Milestone #1: Research Related Work

Arturo Cuevas (acueva8), Michael Gulson (mgulson2), Nathan Sparacino (nathans5), Katie Stapleton (kes8)

The original CPU implementation by Kevin Beason is one of the most famous ray tracing algorithms most likely because the code is only 99 lines and only requires a CUDA extension. Some of the solutions go beyond just the CUDA library to use extensions like Visionaray, a library including classes for different materials and the formulas for how light behaves with them, and OpenCL. Our overall goal is to select the most simple but effective approach. We found four ports to CUDA that are explained further below.

First, Sam Lapere's implementation which can be found on his public blog and Github[1]. The blog is well documented with tutorials, prerequisites, references, and modifications to the original source code. The code was made using the K.I.S.S method as it is concise with only the modified functions explained. With the CUDA based code on his blog and the Visionarray integrated version on his Github, it will be interesting to find out the performance differences.

The next implementation found is by Matthias Moulin which he named "The Rosetta Small Path Tracing Project"[2]. Matthias also made changes to Beason's 99 line algorithm, and he applied the algorithm to multiple different languages and for both single-threaded and multi-threaded processes. Based on the time each one took to generate the same 1024x768 pixel image, it was clear the CUDA multi-thread process was the fastest at 7.89s with the next fastest, C++ multi-process, at 16.71s. Matthias did not specify what the changes were to Beason's algorithm in his documentation. When looking at this CUDA code it will be important for us to keep in mind the code was written for many different languages and implementations.

Another implementation deriving from the Beason 99 line algorithm is by Aaron Jenson.[3] Aaron chose to use CUDA as his programming language of choice over OpenCL because of better documentation and concise programming syntax, reducing from about 1450 to 750 lines of code. In the algorithm, Aaron limited the ray-tracing up to six bounces in the scene, it will take one ray (pixel) from the camera to track the color as it bounces around the image. The scene, lights, and camera can be dynamic because the sampling process reset itself. On a CPU-based smallpt, it takes 5 minutes to reach 200 samples on a 800x600 pixel window. The GPU-based smallpt takes 15 seconds to sampel 200 samples on a 1024x768 pixel window. Through more sampling, the image becomes clearer.

Stefan Zellman implemented Kevin Beason's smallpt sequentially in C++ utilizing the visionaray ray tracing template library sequentially and used clever methods to port the code to CUDA and work on a GPU. Using other provided data structures in the Visionaray template library, Stefan Zellman explains in his article how to implement smallpt sequentially in C++ and "port" the application to CUDA.[4] He converts the data structures to be more CUDA friendly using the function in the thrust/device_vector.h library, device_vector(). Stefan created a buffer and modified the kernel to handle the changes in data types. He also makes changes to some of the methods in the data structures to allow them to be viable on the GPU. Utilizing the data structures provided in the Visionaray template library the resulting code is remarkably legible and simple. On his computer the execution on the CPU took .244 s while on the GPU it took .033 using floats.

In closing we feel that the Stefan Zellman approach is the most robust and up to date with current libraries. We are going to follow Zellman's lead, but will keep the design free flow to combine other's results and our own ideas. End goal is to try as many approaches to learn as much as we can about parallel programming.

Milestone #1: Research Related Work

Arturo Cuevas (acueva8), Michael Gulson (mgulson2), Nathan Sparacino (nathans5), Katie Stapleton (kes8)

Citations:

1. https://raytracey.blogspot.com/2015/10/gpu-path-tracing-tutorial-1-drawing.html.

2. Moulin, Matthias. “The Rosetta Small Path Tracing Project.” Github, 2019, https://github.com/matt77hias/smallpt.

3. Jensen, A. (2014, May 13). PDF. San Jose. http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring14/jensen/180H-Jensen-SP2014.pdf

4. Zellmann, Stefan. “Porting Smallpt with Visionaray.” Medium, Medium, 17 Jan. 2018, https://medium.com/@stefanzellmann/porting-smallpt-with-visionaray-b1e2a755e6f8.