CS 488 Project Proposal

Dusan Zelembaba
20349855

November 12, 2012

Consulted: Marta Kryven

Purpose :

To create a unique, photo-realistic scene by further exploring ray tracing concepts shown in class.

Topics :

- Constrictive Solid Geometry
- Texture, Bump, Transparency Mapping
- Refraction
- Distributed Ray Tracing (Soft Shadows, Depth of Field)
- Anti-Aliasing
- Fractal Terrain

Statement :

The interesting scene I will render is an ant on a leaf with a background of a lake and some mountains. On the leaf there will be some raindrops and extra branches potentially could be added. To bring focus to the ant, the background will be blurred out.

In order to make this scene look crisp anti-aliasing techniques will be used to remove "jaggies". Stochastic sampling will be used to achieve this [**?**]. To make the scene photo realistic, texture mapping will be used on the leaf and branches. Additionally, bump mapping will also be used to give the objects more of a 3D texture. Furthermore, since we will need to map the leaf onto a primitive (and the leaf isn't a perfect sphere), we will need to implement transparency mapping that makes unmapped pixels transparent. As an added effect, rain drops will be placed on the leaf and refraction will be implemented to make them more realistic. To model the ant, constructive solid geometry will be used.

To further add to the realism, distrubuted ray tracing will be used to implement soft shadows and bring focus to the ant using depth of field [**?**]. Finally, the background of the scene will require implementing fractal terrain as described in the course notes [**?**]. As reflection was completed as an extra feature in A4 we will get reflections on the lake for free.

I believe this project is interesting because it will allow me to delve deeper into the aspect of this course I found most interesting – ray tracing. This project will be challenging because of the many different algorithms that will have to work together to achieve all the desired effects. Each feature individually might not be greatly challenging but putting them all together to get one final polished scene will be the biggest challenge. In particualr, efficiency will be a concern. Even though it is not listed as an objective, work will need to be done so that the final scene renders in a realistic amount of time. For example, distributed ray tracing will greatly increase the number of rays that need to be tracked, and when combined with the amount of polygons in fractal mountains, will cause a huge increase in render time.

Technical Outline :

This project is an extension of the A4 ray tracer that we implemented. Here we will describe the major changes that will be required to this ray tracer to implement the desired features.

First we give a brief overview of the current state of the ray tracer.

Code Layout

There are four files that the main parts of the ray tracer are implemented, a4.cpp, scene.cpp, primitive.cpp and material.cpp.

a4.cpp: Here lies the code that determines which rays to intersect with the scene, intersects them with the scene, and puts the resulting colour value into a .png file.

scene.cpp: This is where the code for hierarchical transformations is along with the Phong lighting calculations.

primitive.cpp: All our primitives and intersect functions are here

material.cpp: Just a Phong material is here that is used by scene.cpp to do its lighting calculations.

Extra Primitives

To model the ant, extra primitives will need to be added to primitives.cpp. This should be fairly straightforward as all we should need are cones and cylinders, both of which have quadratic equations as described in the course notes.

This will require new commands to the lua interface, gr.cone, gr.cylinder.

Anti-Aliasing

The idea behind stochastic sampling is that the eye is good a picking up patterns, but not as good at picking up noise. So, instead of sending rays through each pixel exactly, we offset the each ray by a random amount and send that ray into the scene instead. Hence, this should only require a small change to a4.cpp, where we choose what ray to intersect with the scene.

Refraction

Refraction will be implemented in a similar manner to how reflection was implemented in A4. Snell's law will be used to determine the recursive ray to shoot out [**?**].

A new parameter will need to be added to gr.material to specify the index of refraction.

Texture Mapping

This will mainly require changes to material.cpp. Since we will want to support texture mapping and bump mapping of the same object we will add these as members of the base material class that can be toggled on and off. For texture mapping, we will simply return the colour in the texture map if it exists. For bump mapping, scene.cpp will be updated to ask the material to pertrub the normal if need be.

As mentioned above, we will also have to implement transparency mapping. This will require a change to the intersect routines in primitive.cpp. Once a point of intersection is found, an extra test will be added to check if this point on the material is transparent.

This will require new commands to the lua language. Bump mapping can just be toggled so we will add material:bump(), and for texture mapping an image file will have to be supplied so we will add material:texture_map(string filename).

Constructive Solid Geometry

This is going to require some pretty large changes to scene.cpp and primitve.cpp. We will implement CSG as described in the course notes, so we will apply the boolean operations on the line segments being passed up the recursive tree. This will require changing primitive.cpp to return line segments instead of a point and require scene.cpp to handle sets of line segments and apply boolean operations on them. Note that transparency mapping will cause complications in primitive.cpp as we will also have to check if any points on the line segment are transparent, and potentially return a set of line segments.

The data structure we plan on using here is a `std::list<std::pair<int, int> >`. Here we will just store the "t" value represeting the intersection points (i.e. $poi = eye + t * ray$). This allows us to pass the list up without having to transform any points and `std::list` give us efficient insertion/deletion at any position in the list.

Distributed Ray Tracing

For soft shadows, a change to scene.cpp will be required to modify the way we determine if a point is occluded by another object. Instead of sending one ray to the light source we will send many rays according to a distribution function and average the information we recieve.

For depth of field, a change to a4.cpp will be required to distribute rays over the aperture instead of just a point.

Depth of field will require modifying the lua interface to specify where the focal point is. gr.render will be modified to include an optional argument specifying the focal point. If no focal point is specified, no depth of field effects will be created.

Fractal Mountains

This will be implemented as described in the course notes. A new primitive will be created that will create a polygonal mesh representing mountains.

As this will require many polygons that will slow down the ray tracer, we will first ”cheat” and take advantage of the fact that these will just be in the background. Instead of simulating a life-sized mountain, we will just create a tiny mountain that appears to be in the distance.

Once this is working, we will try and model a real mountain by breaking it into many polygonal meshes and relying on bounding volumes to keep the render time reasonable.

A new lua function will be created to specify a fractal mountain with the desired parameters.

Efficiency

The major hit to efficieny that is anticipated is the combination of distributed ray tracing and fractal mountains. The biggest help here will be effectively breaking down the fractal mountains into many polygonal meshes. However, if this isn't enough, we will try using multiple threads to decrease the render time. Since determining the colour of each pixel doesn't modify the scene at all, we can easily add multiple threads to concurently determine the colour of multple pixels.

Objectives:

Full UserID: dzelemba **Student ID: 20349855**

___ **1:** Refraction has been implemented.

___ **2:** Stochastic sampling has been implemented for anti-aliasing and before and after images are provided to demonstrate this.

___ **3:** Texture mapping has been implemented.

___ **4:** Bump mapping has been implemented.

___ **5:** Cones and cylinders have been added as extra primitives.

___ **6:** Multi-threaded rendering has been implemented.

___ **7:** Union, intersection and difference have been implemented for constructive solid geometry and a sample image showing combinations of these with different primitives has been provided.

___ **8:** Depth of field effects have been implemented and an identical scene with 3 different focal points is provided to demonstrate this.

___ **9:** L-system plants have been implemented.

___ **10:** A unique scene demonstrating the above objectives has been provided.

___ **A4:** Reflections were implemented as my A4 extra feature.