

Report: Computer Graphics Assignment3 – Sampler and BRDF in PBRT

Name: Ruofei Qian

UUN: s1872408

Overview

This assignment requires to implement a new sampler which apply dart throwing Poisson Disk Sampling method as an algorithm as well its relaxed version. After implementing this sampler, the next step is to implement a new material named “anisotropic Phong BRDF” accompanying with the BxDF in PBRT-v3.

After implementing the code, in this report, the implementation details and how to use them in the PBRT input file is necessary. Additionally, there should be comparison between the Poisson Disk method and other baseline samplers.

On the other hand, this report includes an experimental evaluation part that describe the differences between different combinations of samplers and integrators and provide a conclusion indicate that which combination is better.

Task 1

1 Dart throwing Poisson Disk Sampling method

The classes (.cpp and .h files) of relevant sampling methods should be included in the samplers folder and generally they show no differences with other samplers.

Except that in the CreatePoissonSampler function, this function define the acceptable parameters when using this sampler.

- params.FindOneInt("numPoints", 16) // default number of sample points per pixel, can be manually set in the input file
- params.FindOneFloat("minDist", 0.15) // default acceptable minimum distance between two sampling points, can be manually set in the input file

Important Algorithms are included in the "core/sampling.cpp" and its header.

1.1 Basic version

the main function is named **generatePoissonPoints2D** in the "core/sampling.h", it will create a new vector, **samplePoints**, to save the saved sampling points as well another vector, **processList**, contains all active points.

The First sample points is added into both lists randomly, then the whole process enter into a loop to search for the required number of points. In each loop, firstly, a random point will pop up from the **processList** (remove the point from process list at the same time):

```
Point2f point;  
if(!processList.empty()) point = popRandom(processList, rng);  
else point = Point2f(rng.UniformFloat(), rng.UniformFloat());
```

Then, it well try to generate a candidate point around the pop-up point. The radius is from the **minDist** to 2 x **minDist**.

```
Point2f candidatePoint = generateRandomPointAround(point, minDist, rng);
```

Then the candidate point will be compared with each point in the sample points list, whether the distance is greater than the ***minDist***, if it is, the candidate point will be add to both ***processList*** and ***samplePoints***.

```
for(int i = 0; i < 30; i++)
    if(isAcceptable(samplePoints, candidatePoint, minDist))
        samplePoints.push_back(candidatePoint);
        processList.push_back(candidatePoint);
```

The comparison will continue for a certain times. Each time the acceptable point will be added into the sample point list. Besides, at any time if the number of sampled points has reach the target number (given in the input file, the value of numPoints), the loop will be terminated.

Return samplePoints;

In the “core/sampling.cpp” file, there will be both PoissonDiskSample1D and PoissonDiskSample2D, it will just call the ***generatePoissonPoints*** function and add the generated points into a Point2f list.

1.2 Relaxed version

The only difference between the relaxed version and basic version is that the ***minDist*** in the relaxed version will be decreased by certain value if the algorithm continuously fail to find a candidate point for a certain times. However, the minDist in Relaxation should start from a relatively larger value. Currently, by default, if the program fail to find a candidate for 30 times, the ***minDist*** will decreased by 0.99 (***minDist*** = ***minDist*** * 0.99).

```
bool relaxed = false;
for(...)
    if(isAcceptable(samplePoints, candidatePoint, minDist))
        relaxed = true;
    ...
    if(!relaxed) minDist *= 0.99;
```

Compared with basic Poisson Disk sampler (using same numPoints: 200):

This is the basic:

This is the relaxation:



The noise in these two images generally are not visible, but the shape of the moon in the relaxation is more rounded. Besides, the shadow on the ground seems more natural in the relaxed version.

1.3 Use Poisson Disk Sampler in PBRT input file

Select Poisson Disk Sampler in input file is like the commands below:

```
#Sampler "relaxedpoissondisk" "integer numPoints" [200]
//relaxed poisson disk with sample per pixel value 200 and default minDist
#Sampler "poissondisk" "integer numPoints" [200]
//basic poisson disk with sample per pixel value 200 and default minDist
```

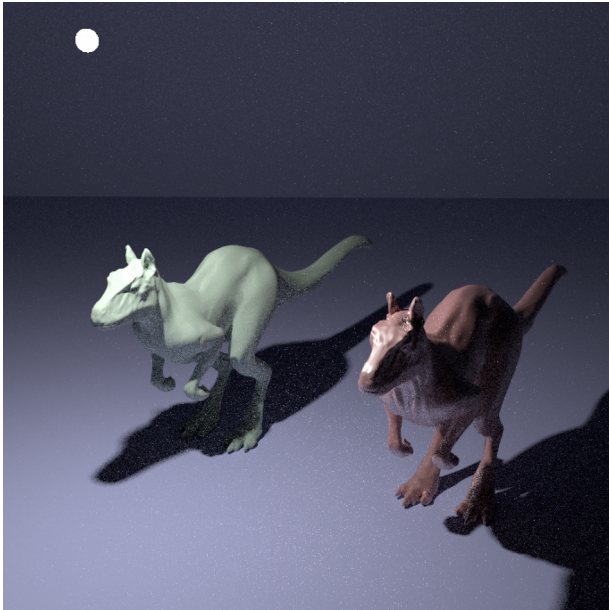
The default value of minDist is: $\sqrt{\text{numPoint}} / \text{numPoint}$
or define minDist manually like this:

```
#Sampler "relaxedpoissondisk" "integer numPoints" [200] "float minDist" [0.15], any invalid value like the
value less than 0 or larger than 1 will be detected and set to default value.
```

1.4 Comparison between the Poisson Disk Sampler and other baseline sampler

Here I choose the stratified sampler to compare with the Poisson Disk Sampler. In order to avoid other factors' influence, I set the value of samples per pixel to 200 for both and keep others as default values.

This is the result of stratified:



This is the result of Poisson Disk

It is quite clear that in the same circumstance, the stratified sampler made a lot more noises than the Poisson Disk sampler made. However, stratified method only requires several seconds to generate the result while Poisson Disk method's cost much longer.

2 Anisotropic Phong BRDF

2.1 Anisotropic Phong BRDF material

For this part I create a pair of anisotropic .cpp and .h files in the materials folder. The constructor receives R_d , R_s , ν_u and ν_v as parameters, all of which are coming from the pbrt input file like this:

```
Material "anisotropicPhong" "rgb  $R_d$ " [0.1 0.1 0.1] "rgb  $R_s$ " [.5 .5 .5] "float  $\nu_v$ " [10000.0] "float  $\nu_u$ " [10000.0]
```

- R_s : a colour / spectrum / RGB used to specify the specular reflectance at normal incidence.
- R_d : a colour / spectrum / RGB used to specify diffuse reflectance
- ν_u / ν_v : two phong-like exponents, control the shape of specular lobe

The classes should implement two functions: ***ComputeScatteringFunctions*** and ***CreateAnisotropicMaterial***:

- the ***ComputeScatteringFunctions*** receives `SurfaceInteraction *si`, `MemoryArena &arena` as parameters to allocate memory and evaluate the given four parameters.

- the **CreateAnisotropicMaterial** then receive the given values as well, calling the constructor to create a new anisotropic material by adding relevant values into a TextureParams.

2.2 Importance sampling the anisotropic Phong materials

To actually implement this new material model, then we should create a new class that extends the BxDF class. In my implementation, this class is embedded in the “core/reflection.cpp” and its header file. Its constructor receive 6 parameters for its use: Spectrum &Rd, &Rs, const Float nu, nv and const Vector3f &dpu, &dpuv, it should also implement the following virtual functions:

- Spectrum f(const Vector3f &wo, const Vector3f &wi) const;
- Spectrum Sample_f(const Vector3f &wo, Vector3f *wi, const Point2f &u, Float *pdf, BxDFType *sampledType) const;
- Float Pdf(const Vector3f &wo, const Vector3f &wi) const;

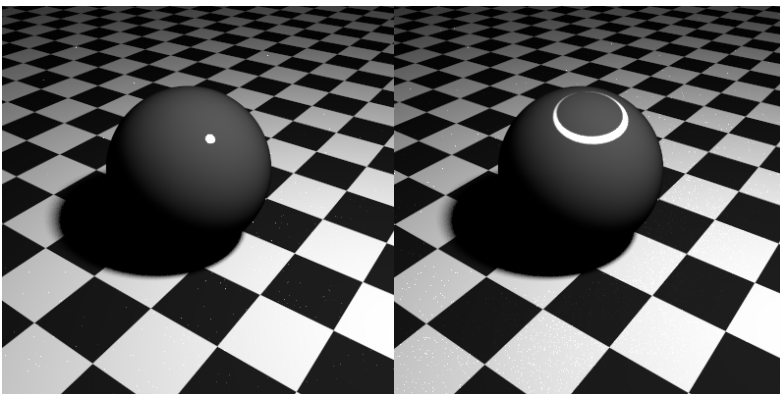
The Spectrum f() function is used to calculate the sum of the diffuse and specular value, the relevant formula is given by the provided paper published by M. Ashikhmin:

“The model is a classical sum of a “diffuse” term and a “specular” term.
$$\rho(k_1, k_2) = \rho_s(k_1, k_2) + \rho_d(k_1, k_2)$$

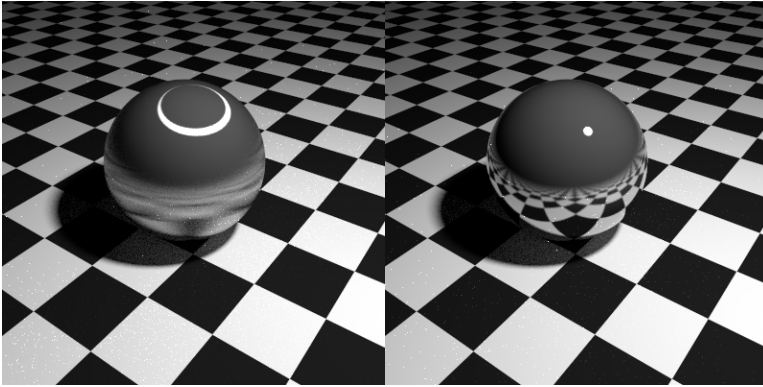
The Spectrum Sample_f() and Float Pdf() functions are used to do the importance sampling. My implementation for these two functions are based on the resources provided in this link:
<https://www.csie.ntu.edu.tw/~cyi/courses/rendering/pbrt-2.00/html/classAnisotropic.html>

2.3 Differences between rendering with or without importance sampling

For the given pbrt input file, without implementing these two functions, the result will be similar to the following images:

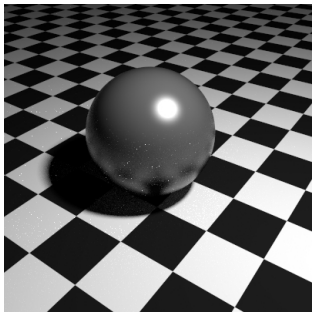
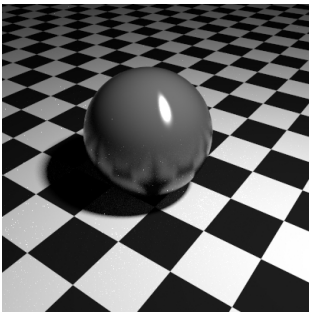
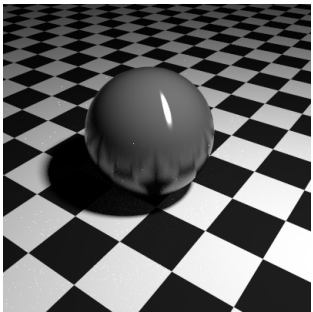
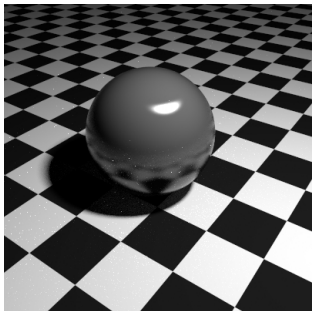
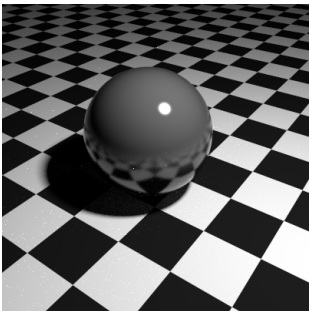
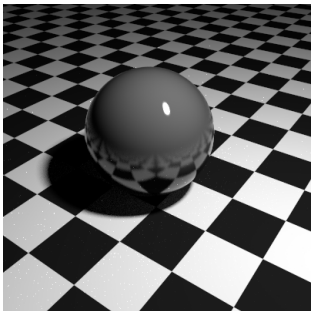
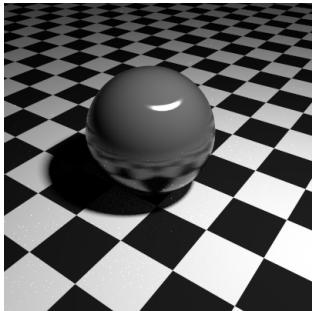
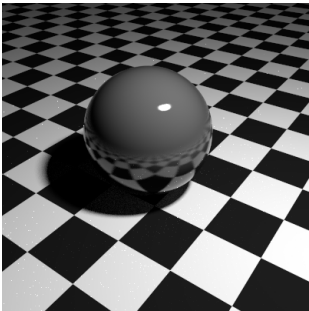
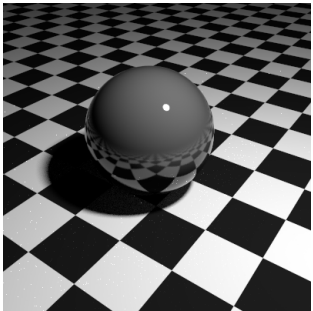


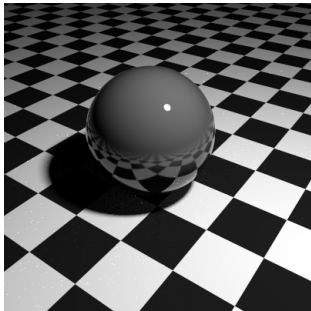
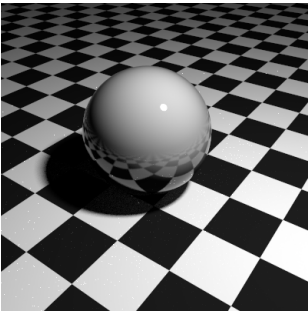
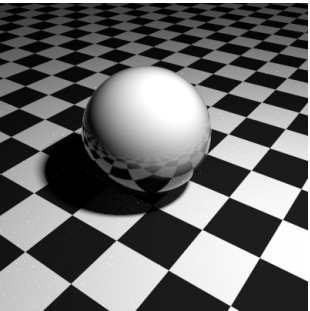
After implementing these two functions, the result will be like this:

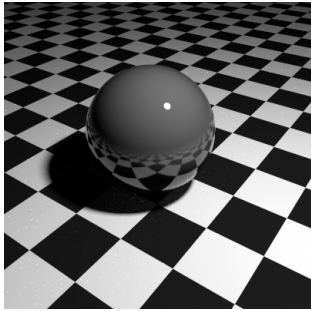
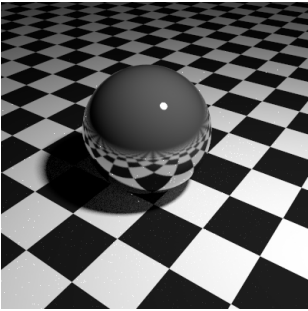
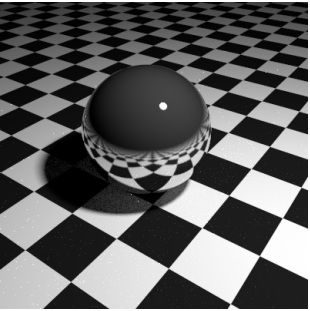


Other than adding reflectance under the half bottom of the sphere, to some extent it will also reduce some random noises.

2.4 Grid to show how the different parameters change will have influence on the behaviour


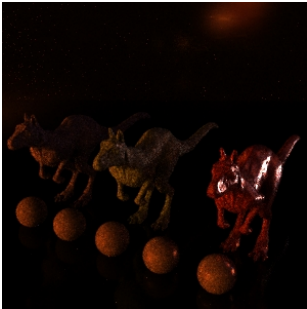
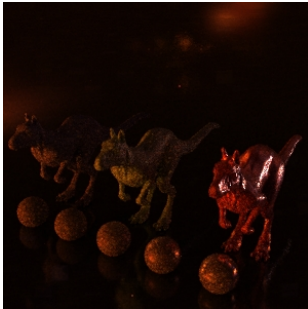
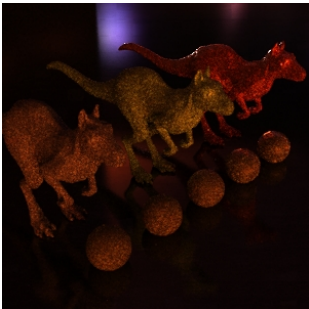
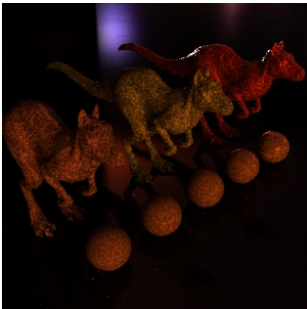
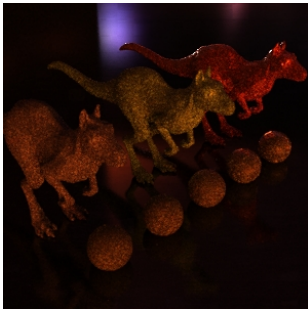
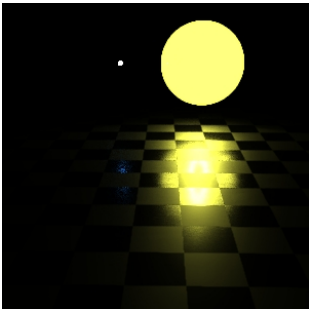
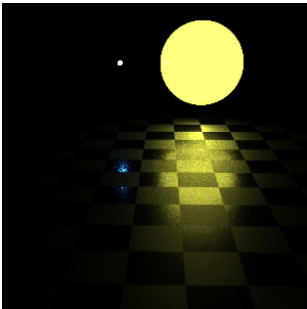
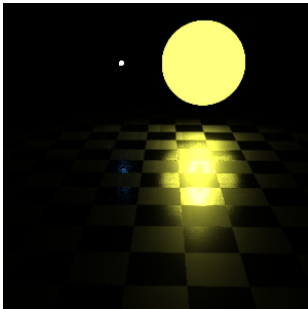
	Nu = 100	Nu = 1000	Nu = 10000
Nv = 100			
Nv = 1000			
Nv = 10000			

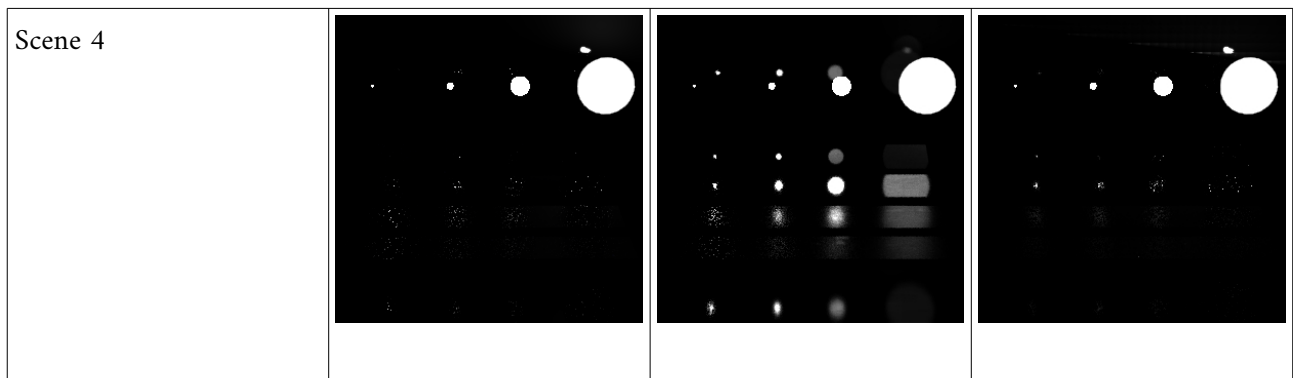
$N_v = 10000$ $N_u = 10000$	$R_d = [0.1 \ 0.1 \ 0.1]$	$R_d = [0.4 \ 0.4 \ 0.4]$	$R_d = [0.7 \ 0.7 \ 0.7]$
			

$N_v = 10000$ $N_u = 10000$ $R_d = [0.1 \ 0.1 \ 0.1]$	$R_s = [0.1 \ 0.1 \ 0.1]$	$R_s = [0.4 \ 0.4 \ 0.4]$	$R_s = [0.7 \ 0.7 \ 0.7]$
			

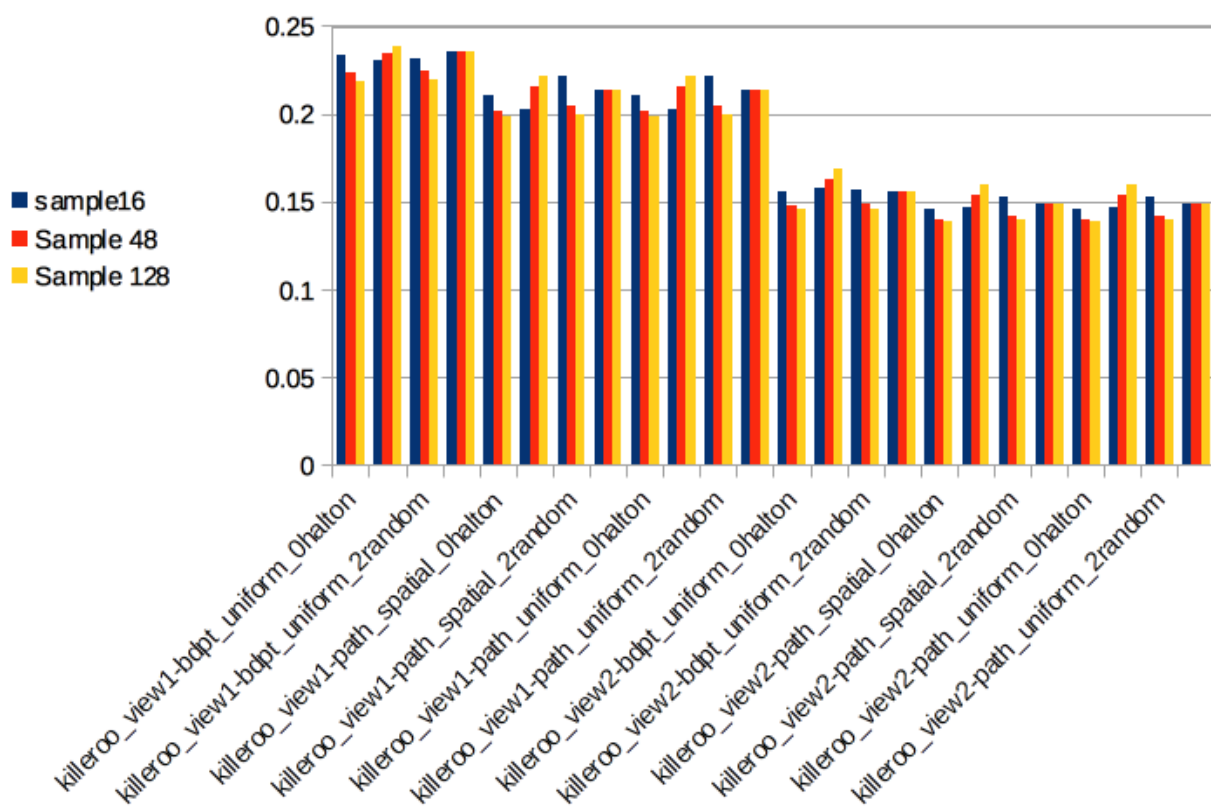
Task 2

1 Qualitative Comparison

	Integrator = path uniform	Integrator = bdpt uniform	Integrator = path spatial
Scene 1			
Scene 2			
Scene 3			



2 Convergence Plots



3 Results Discussion

In the combination, I provided 4 samplers: poisson disk, halton, random and stratified with 3 integrators: bdpt_uniform, path_spatial and path_uniform.

Here include the combination and comparison for the first 2 scenes.

The plot above shows the specific combination of samplers and integrators for four scenes provided in Tutorial 7. It is relatively clear to show some results.

- 1) when the number of samples increases, methods include halton and random's MSE is decreased obviously.
- 2) the performance of stratified method is really steady, generally show no fluctuation when the number of samples changed.
- 3) with the increasing number of sample points provided, the poisson dart method shows no obviously change as well, and some times the MSE value become larger.
- 4) Generally, the best performance always take place between random and halton no matter the sample points is relatively large or small.
- 5) The performance of all the combination are much better in scene 2 when compared to the scene 1.

Compare and contrast all the combinations above, halton sampler accompany with path integrator obtain the best performance while random sampler with uniform path integrator can gain a similar relatively low MSE value as well. This performance can be observed both when number of samplers are low(16) and higher(128).

Summary

In conclusion, when the number of sample points are not really high (may less than 200), some basic samplers and integrator combinations may show a higher correctness and effectiveness than the Poisson Dist Sampling method.

However, as in the implementing period, while the required number of sampling points per pixel increased and if some guard functions and algorithms that ensure the method will be able to obtain the targeted sample points number, provided with longer processing time, the result of Poisson Disk will be better than other methods especially random.

Reference

1 <https://www.csie.ntu.edu.tw/~cyu/courses/rendering/pbrt-2.00/html/classAnisotropic.html> Anisotropic Class Reference, pbrt-2.00

2 Michael Ashikhmin and Peter Shirley. An anisotropic phong brdf model. J. Graph. Tools, 5(2):25–32, February 2000.