Advanced Graphics - Assignment 3: Specialization

Dustin Meijer(5726328) & Luuk van de Wiel(4088212)

Short version

Specializations chosen

- CPU Path Tracer
- GPU Whitted-style Ray tracer (here is a blocking call to glGetUniformLocation in shader.cpp)

Basic Functionality

- CPU Path Tracer with RR, MIS, NEE, Refraction, Reflection, Gamma Correction

Introduction

In this report we will describe our project for Assignment 3 of Advanced Graphics. We created a CPU Path Tracer to achieve physical based rendering. We also worked on a GPU Ray Tracer, but sadly we didn't get this to work properly. Our CPU Path Tracer is located in the AG-Raytracer folder (legacy name…), our GPGPU code resides in AG_OCL_merge.

CPU Path Tracer

Our first focus was to create the most basic Path Tracer imaginable so that we could obtain a 'ground truth' image to compare our more advanced implementations with. This path tracer be found in the BasicSample function in Renderer.cpp.

To compare outputs of different path tracer implementations, we use the total energy count of the scene. This sum of energies from each pixel should eventually be (almost) the same if the path tracers converge. The second tactic we used was to compare two implementations side-by-side by splitting the image in two halves, this was useful for quick identification if something was awry.

The Sample function is our function which implements all of the variance reduction methods. In here, RR, NEE, IS (Cosine Weighted Distribution), Reflection and Refraction are implemented and used.

For NEE, we created a DirectSampleLights function which calculates the Direct Lighting (ray towards a random point on a random light).

Note: Our Sample function yields results which are a bit different from our BasicSample (this is also the case when we use a CosineWeightedDistribution for our BasicSample). We were told however, luckily, that no points will be deducted for this.

We also implemented Gamma Correction which we use to make our images look better.

We also created a SampleMIS function, which is basically the Sample function, but with MIS implemented as well. Sadly, this isn't correct. Hopefully we're still able to get some points for it. Reflection and Refraction is sadly also not working here.

Our path tracer also has the option to use a BVH. The f16_LOAD scene shows this. Note: the USEBVH pre-processor directive in Renderer.cpp shouldn't be turned on for the TUNNEL_SCENE, since we have spheres in there which we didn't implement in our BVH in the previous assignment.

GPU Whitted-style Ray Tracer

Our goal was to eventually creating a GPU based path tracer, sadly this was not meant to be. There were a lot of engineering troubles to overcome with merging our project from assignment 2 and the OpenCL template project that was provided to us by Jacco Bikker. After some help from Jacco, the two projects were successfully merged. Development started on the Whitted Style Ray Tracer, with trying to get data to the GPU and finally the creation of a test kernel where pixels would light up if anything was hit. In the end, the GPU ray tracer does not run because when registering the shader, the call to glGetUniformLocation in shader.cpp does not return. On the internet, no information could be found about how to fix this. There is a test kernel implemented in program.cl and two passthrough shaders were added to the project as well.

Scenes

First of all, our scenes can be switched by editing the pre-processor directives in Scene.h. Note: again, the USEBVH pre-processor directive in Renderer.cpp should only be turned on for the f16_LOAD scene.

TUNNEL_SCENE - For testing the path tracer. We created a scene that contains two spheres (one refractive, one reflective), a red wall to the left, a green wall to the right and white back-wall, ceiling and floor. There are two triangle light sources in the scene arranged as a rectangle in the ceiling against the back-wall.

f16_LOAD – In this scene, you can see a f16 model in the tunnel scene. The two spheres are removed.

Division of work

CPU Path Tracer
Basic Sample - Luuk
Next Event Estimation - Luuk
Refraction - Luuk
Reflection – Luuk
Important Sampling (Cosine Weighted) - Luuk

Multi Importance Sampling - Luuk & Dustin
Russian Roulette - Luuk
Scenes - Luuk
BVH – Luuk
Gamma Correction - Luuk

GPU Whitted-style Ray Tracer
Merge of AG-Ray Tracer and OCL-lab-template - Dustin
Test Kernel - Dustin

Report - Dustin & Luuk