TINGE

Task 6

Improvements and Niceties

Released: 18 April

Welcome to the beginning of the end. It has been a good three months of effort that all of us have put into this little adventure of ours and soon, we will be able to showcase our accomplishments (and flex) to the other DCs, the work that we have done. But before that, it is imperative that we make our renders look better and ensure we don't wait an eternity for the renders to appear! This is a very good article that summarizes all of our discussions so far.

This assignment has been tailored into 3 sections, and accordingly **we have divided you into 3 groups on WhatsApp.** Also, when you submit your work in a Pull Request, **we will not accept it, unless everything is well documented!**

§1 Realistic Scenes

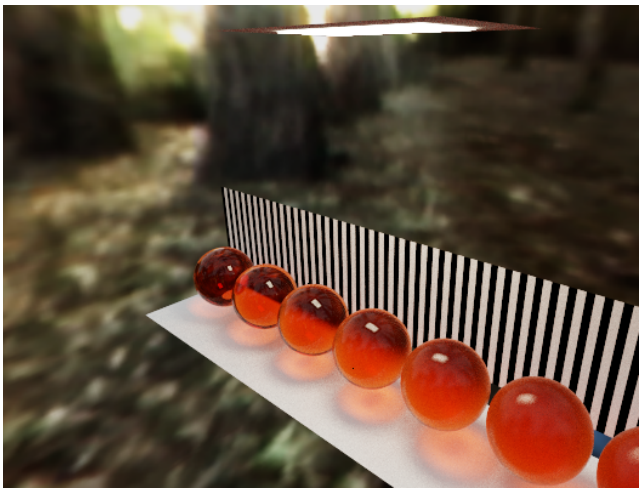

Figure 1: A rough dielectric material

§1.1 Materials

Create two material models called:

- **Dielectric** - Implement the Cook-Torrance model for dielectrics. Keep in mind that glass is a dielectric as well :) You will see a mixture of all the material models you have seen before.
- **Conductor** - Conductors behave more or less like the ideal metallic material that we had implemented before. The only thing differentiating our ideal metallic material from a physical conductor is the fact that the refractive index of the metal is highly frequency dependent and the fresnel "shine" of the metal appears to be of a different color than the base color of the metal itself.

§1.2 Lighting

Implement environment mapping in the Renderer. We will be using equirectangular HDRI maps as our environmental maps. Read up on Cube Mapping to get an idea of how you would project the map to a skybox.

Figure 2: HDRI map

Figure 3: Corresponding Skybox

Modify the functions `Renderer::env_map(const std::string& envmap_file_path)` and `Renderer::env_light(Vec3 sky_color)` functions accordingly in the Renderer class to implement the same.

Also modify the functions `Renderer::illuminance` and `Renderer::thread` such that they use the environment map / sky color gradient lighting accordingly.

§2 Why is it all so slow!

You may have noticed that our renderer has started to become very slow - taking nearly a minute to render fairly simple scenes. We will be looking at some techniques to counter that and make our processes efficient.

We aren't speeding up the time required to compute a sample. We are simply making each sample a better representation of the final render so that we can get a good enough image at very few samples. The cost per sample may increase but the number of samples needed to get the same quality of image would decrease.

§2.1 Importance Sampling

Having discussed the importance of importance sampling, it is now time to go ahead and actually implement it. We will be importance sampling in two fronts: the material bounce direction and in taking direct lighting contributions.

§2.1.1 Sampling Materials

Implement importance sampling in sampling the direction w_i in estimating the L_r integral term in the rendering equation for the Ideal Diffuse and Cook-Torrance material wherever random sampling is needed. You may need to collaborate with the other team which is implementing Cook-Torrance.

§2.1.2 Sampling Lights

You cannot always guarantee that a randomly sampled w_i direction will always end up hitting a light source eventually. So, we sometimes directly send out a ray w_i directed towards a randomly chosen emissive body. However, this would lead to overexposure of the surface since this is an artificial ray that we are sending out. We are in essence sampling more rays in the direction of the light source (thereby importance sampling in a PDF directly proportional to the projected area of the light source over the hemisphere of the surface point). Implement this!

§2.2 Rusian Roulette

Read up this nice article which describes the techniques of Russian Roulette and Splitting which can be used to terminate rays early on if we predict that the illuminance contributed by the ray is insignificant.

Implement relevant changes in our renderer class to incorporate Russian Roulette.

§3 Niceties

§3.1 3D Models

Ashwin has started working on a class called Mesh which are used to represent an aggregate of triangles which collectively represent a 3D Model. However, this is somewhat inefficient. Each triangle will have its own transformation matrix (frame) and if our model has N triangles then we will be doing $N \times \#(Rays) \times \#(Samples)$ matrix multiplications which will be extremely expensive! Instead, since the entire model will have a single transformation matrix, we can represent a 3D model as follows:

- Has a frame of its own (for the model itself, not for each triangle)
- All triangles are already transformed by the frame to world matrix before starting the rendering process, this is to avoid matrix multiplications per ray later on
- Has a bounding box for the model

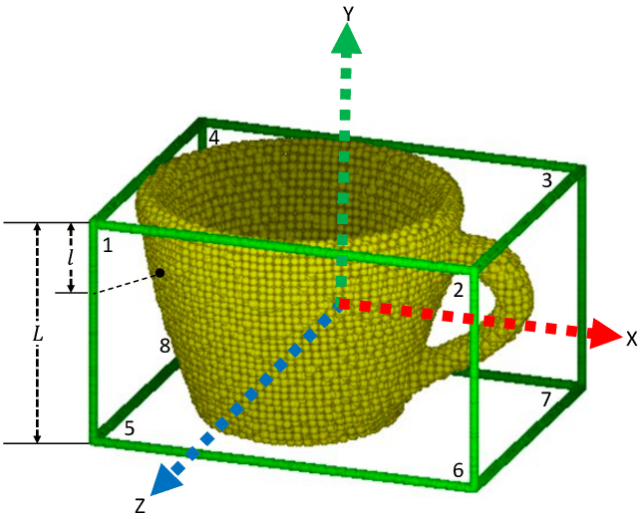

Figure 4: Bounding box

- We first test the ray against the bounding box. If it hits the bounding box, then we check against individual triangles. Note that triangles are already represented in world space and are not in frame space, so we are saving on matrix multiplications now.

Use the `obj_loader` library in our repository to load 3D models into the scene. In the file `scene_generator.cpp` we have already been loading a 3D model of the box. Swami had written that code. Take inspiration from it.

§3.2 Denoising our Image

Our image looks somewhat grainy when you zoom in close enough. This is because we are taking finitely many samples in the Monte Carlo Estimator and retroactively claim it to be the true image. In order to obtain the true value of the integral, we would have to wait for all eternity.

A good compromise is to choose a reasonable amount of samples, but then apply some sort of denoising algorithm to output a pretty image. Keep in mind that the denoised image is not the "true image" that we are looking for, but a filtered out (modified) one which looks prettier in comparison to our rendered output. That's all!

Read up on the median, mean and gaussian filters from resources online. Which filter do you think is more suited for our purposes?

Implement your chosen filter in the Renderer class before saving the image finally. Modify the render function such that it also takes in a boolean input to decide whether the user wants the image to be denoised or not.

Good luck on your code! Here's Master of Torture signing off~