

CS334: Final Assignment

Out: March 19th, 2008

Back/Due: Demo Day, April 25th, 2008

Objective

Now that you have an understanding of basic interactive computer graphics, the focus of this final assignment is to give you the freedom to extend your knowledge in a particular area of interest to you. Below are two suggested individual projects in the areas of ray-tracing and geometric simplification. Alternatively, you may also suggest your own project (which if you are doing, you should have communicated to be by now). All projects will be demonstrated to a public audience on the due date. The demo includes a live presentation of your project/program and a short PowerPoint presentation explaining how it works. The time slot is to be determined but roughly 10 minutes per person.

Please choose one of the projects below and send me (and the TA) an email within one week (March 26, 10:29am) indicating which project you will be doing – if not, I will randomly assign a project to you.

As a prelude to choosing a project, I recommend you browse through the papers (mostly PDFs) at <http://trowley.org>, in particular for the SIGGRAPH, Eurographics, and Symposium on Interactive Graphics conferences. Browsing through the papers might inspire you with regards to options. Most, if not all, of the below projects have previous works published at the aforementioned venues. Depending on your interest and project objectives, I can point you to several specific papers which you can read for background information.

Below are guidelines for the projects – I encourage all to go above and beyond the descriptions given. Grading will be based on how well you *implement* the project and *demonstrate* it (live demo and PowerPoint presentation), your *understanding* of the general concepts, and the *features and functionality* of your program. In other words, if you do something but don't know how it works or how to show it off, you get serious points off. Furthermore, you must implement the project in its entirety and no downloaded or previous implemented code is permitted unless explicitly approved by the instructor (I am not trying to discourage using software libraries but rather trying to prevent you from getting credit from what it implements ☺).

On April 18th and in class (one week before demo day), each of you will describe briefly (~3 minutes), via a few PowerPoint slides you must email the instructors and TA by 11:59pm on the 16th, what your project is and will give a summary of the project status so far. The objective of this session is to give you feedback and to encourage you to plan ahead on your presentation which is **very important. This preliminary presentation will count towards your final assignment grade. You may of course setup an appointment with me or the TA at other times if you have questions.**

This final assignment will take effort and I vehemently encourage you to **start working on the assignment well before the due date** – i.e. today! As with the previous assignments, I prefer a nicely working, well-implemented and demonstrated project as opposed to a half-working buggy project with lots of half-baked complex features.

1. Ray-Tracer

A ray-tracing program provides a totally different way for you to generate rendered images. Ray-tracing uses a very simple method and easily supports very apparently-complicated effects. Unfortunately, the computational cost of ray tracing is usually very high. It may take from minutes to hours to render a single image.

As explained in class, a ray-tracer shoots imaginary rays from the viewer through each pixel of the image plane and bounces the ray off opaque objects in the scene. The color of the ray is the accumulation of the colors seen (optionally attenuated by distance) of the first N -reflections. A ray-tracer also supports a myriad of other effects such as transparency, inter-object reflections, refraction, and shadows.

This assignment consists in creating a ray-tracer for at least several simple object types (e.g., sphere, box, etc). The scene should be described via some file format that allows easy editing of the scene. The ray-tracer should at least support first-order reflections (this means if you have two shiny spheres next to each other, they will be able to see a reflection of each other on the surface of each other). A typical ray-traced image might be 256×256 or 512×512 pixels in size and each ray might bounce 1 or 2 times. This means $256 \times 256 \times 3$ or $512 \times 512 \times 3$ rays might be traced in the scene.

To accelerate rendering, a hierarchical spatial subdivision could be used to accelerate the ray-object intersection tests. For example, when using an octtree, rays are recursively checked for intersection against octtree boxes until a leaf node is reached. Then, testing is done against the object (or polygons) stored in the leaf node.

Additional features include supporting more object types (e.g., cone, cylinder, triangles, polygonal objects, etc), transparency, refraction, shadows, etc.

2. Simplification

The goal of simplification is to accelerate rendering by simplifying the objects and thus reducing the rendering cost. Simplification is usually done either by specifying a maximum allowable error from the original model or by indicating a desired performance and simplifying the model as necessary to achieve the goal. One way to accomplish this is to use a hierarchical spatial subdivision data structure as a simplification data structure. The simplification can be focused on single “objects” or can work for any collection of polygons and objects (e.g., an entire scene).

For example, using an octtree, you can implement a vertex clustering tree. The leaves of the tree contain one vertex per octtree node. Higher-level nodes can be used to represent a

cluster of points with a single point (e.g., the midpoint of the octtree node). The trick here is to maintain reasonable triangle connectivity.

Another option, is to replace each octtree node with a colored box that approximates the geometry inside the node. Thus a very simplified object will have a “boxy” look to it. An alternative is to use a hierarchy of spheres instead of boxes (if you believe spheres are better approximations to a group of geometry than boxes).

Your imagination might find other approximation schemes.

In all cases, you must be able to “extract” a version of the scene/object at some desired level of detail. The user should be able to interactively specify (1) an error tolerance (e.g, via a GLUI slider) and this should result in a particular simplification; or (2) the user should be able to specify a desired rendering performance (i.e., a desired number of polygons to render) and the system should extract a model of (at most) that number of polygons.

Bells and whistles include allowing the simplified object to move (e.g., bounce) in a world box so that the performance acceleration can be viewed, automatically choosing a level of detail based on distance to the viewer, nicely handling shading, lighting, and texturing effects, and visualization tools to help the observer understand the simplification process.

You may obtain/generate your own large models. I can also provide you with some large models and a program to generate arbitrarily large models procedurally. The object geometry is stored in a simple file format, called “.sim”. Countless file formats are possible, this is just one of them. The format is:

```
# this is a comment (textures and polygons can be intermixed)
texture <num textures>
filename0.ppm
filename1.ppm
filename2.ppm
polygon <num polygons>
<num verts> <tex index, -1 == no texture> [NORMALS] [RGB] [UV]
<x> <y> <z> <r> <g> <b> <u> <v>
...
<x> <y> <z> <r> <g> <b> <u> <v>
<num verts> <tex index, -1 == no texture>
<x> <y> <z> <r> <g> <b> <u> <v>
...
<x> <y> <z> <r> <g> <b> <u> <v>
...
```

The format allows you to specify textures and polygons and the position, normal, color, and texture coordinates per vertex. I will post example files .sim on the course website. In addition, you will be given an executable which you can run and generate a model of arbitrary size. The program is called “makepipes” and is to be used as “makepipe sim <x> <y> <z>” which generates a connected entanglement of pipes of <x> by <y> by <z>

pipe units. Try “makepipe sim 4 4 4 > pipes.sim” to get an initial model, then try increasing the size until your program cannot handle it.

3. Your Own Project

If you wish to propose and present your own final assignment, that is fine, **but you must provide a project description by March 26th when final project assignments are due.** Your project description should be one-page long and should include the following sections:

- a. title
- b. summary of the objective
- c. main components of the project to be developed
- d. what your demo will consist of

If you have more questions about an independent project, please ask the instructors or TA.

Demo Day

Final Assignment demo day is April 25th. Closer to that date we will setup a schedule via a democratic algorithm. Essentially, we will put up department-wide announcements, provide food and snacks, and have a demo-fest. Each student will present and demonstrate her/his assignment. You must also provide on that same day a CD containing your source code, binaries, project and presentation – program must be executable from the CD. No extensions, no late penalties, no late passes, and no exceptions to this due date will be given.

If you have more questions, please see myself or the TA.

Good luck!