Physically-based CUDA-accelerated Path Tracer

Yash Shah

October 2022

Description

The project involves the creation of a renderer, which utilises path tracing to create photorealistic images. The path tracing algorithm is notoriously computationally expensive. As a result, to achieve render times which are not excessively long, we exploit the embarrassingly parallel nature of the problem and use CUDA GPU acceleration.

Additionally, this path tracer should be able to realistically model a variety of materials in a physically accurate manner. This will be achieved by computationally modelling light transport. This means that the interaction of light with materials will be governed by a BRDF, i.e. Bidirectional Reflectance Distribution Function, which provides a weighting to a particular pair of incident and reflected light vectors. This BRDF can encapsulate how a particular material interacts with light, helping in simulating properties such as diffuse and specular reflection.

A cornerstone of the problem involves the calculation of the total incoming radiance at a particular point on an object. Similar to many ray tracing implementations, this path tracing approach is also "backwards", meaning that a ray is shot from the camera and followed into the scene, instead of shooting a ray from a light source and hoping it reaches the camera. This leads to a great increase in efficiency since most of the rays from light sources

will not reach the camera. However, this also means that simulating certain effects such as caustics becomes much more difficult.

The problem of calculating the total incoming radiance at a particular point is analytically impossible for even moderately complex scenes. As a result, we are forced to use a sampling approach, notably Monte Carlo Sampling. Using this approach allows us to get an estimate of the true value which improves with the number of samples. Increasing the number of samples reduces the noise we see in the result, but also increases the computational requirements. To help reduce the number of samples, the Monte Carlo approach is augmented with a technique called Importance Sampling. By guiding the sampling process according to where the BRDF values are high, we can reduce the variance in the estimator and thus achieve convergence faster (with fewer samples). The project also aims to integrate an existing denoiser to further reduce these sample requirements.

Additionally, there should be support for Constructive Solid Geometry, allowing the creation of complex objects from a set of primitives and union, intersection and difference operations, like in CAD software.

Lastly, as extensions, the project can also be further developed to support the rendering of meshes. The path tracer can also be extended to support more advanced features such as caustics, by integrating a bi-directional path tracing approach, where there is an initial forward pass, caching its result, in a technique called photon mapping. This is then followed by a traditional backward pass, as described earlier, where this time we also use the additional information gathered from the forward pass. A custom denoiser can also be developed, instead of using an existing one, if time permits.

Starting Point

The project will make use of some existing libraries. These include a unit testing framework, Catch 2, along with a static analyser, clang-tidy. These will ensure that the code is correct and of good quality. Finally, CMake will be used to maintain project structure and make compilation with these libraries easier.

Apart from the use of these libraries, the rest of the project will be done

from scratch. No code has been written prior to the start of the project. I do have limited knowledge of CUDA programming from the books "Programming Massively Parallel Processors" and "CUDA by Example" as well as knowledge about ray tracing from the first-year graphics course as well as the book "The Ray Tracer Challenge".

Success Criterion

The project will be considered a success if it can render a small variety of different shapes (such as spheres and planes), using path tracing and GPU acceleration. This result can be checked by creating a similar scene in a program like Blender and then comparing the similarity of the two outputs. Details for testing similarity include whether the shapes are correct, the lighting, shading, and shadows. It should also be able to output the resulting render to a PPM file.

Timetable

Weeks 1–2

4th–17th October

Performing some additional research around path tracing in areas such as BRDF and simulation of transparent materials. Also getting a refresher on CUDA and setting up the programming environment (integrating the libraries with CMake, working with a unit test framework (Catch2) and a static analyser (clang-tidy).

Weeks 3–4

18th–31st October

Implement a canvas and support exporting the result in PPM format. Support ray-sphere intersections, basic lighting and shading and be capable of rendering a very basic scene.

Weeks 5–6

1st–14th November

Add support for simple shadows (requires implementing a feature to see if the ray from the intersection point to a light is blocked). Allow multiple shapes - will need refactoring of the code to abstract away what a shape is.

Weeks 7–8

15th–28th November

Add support for Constructive Solid Geometry. Also, allow the rendering of patterns.

Weeks 9–10

29th November–12th December

Begin writing the dissertation, including a detailed description of the project, the PPM format, how intersections are calculated, implementation of the camera, calculation of shadows, Constructive Solid Geometry and patterns. Avoid programming in this period.

Weeks 11–12

13th–26th December

2 week break.

Weeks 13–14

27th December–9th January

At this point, we have a ray tracer. We need to transform this into a path tracer by each intersection spawning multiple new rays. These new rays will be randomly generated and will be weighted using a BRDF. The results can then be added to get the radiance at that point.

Weeks 15–16

10th–23rd January

The previous work package is quite large and will likely take more than a fortnight. As a result, this work package is dedicated to finishing off the path tracer conversion as well as adding support for importance sampling. Add to the dissertation details about this, emphasising how the path tracing approach differs from the ray tracing one. *The success criterion should have been met now.*

Weeks 17–18

24th January–6th February

At this point, the path tracer is capable of rendering quite stunning images. However, support for transparency is still missing and this work package aims to implement this. Write about how light transport functions with transparency in the renderer in the dissertation.

Weeks 19–20

7th–20th February

Add support for area lights and thus soft shadows. Also, integrate the denoiser (most likely NVIDIA OptiX). Add information regarding this to the write-up.

Weeks 20–21

21st February–6th March

Produce some nice high-resolution renders, work on evaluation and continue writing the dissertation.

Weeks 22–23

7th–20th March

There should now be time left to implement some of the extensions. The first one is to support adaptive sampling so that areas, in which the noise is deemed to be low, get sampled less. This should increase the performance of the path tracer.

Weeks 24–25

21st March–3rd April

Support for meshes in the OBJ format can now be added.

Weeks 26–27

4th–17th April

Write and integrate a naïve custom denoiser.

Weeks 28–29

18th April–1st May

Write about the extensions implemented in the dissertation and submit a draft to the supervisor and Director of Studies.

Week 30

2nd–8th May

Make some final finishing touches to the project and dissertation.

Special resources

My project will require the use of my own machine. This is because the project uses CUDA-acceleration, requiring an NVIDIA GPU.

I will be doing my work on my own laptop equipped with:

- CPU: *Intel Core i7-9750H @ 2.6 GHz*
- RAM: *16 GB DDR4*
- SSD: *1 TB*
- GPU: *NVIDIA GTX 1650*

In case of failure of this machine, I have a PC at home, which I can bring to Cambridge, equipped with:

- CPU: *Intel Core i7-4790 @ 3.6 GHz*
- RAM: *16 GB DDR3*
- SSD: *500 GB*
- GPU: *NVIDIA RTX 2060 Super*

Additionally, I will be backing-up the code I write to my own remote GitHub repository. Back-ups will be made whenever any substantial code changes are made. The dissertation will be backed-up in the same repository as the code.

It may also possibly need the use of the university's HPC cluster for high-resolution renders, since these may take several hours otherwise on my personal laptop. The use of the HPC cluster will likely incur a cost that is not too substantial for me to pay for personally. If this resource cannot be secured, my PC should be capable of producing these renders. Although the HPC resources would be useful, they are not strictly necessary for the evaluation of the project.

Evaluation

The evaluation of the project will focus on several things. The first is how the renderer scales with increasing resolutions. This will involve rendering the same set of scenes with different resolutions and looking at how this

impacts the time taken. Also, we must explore how the renderer scales with increasing scene complexity, particularly the addition of more light sources. This will be evaluated in the same way as the resolution increasing.

Secondly, the evaluation must focus on noise. We can render a reference scene with an incredibly high sample count ($>10,000$ samples per pixel), without a denoiser. Then, we can render the same scene with varying number of samples and compare these to the reference scene using image quality metrics such as PSNR (Peak signal-to-noise ratio) and SSIM (Structural Similarity Index). Additionally, we can run the denoiser on these renders and again compare the image quality to the reference. We can also make changes to the RNG (Random Number Generator) and see if it impacts the image output.

Lastly, the evaluation will look at how other publicly-available renderers do with the same scene. This also involves comparing with rasterisation techniques. The comparison involves testing how long the render takes as well as the image fidelity.

Supervisor

Joseph March *(jgm45)*