CSC4140 Assignment VII

Computer Graphics

May 1, 2022

Ray Tracing II

This assignment is 8% of the total mark.

Strict Due Date: 11:59PM, May 1^{st}, 2022

Student ID: 118010141

Student Name: Jingyu Li

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

Signature: JINGYU LI

Contents

1 Overview

This assignment aims to realize a path-tracing renderer, add more features and finish a series of tasks. Ray-tracing technique is vital in computer graphics field since it compensates the weakness of rasterization: the global effect is not satisfied, and hard to deal with the case where light bounces more than one time. Ray-tracing is slow but can obtain a high-quality realistic picture. It usually works off-line instead of real-time rendering. The main mechanism for path-tracing is that we start from camera, focusing on a pixel on image, and cast a ray to a light source (might be an object), where the light bounces a few times, and we calculate the attribute (radiance, color) on the pixel according to the hit-point on the light source. The ray-tracing process really consumes a huge amount of CPU time and patience!

more complicated materials, environment lights, or depth of field effects

Based on the code of homework 6, we put additional features in our renderer to make it more powerful. I choose part 2 to finish. Starting from implement BRDF for microfacet material, we then realize environment light effect, with some statistical knowledge. In the last part, thin-lens camera is simulated and we can adjust aperture size or focus distance to obtain different depth of field effects. This is the road-map of this assignment.

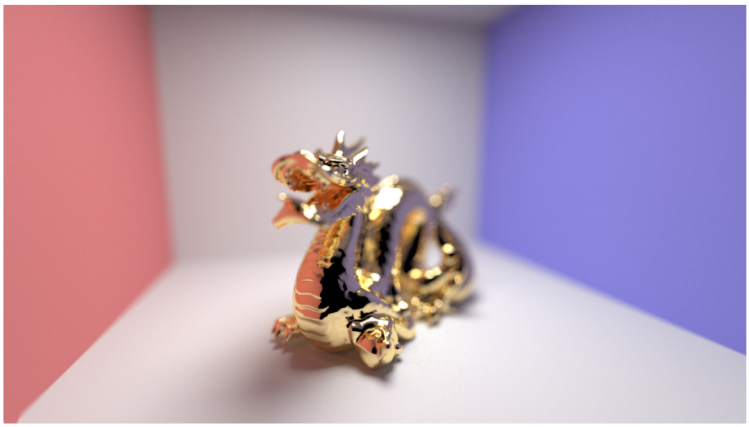

Figure 1: Golden dragon

2 Part 2: Microfacet Material

2.1 Design and Implementation

Between part 1 and part 2, I choose part 2 - microfacet material to implement. Microfacet material can be seem to be formed as many little mirrors. In this part we mainly implement this BRDF formula:

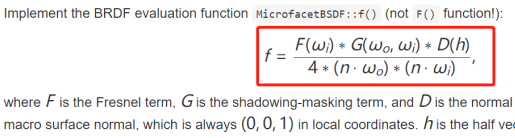

Figure 2: microfacet

2.2 Results and Analysis

Show a sequence of 4 images of scene CBdragon_microfacet_au.dae rendered with α set to 0.005, 0.05, 0.25 and 0.5. The other settings should be at least 128 samples per pixel and 1 samples per light. The number of bounces should be at least 5. Describe the differences between different images. To change the α value, I opened the .dae file and search for microfacet and then run the program. The results are shown as follows.

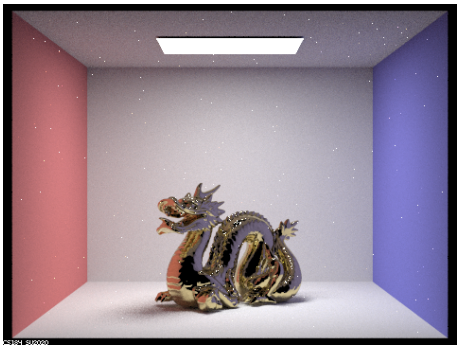

Figure 3: CBdragon alpha 0.005

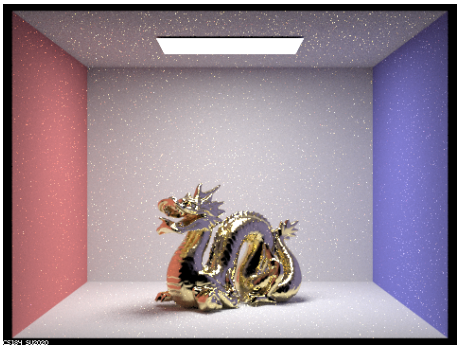

Figure 4: CBdragon alpha 0.05

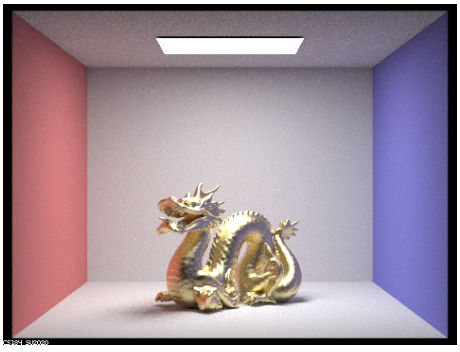

Figure 5: CBdragon alpha 0.25

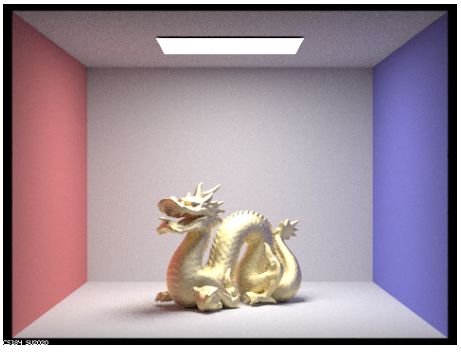

Figure 6: CBdragon alpha 0.5

As can be seen from the comparison, the α value represents the smoothness of the surface. The smaller α is, the smoother the macro surface will be. In other words, the macro surface tends to be diffuse whenα is small and glossy when α is large. When α equals to 0.5, the dragon exhibits the best view effect, where the body is pure golden and looks beautiful.

Show two images of scene CBbunny_microfacet_cu.dae rendered using cosine hemisphere sampling (default) and my importance sampling. The sampling rate is fixed at 128 samples per pixel and 1 samples per light, the same as previous one. The number of bounces should be at least 5.

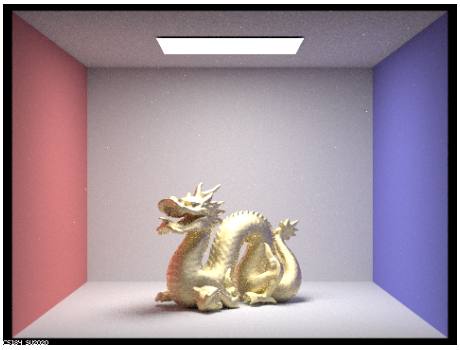

Figure 7: CBdragon alpha 0.5 cosine

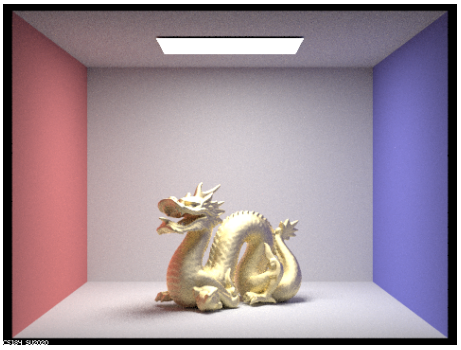

Figure 8: CBdragon alpha 0.5 importance

Compare these two images, there are not so many differences and nearly the same (I guaranteed that the code settings are different). The importance sampling may generate a little bit better result with less noise than cosine hemisphere sampling.

Show at least one image with some other conductor material, replacing eta and k. Here I choose **AL**, screenshot as follows. I have looked up values for real data in $https://refractiveindex.info/?shelf = mainbook = Alpage = Rakic$.

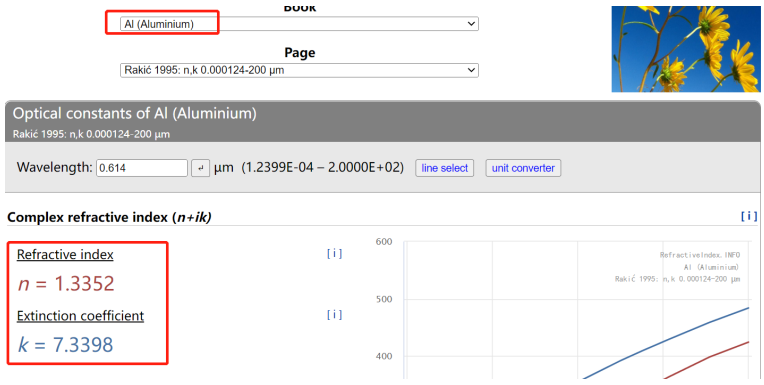

Figure 9: AL

```
<microfacet>
  <alpha>0.5</alpha>
  <eta>1.3352  1.0109 0.68955</eta>
  <k>7.3398  6.6157 5.6471</k>
</microfacet>
```

Figure 10: AL parameters

Here is the rendered result

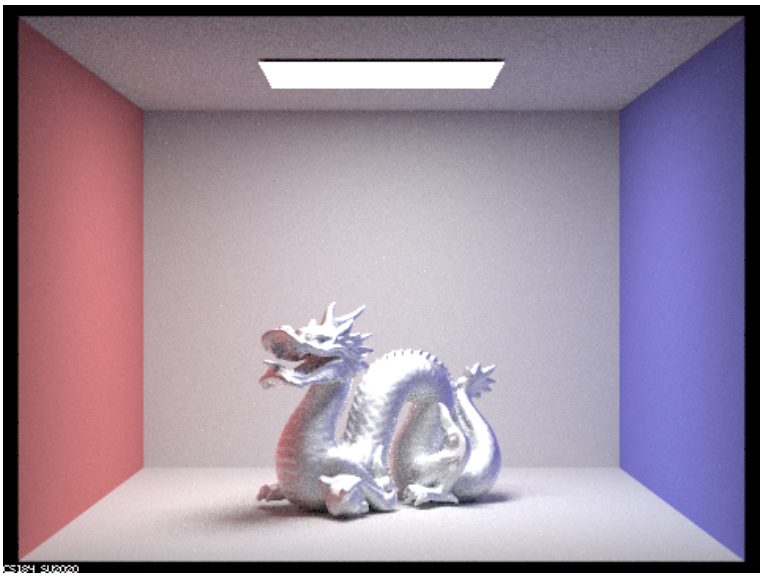

Figure 11: dragon al

Perfect! It's quite like aluminum material!

3 Part 3: Environment Light

3.1 Design and Implementation

The /exr file I use in this part is **doge.exr**. I pick this .exr file to use for all sub-parts here.

Environment light is a new type of infinite light source. An environment light is a light that supplies incident radiance from all directions on the sphere. It's unlimited. The source is thought to be "infinitely far away" and is representative of realistic lighting environments in the real world. For example, the sun shines all the things in nature. As a result, rendering with environment lighting can be quite striking.

The intensity of incoming light from each direction is defined by a texture map parameterized by two angles, which will be implemented in the code. With environment light we can render realistic items put under the sun, like grass or a cute bunny.

3.2 Results and Analysis

Show the probability_debug.png file for the .exr file I am using (doge.exr), generated using the save_probability_debug() helper function after initializing the probability distributions. The image is shown as follows. The distribution is reasonable and similar with that in the homework document.

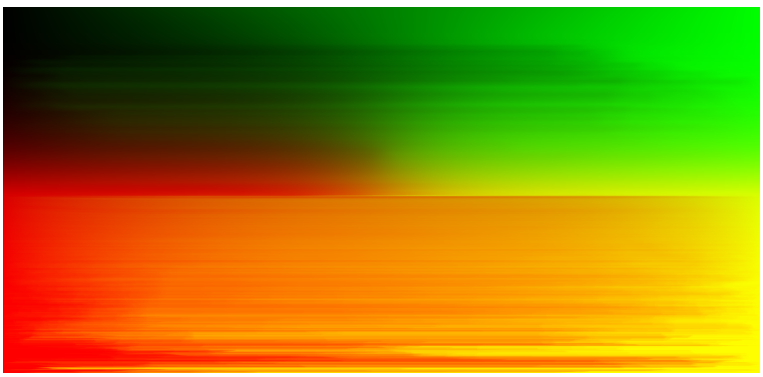

Figure 12: probability debug

Use the bunny_unlit.dae scene and my environment map doge.exr file and render two pictures, one with uniform sampling and one with importance sampling. Use 4 samples per pixel and 64 samples per light in each. Compare noise levels. The results are as follow.

As we can see, under the current setting, importance sampling performs better than uniform sampling. The noise level is higher for uniform sampling, and there are more black dots, though it's hard to find. Importance sampling can suit the requirement better. Much like light in the real world, most of the energy provided by an environment light source is concentrated in the

Figure 13: bunny unlit uniform

Figure 14: bunny unlit importance

directions toward bright light sources. Therefore, it makes sense to bias selection of sampled directions towards the directions for which incoming radiance is the greatest. So importance sampling is better.

Use a different image (recommend bunny_microfacet_cu_unlit.dae) and my environment map doge.exr file and render two pictures, one with uniform sampling and one with importance sampling. Use 4 samples per pixel and 64 samples per light in each. Compare noise levels. The setting is the same as previous part.

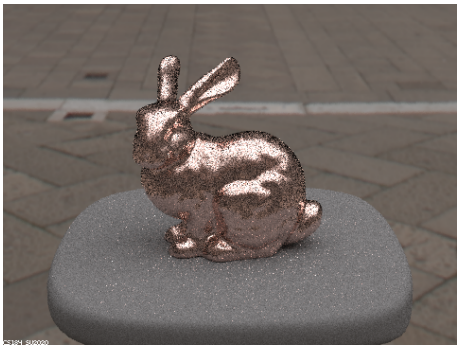

Figure 15: bunny micro unlit uniform

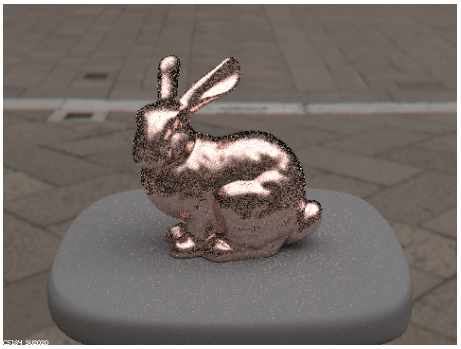

Figure 16: bunny micro unlit importance

As we can see, under the current setting, both images have a certain number of noises, but the difference is smaller than that in the previous material. Importance sampling still performs better this time. The reduced difference is partly because microfacet material includes many little mirrors and diffuse the light equally, so no need to importance sample the light. But we can see the surface is smoother in importance sampling, which requires some good eyesight!

4 Part 4: Depth of Field

4.1 Design and Implementation

The pinhole camera generates the image directly, all the lights go through the middle point and cast on the pixels, so all positions in the image are clear. Everything is in focus. However, no depth and no blur effect and less radiance for pin-hole camera. Thin-lens camera makes use of a len with no thickness. The light will pass through all the range in the len and refract to some places on the image. There are more radiances coming in and one can adjust the focal distance and aperture size to have different light exposure. The real cameras, including human eyes, are actually modeled as lenses with finite apertures.

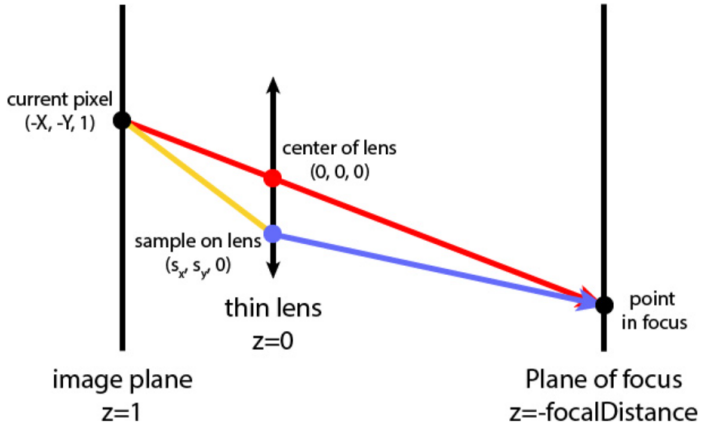

Figure 17: thin len

4.2 Results and Analysis

For these subparts, we recommend using a microfacet BSDF scene to show off the cool out of focus effects you can get with depth of field! So here I use the classic golden dragon, it's beautiful!! Under setting specified by: $./pathtracer-t4-s256-l4-m12-fCBdragon\_microfacet\_au\_camera.png-r19201080-b1.23-d4.56../dae/sky/CBdragon\_microfacet\_au.dae$

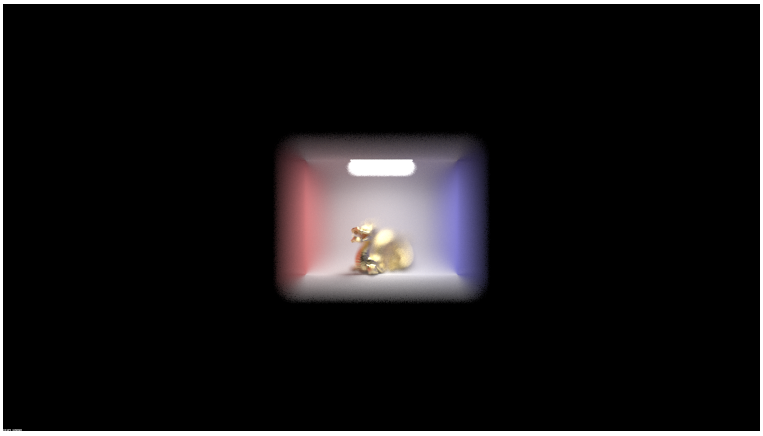

Figure 18: Microfacet BSDF

Show a "focus stack" where you focus at 4 visibly different depths through a scene. We use -d to adjust the focal depth (4.5, 4.75, 5, 5.25). The aperture size is 1 (-b).

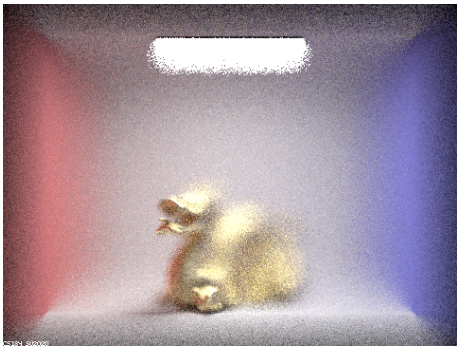

Figure 19: dragon depth 4.5

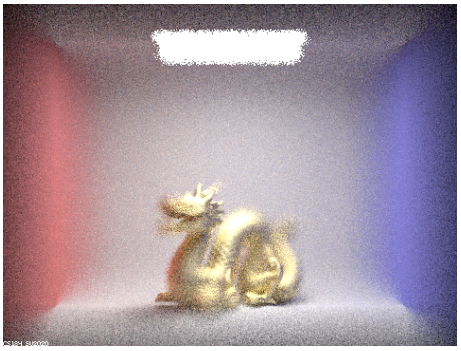

Figure 20: dragon depth 4.75

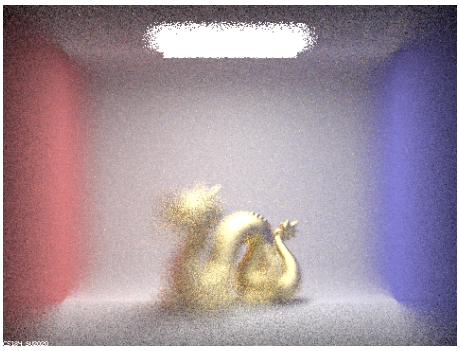

Figure 21: dragon depth 5

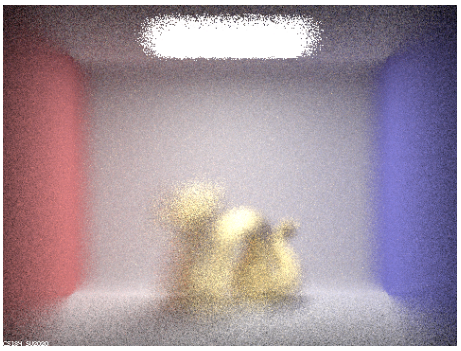

Figure 22: dragon depth 5.25

As can be seen, adjusting the focal distance generates different clearness. Only the proper focal distance, scene distance, and camera distance can provide a clear golden dragon image (the best is around 4.75 depth). The clear part transits from dragon's head to its body. The depth of view is changing.

Show a sequence of 4 pictures with visibly different aperture sizes, all focused at the same point

in a scene. Now the -d is 5 (the same focal distance), and the aperture sizes are 0.8, 0.9, 1, 1.1, respectively.

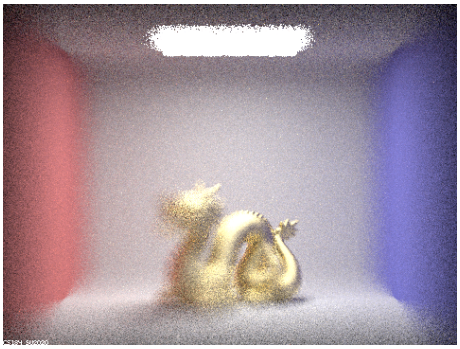

Figure 23: dragon aperture 0.8

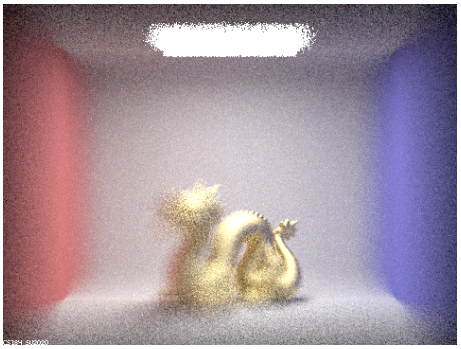

Figure 24: dragon aperture 0.9

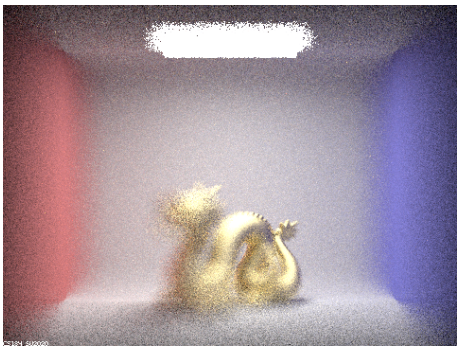

Figure 25: dragon aperture 1

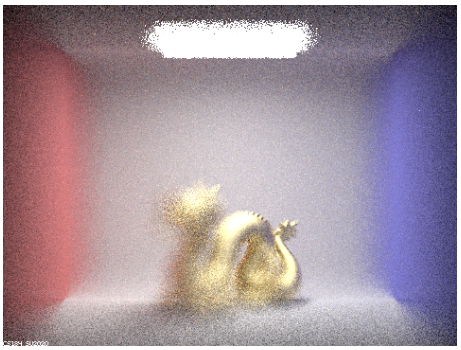

Figure 26: dragon aperture 1.1

As we can see, the effects are similar, with different aperture size, the image gives out different clearness. The depth of view for different dragons are different. But the difference is not that much.

5 Problems and Solutions

In this section, I will share some problems/bugs/eventful debugging experience when finishing this assignment.

(a) Write y * w + w as y * (w + 1) error!

(b) write normal multiply as division!

(c) calculate cdf as pdf!

(d) fail to differentiate pdf_envmap and envMap->data

(e) confuse x-y coordinate

6 Execution

(a) Run **sh ./compile_run.sh** to compile and run

(b) Render with fewer samples reduces the time.

(c) Many results are not shown but the graph can be rendered properly.

(d) Use script file to run the experiment.

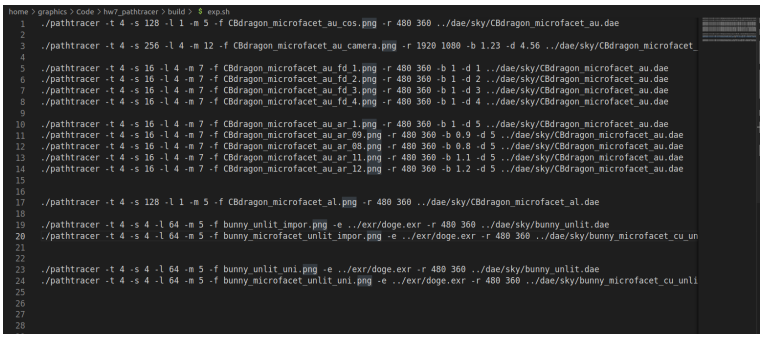

Figure 27: A sample script

(e) Read the code for more details

7 Summary

Here I will summarize what I have learned when writing this assignment.

(a) Great comprehension of ray tracing in computer graphics. For example, microfacet, environment light, camera.

(b) Solve programming problems and have eventful debugging time. Although the workload is pretty pretty heavy!!

(c) Object-Oriented programming paradigm, use class to encapsulate.

(d) Document reading and understanding, much material!

(e) Meaningful time on debugging and coding skill improvement.

(f) Meaningful discussion with peers.

(g) Time Scheduling. I should really start earlier.

(h) Report writing. A high-quality take-away write-up for future review.

(i) By the way, the document and code comments are detailed and clear, Kudos!

That's all.