CIS565: Project 2: CUDA Pathtracer

Fall 2013

Due Wednesday, 10/02/13

NOTE:

This project requires an NVIDIA graphics card with CUDA capability! Any card after the Geforce 8xxx series will work. If you do not have an NVIDIA graphics card in the machine you are working on, feel free to use any machine in the SIG Lab or in Moore100 labs. All machines in the SIG Lab and Moore100 are equipped with CUDA capable NVIDIA graphics cards. If this too proves to be a problem, please contact Patrick or Liam as soon as possible.

INTRODUCTION:

In this project, you will extend your raytracer from Project 1 into a full CUDA based global illumination pathtracer.

For this project, you may either choose to continue working off of your codebase from Project 1, or you may choose to use the included basecode in this repository. The basecode for Project 2 is the same as the basecode for Project 1, but with some missing components you will need filled in, such as the intersection testing and camera raycasting methods.

How you choose to extend your raytracer into a pathtracer is a fairly open-ended problem; the supplied basecode is meant to serve as one possible set of guidelines for doing so, but you may choose any approach you want in your actual implementation, including completely scrapping the provided basecode in favor of your own from-scratch solution.

CONTENTS:

The Project2 root directory contains the following subdirectories:

- src/ contains the source code for the project. Both the Windows Visual Studio solution and the OSX makefile reference this folder for all source; the base source code compiles on OSX and Windows without modification.
- scenes/ contains an example scene description file.

- renders/ contains two example renders: the raytraced render from Project 1 (GI_no.bmp), and the same scene rendered with global illumination (GI_yes.bmp).
- PROJ1_WIN/ contains a Windows Visual Studio 2010 project and all dependencies needed for building and running on Windows 7.
- PROJ1_OSX/ contains a OSX makefile, run script, and all dependencies needed for building and running on Mac OSX 10.8.
- PROJ1_NIX/ contains a Linux makefile for building and running on Ubuntu 12.04 LTS. Note that you will need to set the following environment variables:
    - PATH=$PATH:/usr/local/cuda-5.5/bin
    - LD_LIBRARY_PATH=/usr/local/cuda-5.5/lib64:/lib

The projects build and run exactly the same way as in Project0 and Project1.

REQUIREMENTS:

In this project, you are given code for:

- All of the basecode from Project 1, plus:
- Intersection testing code for spheres and cubes
- Code for raycasting from the camera

You will need to implement the following features. A number of these required features you may have already implemented in Project 1. If you have, you are ahead of the curve and have less work to do!

- Full global illumination (including soft shadows, color bleeding, etc.) by pathtracing rays through the scene.
- Properly accumulating emittance and colors to generate a final image
- Supersampled antialiasing
- Parallelization by ray instead of by pixel via stream compaction (you may use Thrust for this).
- Perfect specular reflection

You are also required to implement at least two of the following features. Some of these features you may have already implemented in Project 1. If you have, you may NOT resubmit those features and instead must pick two new ones to implement.

- From scratch stream compaction (no Thrust).
- Additional BRDF models, such as Cook-Torrance, Ward, etc. Each BRDF model may count as a separate feature.
- Texture mapping
- Bump mapping
- Translational motion blur
- Fresnel-based Refraction, i.e. glass
- OBJ Mesh loading and rendering without KD-Tree
- Interactive camera
- Integrate an existing stackless KD-Tree library, such as CUKD (https://github.com/unvirtual/cukd)
- Depth of field

Alternatively, implementing just one of the following features can satisfy the "pick two" feature requirement, since these are correspondingly more difficult problems:

- Physically based subsurface scattering and transmission
- Implement and integrate your own stackless KD-Tree from scratch.
- Displacement mapping
- Deformational motion blur

As yet another alternative, if you have a feature or features you really want to implement that are not on this list, let us know, and we'll probably say yes!

NOTES ON GLM:

This project uses GLM, the GL Math library, for linear algebra. You need to know two important points on how GLM is used in this project:

- In this project, indices in GLM vectors (such as vec3, vec4), are accessed via swizzling. So, instead of v[0], v.x is used, and instead of v[1], v.y is used, and so on and so forth.
- GLM Matrix operations work fine on NVIDIA Fermi cards and later, but pre-Fermi cards do not play nice with GLM matrices. As such, in this project, GLM matrices are replaced with a custom matrix struct, called a cudaMat4, found in cudaMat4.h. A custom function for multiplying glm::vec4s and cudaMat4s is provided as multiplyMV() in intersections.h.

README

All students must replace or augment the contents of this Readme.md in a clear manner with the following:

- A brief description of the project and the specific features you implemented.
- At least one screenshot of your project running.
- A 30 second or longer video of your project running. To create the video you can use http://www.microsoft.com/expression/products/Encoder4_Overview.aspx
- A performance evaluation (described in detail below).

PERFORMANCE EVALUATION

The performance evaluation is where you will investigate how to make your CUDA programs more efficient using the skills you've learned in class. You must have performed at least one experiment on your code to investigate the positive or negative effects on performance.

One such experiment would be to investigate the performance increase involved with adding a spatial data-structure to your scene data.

Another idea could be looking at the change in timing between various block sizes.

A good metric to track would be number of rays per second, or frames per second, or number of objects displayable at 60fps.

We encourage you to get creative with your tweaks. Consider places in your code that could be considered bottlenecks and try to improve them.

Each student should provide no more than a one page summary of their optimizations along with tables and or graphs to visually explain any performance differences.

THIRD PARTY CODE POLICY

- Use of any third-party code must be approved by asking on the Google group. If it is approved, all students are welcome to use it. Generally, we approve use of third-party code that is not a core part of the project. For example, for the ray tracer, we would approve using a third-party library for loading models, but would not approve copying and pasting a CUDA function for doing refraction.
- Third-party code must be credited in README.md.
- Using third-party code without its approval, including using another student's code, is an academic integrity violation, and will result in you receiving an F for the semester.

SELF-GRADING

- On the submission date, email your grade, on a scale of 0 to 100, to Liam, liamboone+cis565@gmail.com, with a one paragraph explanation. Be concise and realistic. Recall that we reserve 30 points as a sanity check to adjust your grade. Your actual grade will be (0.7 * your grade) + (0.3 * our grade). We hope to only use this in extreme cases when your grade does not realistically reflect your work - it is either too high or too low. In most cases, we plan to give you the exact grade you suggest.
- Projects are not weighted evenly, e.g., Project 0 doesn't count as much as the path tracer. We will determine the weighting at the end of the semester based on the size of each project.

SUBMISSION

As with the previous project, you should fork this project and work inside of your fork. Upon completion, commit your finished project back to your fork, and make a pull request to the master repository. You should include a README.md file in the root directory detailing the following

- A brief description of the project and specific features you implemented
- At least one screenshot of your project running, and at least one screenshot of the final rendered output of your pathtracer
- Instructions for building and running your project if they differ from the base code
- A link to your blog post detailing the project
- A list of all third-party code used