CS 5220/6630 Final Project Proposal

Bryce Evans, Eric Gao, Eric Lee, Stephen McDowell

1 Introduction

For our final project we propose to utilize our knowledge of parallel systems to drastically increase the rendering time of a somewhat simplistic ray tracer. For CS 6630 we have been working with an educational ray tracer *Nori* written in C++, which has recently been ported to Java (named *Wakame*). To better suit the tools we have learned in CS 5220, we will be using *Nori* as the primary starting point for parallel improvement.

The project can be split into two realms:

1. Render something beautiful (CS 6630), and
2. Render something really fast (CS 5220).

The access to the cluster and specifically the Xeon Phis is greatly exciting, and we intend to focus on using the cluster for these purposes. See the *Compilation and Execution* section for expected challenges and pitfalls, and our proposed course(s) of action.

2 Rendering Intentions

In CS 6630, we have been discussing the benefits and tradeoffs of various schemes for solving (or approximating) the rendering equation. The more interesting of which fall under the broad category of *Monte Carlo Path Tracing*. The current assignment for CS 6630 focuses more on direct integration schemes (both light-biased and material-biased), with a finale of Multiple Importance Sampling that combines the two. Later in the semester, we will also investigate volumetric methods that enable the rendering of things such as fog or clouds.

For this project we have two primary objectives that we can focus on, and are currently investigating the level of difficulty and overall interest we as a group have in:

1. Caustics

   Caustics are attractive and important elements of any complex (as well as many simple) scenes that add substance and provide photo-realism. Of particular interest here would be caustics as produced from fluids such as water.

2. Volume Rendering

   Volume rendering is equally exciting in that it allows you to introduce things such as fog or clouds into the scene you are rendering, which provides interesting and immediate effects such as depth cues and/or a better feeling for the scale of the scene being rendered.

Initial research seems to indicate that a viable path to potentially enabling both at reasonable convergence rates would be to implement *Photon Mapping*. There are a wide range of resources out there on the topic, and we are actively researching this. Although alternatives such as *Metropolis Light Transport* exist, we are generally intimidated by the nature of some of these alternatives (and the mathematical complexity therein).

Another interesting area we are considering would be to focus on rendering volumetric caustics (e.g. for underwater finding nemo style scenarios).

In its current form, we need to do a little more research, come to a consensus as a group, and speak again with Professor Marschner to come to an conclusion on what the actual form of rendering we should pursue is.

3 Parallel Intentions

As discussed in lecture, Monte Carlo techniques fall under the category of embarassingly parallel. That said, we do believe that this is an excellent place to start in terms of layered parallelism. Meaning that having an embarassingly parallel algorithm is not a bad thing to want to offload to the Phi boards.

The greater concern of parallelizing something like a path tracer is thread divergence. Unfortunately, solving this is dependent upon the type of rendering algorithm we intend to implement. There are many interesting areas with plenty of publications, though, from GPU Path Tracing (for which the concepts at the very least will be translatable to the Phis), to Ray Bundling, and more.

A rather interesting graph based method described in *GPU-based Out-of-Core Many-Lights Rendering* (http://dl.acm.org/citation.cfm?id=2508413) is of great interest, as its methodology is quite amenable to porting to the Phis.

At a higher level, when considering rendering an image that is $N \times M$ pixels, blocking strategies similar to that employed by the `matmul-` exercise can be be explored to dispatch different sections of the image to be rendered in parallel or in stages (depending on where we seek to run this code – Node or Phi). We intend to employ careful diagnostics of this in conjunction with the parallelization of the solver to find the right balance.

4 Parallelization Stages

It is worth noting at this point that half of our group is currently using the Java version of the framework, and half of our group is using the C++ version. The first and most important stage is to create a unified C++ framework. Pending approval from Professor Marschner, we intend to merge the two immediately after Fall Break, since we will have submitted the current project for CS 6630 by then. Furthermore, we will need to tweak the framework a little to enable rendering without a display so that we can run it on the class cluster (this should not be extraordinarily difficult, but necessary nonetheless). We fully intend, though, to render test scenes with small numbers of samples to debug locally. Being able to see the results (dynamically) is extremely valuable, and we do not want to tie up the cluster resources until we believe that we are producing valid results locally.

After formulating a unified framework, we will begin doing analyses both on and off the cluster to examine the overall behavior of the ray tracer. Though we intend to focus on using the class cluster, we want to remain receptive to alternatives such as CUDA (since at least one of our group members is familiar with this form of parallelization). Analysis of local runs and keeping the original framework in tact will enable us to make a switch if things do not go well with the cluster.

HOWEVER, our primary target is the class cluster. In particular, with the current assignment for CS 5220, we are confident that we will have gained the necessary knowledge / skill set to effectively offload our integrator to the Phis.

5 Compilation and Execution

As we intend to target the Phis, we are generally not concerned with portability. That is, we intend to take full advantage of the sheer power of `icc`. On the other hand, the current build system of *Nori* has support for Microsoft Visual Studio (and its compiler...), Linux, and OSX alike. So our final submission will likely be in the form of

1. An optimized Makefile for `icc` for use on the cluster, and
2. A generic program that runs for any platform*

*depending on how things go, this may or may not be dependent upon said generic system having a CUDA capable device attached. Though this will take a little effort in manipulating the current `CMake` build system, in theory it should not be entirely difficult. The hotfix being use `CMake` and then modify the resultant Makefile to use `icc` and our favorite flags. It is quite reasonable to also produce two makefiles (somewhat similar to `matmul-`) in the case where `CMake` is run on a system that has `icc`.

This is all a long way of saying that, from a software engineering standpoint, we are grateful that Wenzel Jakob (the creator of *Nori*) has gone through such lengths to enable multi-platform support and fully intend to follow his good example by making our final submission as portable as reasonably possible. For example, we can easily surround items such as hand-vectorized intel code or `#pragma offload target(mic)` with conditional `#ifdef` statements :)

6 Summary

At a high level, we are all very excited to produce a sophisticated approximation to the rendering equation on the class cluster. We need to confer with Professor Marschner (before Fall Break) to help us come to a conclusion about what we should attempt to implement for the rendering side.

While in some senses the rendering and the parallelization thereof are quite separable, there are algorithmic nuances that must be considered before non-embarassingly-parallel techniques can be employed.