are deposited is a good choice, since photons are inherently dense in areas with large illumination variance, making the approximation more accurate.

* PPM

In traditional Photon mapping, all photons are deposited at the beginning. Then we use those photons when computing radiance. The more photons we use, the more accurate radiance we get. But the number of photons is limited by the memory size. To break through the limitation, here comes Progressive Photon Mapping(PPM). PPM is a revision of PM, processing different photons in each iteration. This way, PPM can get rid of the memory limitation and use arbitrarily amount of photons we want.

There are three steps of PPM: 1. sample a ray from image plane and calculate the hit point of ray in the scene 2. shoot photons to the scene 3. accumulate the radiance caused by the photons near the hit point within distance r. The most important thing is that we can iterate the step 2 and 3 until the amount of photons is enough.

The accumulation details are below:



At the i-th pass, the radiance value of point x toward direction w is estimated as formula (1). Ni(x) is the summation amount of photons in i-th pass. Mi(x) is the amount of photons found within radius Ri(x). alpha is a user-defined parameter ranging in (0, 1). Phi is the flux of position x toward direction w. fr is the brdf of position x, direction w and direction wp(the direction of photon ray). We can find that the radius shrinks with the iteration. The radiance estimation will converge to the correct radiance value with an infinite number of iterations.

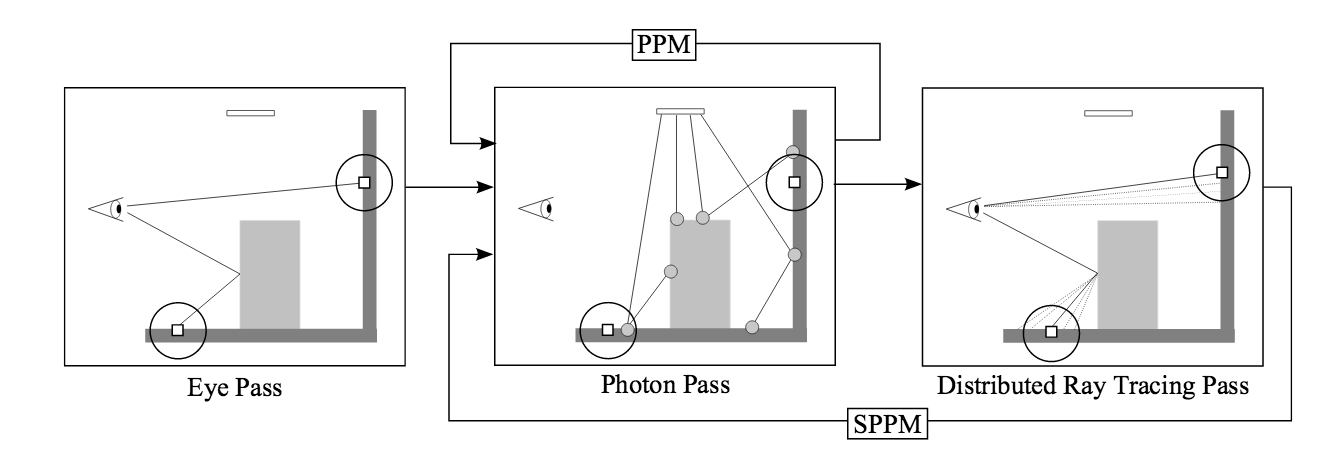
In our implementation, we chose pbrt-v3 to implement this because the architecture of pbrt-v3 is more appropriate than pbrt-v2. In pbrt-v3, there is no renderer any more, instead, it uses integrator as a renderer. Thus, we generate all hit points at once, iteratively shoot photons and accumulate radiance. The benefit of pbrt-v3 architecture is we can reuse the photons in each iteration. Unlike pbrt-v2, if we implement PPM as a SurfaceIntegrator, we can only generate the photons in its Li function. In other words, we cannot reuse the photons we just generated and re-generate plenty of photons when the Li function is called. Thus, implement PPM in pbrt-v3 is faster than in pbrt-v2 because of the architecture.

Besides, to speed up the finding of the hit point affected by the photons generated in each iteration, we implemented a hash function. The hash function split the hit points into different hash by their bound (the region of the hit point within the radius r). When we generate a new photon, we can use the hash function to shrink the amount of hit points of which we need to check whether that photon is within the bound.

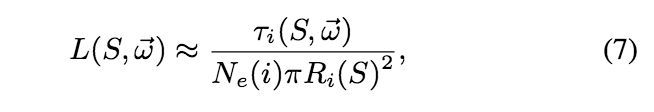
Summary our implementation of PPM, there are five steps: 1. sample an amount of ray from image plane based on the sample rate and calculate the hit point of rays in the scene 2. construct the acceleration data structure to speed up the accumulation step 3. shoot photons to the scene 4. accumulate the radiance caused by the photons near the hit point within distance r 5. average the radiance of every sample rays of each pixel and fill in the value into the image. In these five steps, we can iterate step 3 and 4 many times until we use enough photons.

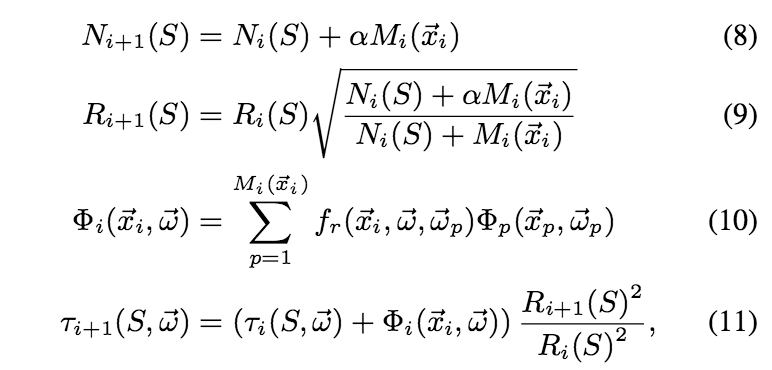
* SPPM

The most important difference between SPPM and PPM is that SPPM uses different hit points within an area in each iteration to accumulate the radiance.



From the figure above, we can find that SPPM re-sample a ray from eye in each iteration. Below is the formula of SPPM.





From the formulas above, we can find that SPPM replace many x with S. This means that SPPM cares about the radiance of an area rather than a point. Thus, there are still three steps of SPPM: 1. sample a ray from image plane and calculate the hit point of ray in the scene 2. shoot photons to the scene 3. accumulate the radiance caused by the photons near the hit point within distance r. But the biggest change is that the iteration contains step 1 to 3 rather than only step 2 and 3.

**Comparison**

We ran our experiments on cml server 21, with hardware specification as follows:

Intel Xeon CPU E5-2650 v2 having 32 cores, 2.6GHz. 128G memory.

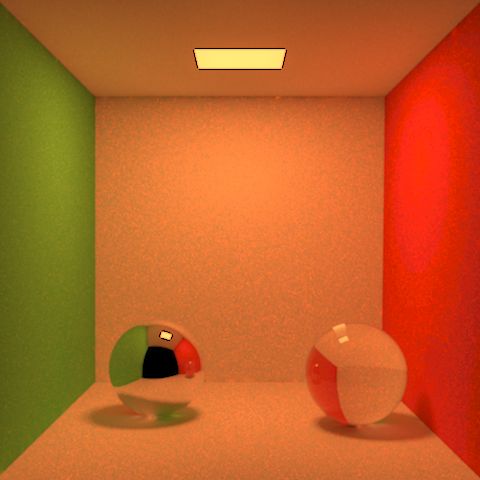
* PM v.s. SPM+

PM 32 samples/pixel 570.4s

SPM+ 32 samples/pixel 378.9s

PM 64 samples/pixel 1048.9s

SPM+ 64 samples/pixel 731.4s

  PM32, PM64  SPM+32, SPM+64

We Found that the speedup factor is not as huge as what the paper indicated, which we thought is because we didn’t optimize the part of finding the nearest photon. That is, instead of utilizing the existing kd-tree and run a loop while to increase the searching distance until we find the first photon, we should implement a hashing-based method, which will drastically shrink the required time.

* PPM v.s. SPPM

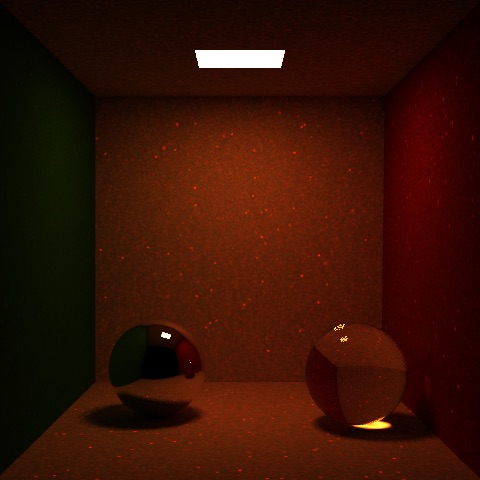
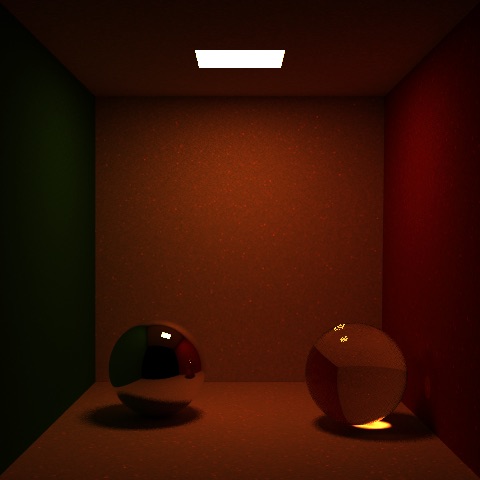
Fixed iteration

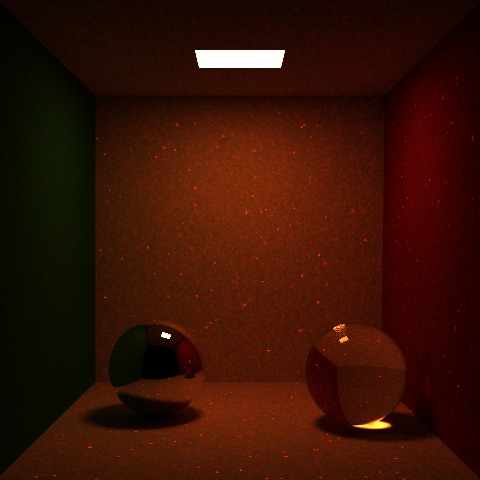
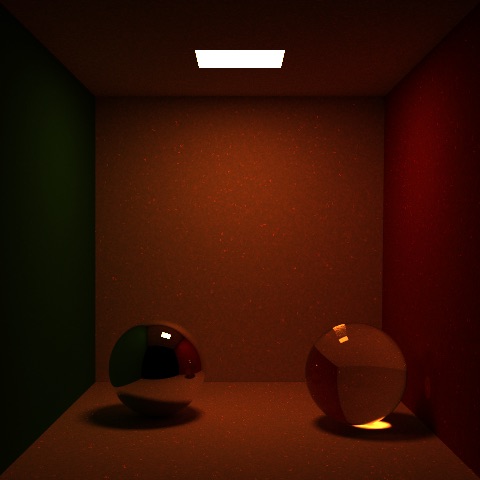
PPM(16 sample/pixel) 100 iteration 50.5s

SPPM 100 iteration 9.0s

PPM(16 sample/pixel) 500 iteration 239.5s

SPPM 500 iteration 42.5s

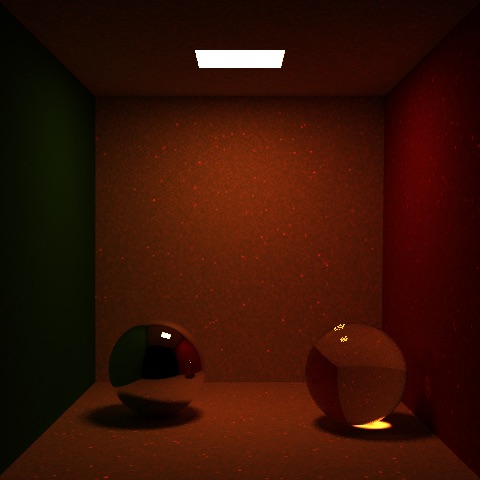
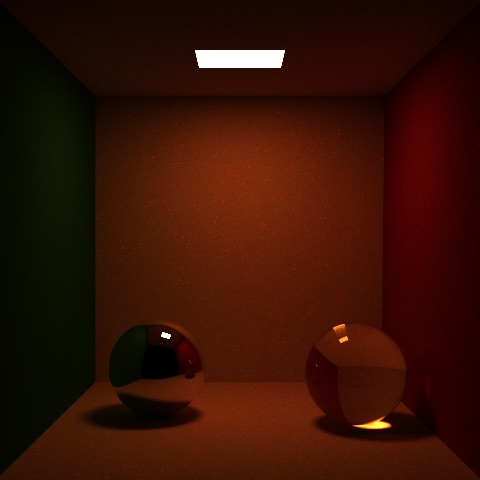
  PPM100, SPPM100

  PPM500, SPPM500

Fixed time

PPM(16 sample/pixel) 170 iteration 82.5s

SPPM 1000 iteration 84.5s

  PPM170, SPPM1000

From the results above, we can find that SPPM is better than PPM no matter on speed and quality. In the fixed iteration part, we found that SPPM is around 5 times faster than PPM and images generated by PPM have more noise than SPPM. In the fixed time part, we can find one thing. With the same time, SPPM can run 1000 iterations but PPM can only run 170 iterations with 16 samples per pixel.

SPPM can have such good speed and quality because SPPM bared the anti-aliasing in mind. SPPM samples a new sample ray in each iteration and accumulates the radiance of the new hit point to the pixel. In this way, SPPM can take the radiance of many different sample ray and get the accurate radiance of the pixel. However, PPM only wants to get the accurate radiance of a ray. Thus, PPM has to do the anti-aliasing through sampling some rays of each pixels at the beginning and averaging their radiance in the end. This wastes many times to calculate accurate radiance of each hit point although we just want the correct average radiance value.

* PM v.s. SPM+ v.s. PPM v.s. SPPM

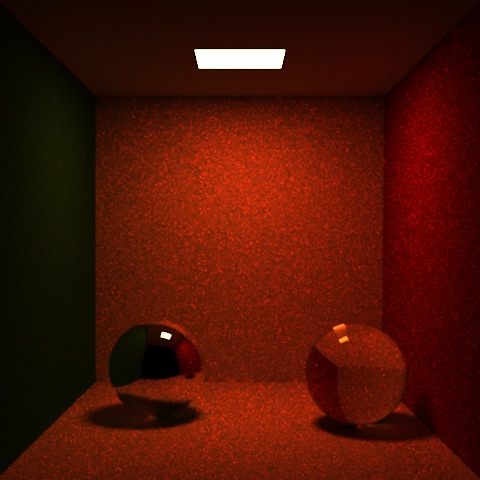
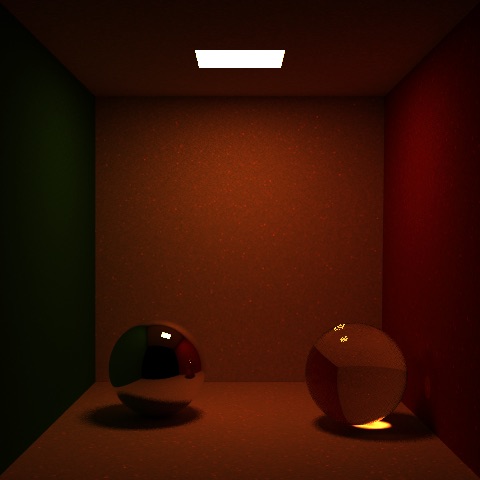
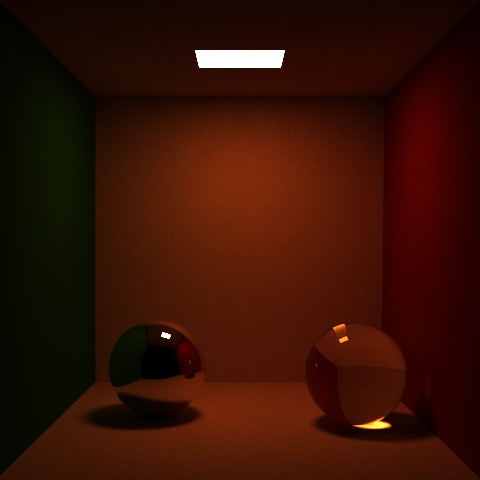
Fixed time

PM(16 sample/pixel) 280.5s

SPM+(24 sample/pixel) 276.5s

PPM(16 sample/pixel) 500 iteration 239.5s

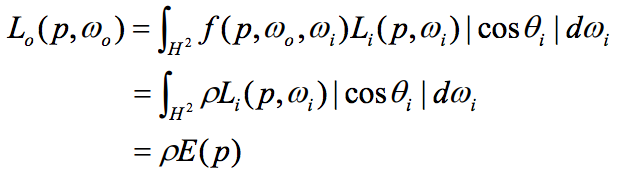
SPPM 3000 iteration 252.5s

  PM16, SPM+24   PPM500, SPPM3000

From the result above, we found that the newer the better. But one thing interests us. PM and PPM use the same sample rate. But PM got more noise than PPM. Besides, PPM ran many iterations but spend around the same time as PM and got better result. We originally thought that PPM needs many iterations to wait for radiance convergence. Thus, PPM should spend a long time to get a better quality than PM. But in the results above, PM and PPM use the same sample rate and spend around the same time, but PPM can runs 500 iterations and we can say that PM just only runs 1 iteration. We guessed that there may be two reasons: 1. final gathering spent too much time but didn’t have a better quality than 500 iterations PPM 2. there is a big improvement of performance in pbrt-v3 so that PPM can use the same time but runs more iterations.

**Obstacles and Difficulties**

The obstacles we met when implementing SPM+ is in the original version of Photon mapping, there is no usage of “irradiance”. However, the paper keeps using the term. After asking Prof. we suspected it might be of the same concept as irradiance caching, where assumption that diffuse surfaces are Lambertian is made.



So, we managed to spot where the red-circled part is computed in *photonmap.cpp* and do some modification to what *Christensen*’spaper proposed.