15-462 Project 2: Geometry

Release Date: Tuesday, February 3rd, 2014

Due Date: Tuesday, February 18th, 2014, 23:59:59

Starter Code:
http://www.cs.cmu.edu/afs/cs/academic/class/15462-s14/www/project/p2.tar.gz

1 Overview

In this project, you will have the opportunity to learn and implement a simple shader followed by a subdivision algorithm. Specifically, you will use the loop subdivision algorithm to subdivide the mesh and modify the mesh you are rendering accordingly. This document, the subdivision lecture and chapter 9 of the Red Book (Version 1.1) will be extremely helpful as they should cover most of what you need for this assignment.

2 Description

In the previous project, you implemented a class to render 3D meshes. In this assignment, you will first render a 3D mesh with a shader applied to it, and then you will add a method that will subdivide a given mesh and make it more refined with each iteration of the subdivision.

Subdivision is a technique which aims to increase the quality of a mesh by generating additional vertices. Surface subdivision is a recursive process. The process starts with a given polygonal mesh and with each iteration, a refinement scheme is applied to the mesh. New vertices and faces are created based on the vertices around them. Some refinement schemes (like the one you will be implementing), then take the old vertices and modify them as well. This allows one to start with a very coarse mesh and procedurally create something much more smooth.

As always, we will direct you towards resources for help, but if you need further clarification or have a question, the best place to go is to the piazza page for this course (https://piazza.com/class#spring2014/15462/). For this particular assignment, the subdivision lecture and this document will be the most helpful resources. If you're interested in or would like a reference book for shaders, there is a companion to the Red Book: "OpenGL Shading Language" by Randi J. Rost, et. al.

3 Required Tasks

Input: The input is a virtual camera and a loaded mesh. Shaders are loaded in the program itself and are not given as input arguments in this assignment.

Output: The output should be a rendering of the mesh using the given camera and the specified shader program (see section 7.2). With each press of the 'y' key, the mesh should be subdivided using the loop subdivision algorithm. See the end of the document for some sample outputs.

For your program, you must:

- Use shaders to render the mesh with a mirror-like material.
- Subdivide each face of the mesh using the loop subdivision algorithm. This includes calculating the new positions and normals.
- If you are in 15-662, you must do at least one of the Extra Credit items detailed in Section 9.
- Submit a few screen shots of your program's renderings.
- Fill out `writeup.txt` with details on your implementation.
- Use good code style and document well. We *will* read your code.

At a minimum, you must modify `./src/scene/subdivide.cpp`, `./src/scene/project.cpp`, and `./shaders/material_frag.glsl`, though you may modify or add additional source files.

`writeup.txt` should contain a description of your implementation, along with any information about your submission of which the graders should be aware. Provide details on which methods and algorithms you used for the various portions of the lab. Essentially, if you think the grader needs to know about it to understand your code, you should put it in this file. You should also note which source files you edited and any additional ones you have added.

Examples of things to put in `writeup.txt`:

- Mention parts of the requirements that you did not implement and why.
- Describe any complicated algorithms used or algorithms that are not described in the book/handout.
- Justify any major design decisions you made, such as why you chose a particular algorithm or method.
- List any extra work you did on top of basic requirements of which the grader should be aware.

4 CMake

In order to ease the process of running on different platforms, we will be trying out CMake (http://www.cmake.org/) for this project. Here is a description from the CMake wiki:

> CMake is used to control the software compilation process using simple platform and compiler independent configuration files. CMake generates native makefiles and workspaces that can be used in the compiler environment of your choice.

4.1 How to use CMake

You will use CMake to generate the build system for your project. These are the instructions for using CMake on the GHC 5xxx machines (these will work for any Linux flavor and OSX as well). Consult the `README` for Windows intructions.

1. After untar-ing the project handout, run the following commands in the root of the project:

   ```
   $ cd build
   $ cmake ../src
   ```

   All build files are now in `build/`. You only need to perform this step once.
2. Write some code.
3. Build your code.

   ```
   $ cd build/
   $ make install
   $ cd ../
   ```

4. Run your code.

```
$ ./p2 scenes/bunny.scene
```

5. Return to step 2.

Warnings:

1. Your code must run on the GHC 5xxxx cluster machines. Do **not** wait until the submission deadline to test your code on the cluster machines.
2. If you run `make` instead of `make install` the executable will not be copied to the project root (from `build/p2/`). If you build your code and you aren't seeing your changes on screen then maybe you ran `make` instead of `make install`.

Keep in mind that there is no perfect way to run on arbitrary platforms. CMake does not install libraries or headers for you. If you experience trouble building on your computer, the GHC 5xxx machines will always work and we recommend you work on them.

4.2 Starter Code

It is recommended that you begin by first reviewing the starter code provided. We are providing you with a large code base to get you started and handle more mundane application tasks. The `README` gives a breakdown of each source file.

The program takes 1 required argument and 1 optional argument. The required argument is the filename of the mesh that will be subdivided. The optional argument is the filename of a cubemap. The starter code will load the mesh and cubemap for you. The starter code stores the mesh in a list of meshes in the `scene` class. You will only have one mesh in a scene for this project. The cubemap is specified by a folder name instead of a file name. The two provided cubemaps are stored in `./cubemaps`. By default, the starter code will load `teide`. We have also provided some sample shaders for you to run and experiment with in the `shaders` folder, and by default the starter code will load the `normal_` shaders as an example. You should change your code to run the `material_` shaders once you have them written.

You can move the virtual camera with '**wasd**' and the mouse, as before. The '**y**' key performs a single subdivision, and the '**f**' key takes a screenshot.

5 Submission Process and Handin Instructions

Failure to follow submission instructions will negatively impact your grade.

1. Your handin directory may be found at one of the two directories below, depending on which class you are enrolled in. Graduate students should be enrolled in 15-662 and undergraduates in 15-462.

   `/afs/cs.cmu.edu/academic/class/15462-s14-users/`*andrewid*`/p2/`.

   `/afs/cs.cmu.edu/academic/class/15662-s14-users/`*andrewid*`/p2/`.

   All your files should be placed here. Please make sure you have a directory and are able to write to it well before the deadline; we are not responsible if you wait until 10 minutes before the deadline and run into trouble. Also, remember that you must run `aklog cs.cmu.edu` every time you login in order to read from/write to your submission directory.

2. You should submit all files needed to build your project, as well as any textures, models, shaders, or screenshots that you used or created. Your deliverables include:

- `src/` folder with all `.cpp` and `.hpp` files.
- CMake build system.
- `writeup.txt`
- Any models/textures/shaders needed to run your code.

3. Please **do not** include:

   - Anything in the `build/` folder
   - Executable files
   - Any other binary or intermediate files generated in the build process.

4. Do not add levels of indirection when submitting. For example, your source directory should be at `.../`*andrewid*`/p2/src`, **not** `.../`*andrewid*`/p2/myproj/src` or `.../`*andrewid*`/p2/p2.tar.gz`. Please use the same arrangement as the handout.

5. We will enter your handin directory, and run `cd build && rm -rf * && cmake ../src && make`, and it should build correctly. **The code must compile and run on the GHC 5xxx cluster machines**. Be sure to check to make sure you submit all files and that it builds correctly.

6. The submission folder will be locked at the deadline. There are separate folders for late handins, one for each day. For example, if using one late day, submit to `.../`*andrewid*`/p2-late1/`. These will be locked in turn on each subsequent late day.

6 Grading: Visual Output and Code Style

Your project will be graded both on the visual output (both screenshots and running the program) and on the code itself. We will read the code.

In this assignment, part of your grade is on the quality of the visuals, in addition to correctness of the math. So make it look nice. Extra credit may be awarded for particularly good-looking projects.

Part of your grade is dependent on your code style, both how you structure your code and how readable it is. You should think carefully about how to implement the solution in a clean and complete manner. A correct, well-organized, and well-thought-out solution is better than a correct one which is not.

We will be looking for correct and clean usage of the C language, such as making sure memory is freed and many other common pitfalls. These can impact your grade. Additionally, we will comment on your C++-specific usage, though we will generally be more lenient with points (at least earlier in the semester). More general style and C-specific style (i.e., rules that apply in both C and C++) will, however, affect your grade.

Since we read the code, please remember that we must be able to understand what your code is doing. So you should write clearly and document well. If the grader cannot tell what you are doing, then it is difficult to provide feedback on your mistakes or assign partial credit. Good documentation is a requirement.

7 Implementation Details

As usual, the Red Book will be a useful resource, particularly the chapter on shaders.

7.1 Mesh and Scene Details

For this project we give you all of the code required to set up the rendering.

The code you have to modify for the shader part of the assignment is in `./shaders/*` and `./src/p2/project.{h|c}pp`.

The code you have to modify for the subdivision part of the assignment is in `./src/scene/subdivide.cpp` and `./src/scene/mesh.hpp`.

7.2 Shaders

Have you noticed that your Project 1 did not look particularly plausible? Have you compared the lighting quality between a game made in 2000 vs. a game made in 2014?

Some of it is certainly due to the geometries and lack of physical plausibility in traditional OpenGL renderings. In this assignment you will have some exposure to the essentials of 'Shaders', small pieces of code that allow you to modify the output of different stages of the OpenGL pipeline. A very large fraction of video game development and 3D animation production is dedicated to writing shaders to increase performance and visual quality.

In this assignment you will implement a very simple reflection shader. You are given a scene with a geometry and an environment map. This reflection shader will reflect a ray from the camera off of a point on the object to a point on the environment map. The shader will then sample the environment map at that point, in order to determine the reflected light color.

Specifically, you should modify `./shaders/material_frag.glsl` when implementing your reflection shader. We have already initialized the code to manage the required set up. The aim of this part of the assignment is to get you comfortable with writing shader code and to get you comfortable with GLSL. The algorithm is not too complicated, but you may have to put in some work to get your head around GLSL.

The starter code takes care of loading and compiling the shader for you. You can compile a shader by calling `shader->compile_from_file(vertShader, fragShader)`. If you are interested in how that process works, take a look at the `shader` class.

7.3 Loop Subdivision Algorithm

7.3.1 Overview

You will be implementing the Loop subdivision algorithm for this project. It is a two-pass algorithm: the first pass creates all the new points and triangles, and the second refines all the old points that were not created during the first pass. It is an *approximating* algorithm, which means that the existing vertices are modified during subdivision.

The Loop subdivision scheme can only be applied to triangular meshes, but this isn't really an issue since a mesh of arbitrary polygons can be transformed into a triangular mesh by splitting faces. Anyway, in this project, all meshes are already triangles.

First we'll define several terms relevant to the algorithm. *Odd vertices* are those that are added during the subdivision, while *even vertices* are those that existed prior to the subdivision. *Boundary edges* are edges that lie on the "boundary" of the mesh. More specifically, if there exists a triangle edge ab that is shared with no other triangle, it is a boundary edge. The vertices a, b can be described as being boundary vertices, *even if* they are also endpoints of other, non-boundary edges. *Interior edges* are the complementary set: any edge shared by at least 2 triangles. Any vertex that lies on *only* interior edges is an interior vertex.

Boundary and interior edges are handled differently in order to preserve the features of the boundary. It's also possible to treat some interior edges as boundary edges to preserve features and prevent smoothing. These are called *crease edges*. However, you do not have to deal with crease edges in this project.

Note: You may assume for this implementation that all edges are shared by at most two triangles. The algorithm only works if this condition holds, which does holds for all meshes we provide.

7.3.2 Odd Vertices

As a reminder, odd vertices are the vertices added during the subdivision. They are added on the first pass. We add a new vertex v for every edge ab, where a, b are vertices. We first compute the position for v, then split the edge to make 2 new edges, av and vb. The new vertex v is computed as a linear combination of the vertices on the surrounding triangles.

If ab is an interior edge, equation 1 gives us the position of v, where the 2 triangles sharing the edge ab are (a, b, c) and (b, a, d). The upper-left picture in figure 1 illustrates this.

$$v = \frac{3}{8}a + \frac{3}{8}b + \frac{1}{8}c + \frac{1}{8}d \tag{1}$$

Equation 2 gives us the position of v for when ab is a boundary edge. The lower-left picture in figure 1 illustrates this.

$$v = \frac{1}{2}a + \frac{1}{2}b \tag{2}$$

This step divides each triangle of the mesh into 4 new triangles.

7.3.3 Even Vertices

Even vertices, the vertices that existed before the subdivision, are handled on the second pass. For each even vertex v we will compute a new position v' as a linear combination of some of its neighbors.

Note: The neighbors here refer to the old neighbors of the even vertices, not the new neighbors formed in the first pass. Let $\mathcal{N}$ be the set of neighbors of v, which are all vertices that share an edge with v.

Equations 3 and 4 describe how to compute v for interior vertices. The upper-right section of figure 1 illustrates this; v and all its neighbors are used in the weighting.

$$\beta = \frac{1}{|\mathcal{N}|}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{|\mathcal{N}|}\right)^2\right) \tag{3}$$

$$v' = (1 - \beta|\mathcal{N}|)v + \beta\sum_{u\in\mathcal{N}} u \tag{4}$$

For boundary vertices, we only use the points on the boundary itself. Equation 5 describes the computation, where a and b are the 2 neighbors that lie on the boundary edges of v. The lower-right section of figure 1 illustrates this.

$$v' = \frac{3}{4}v + \frac{1}{8}a + \frac{1}{8}b \tag{5}$$

7.3.4 Normals

For the purpose of this assignment, you can use the same algorithm to calculate the new normals of the vertices. In general however, subdivision algorithms may use a separate formula to compute tangent vectors of adjacent faces and then just cross them to get a new normal which is computationally less expensive than averaging the normals.

7.4 Suggested Sequence

We suggest that you implement the assignment in the following order. Note that the subdivision portion is worth a far greater percentage of the grade than texturing. However, do not neglect texturing if you get stuck on subdivision.

1. Familiarize yourself with the changes to the given mesh structure.

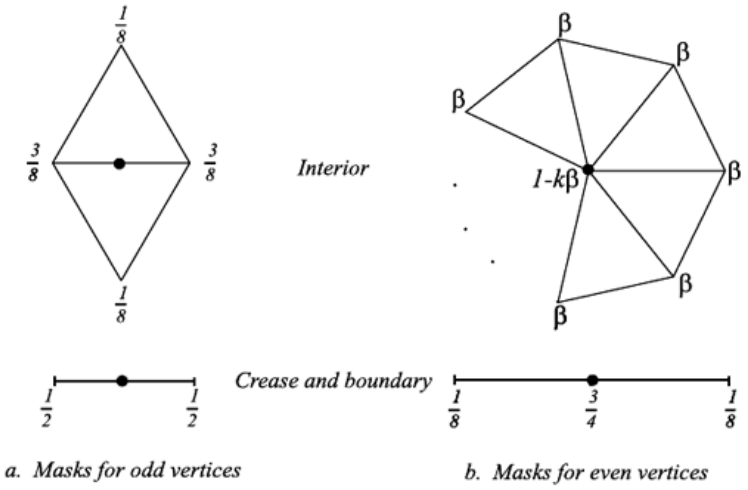

Figure 1: Illustration of the algorithm

2. Implement your shader.
3. Implement the interior case of the odd vertices for the loop subdivision algorithm.
4. Implement the interior case of the even vertices for the loop subdivision algorithm. At this point, you should be able to subdivide the cube, torus and bunny which have no boundaries.
5. Implement the boundary case for odd and even vertices. At this point, you should be able to subdivide all of the meshes (including the open cube and the stegosaurus).

Note: For grading, we will normally do around 3 - 4 subdivisions on simple models and 2 subdivisions for more complex models like the stegosaurus. The subdivisions need not be uber fast, but should be of reasonable speed (less than a minute each).

Note: We will also be comparing the time performance of each student's implementation and choose a few of the fastest for the final project showcase.

8 Masters Students

For students enrolled in 15-662, there is an extra requirement for this lab. You must fulfill at least one of the extra credit requirements listed below. Extra credit may be awarded for impressive achievements beyond the mandatory choice. Please document these in your write-up.

9 Extra Credit

Any improvements, optimizations, or extra features for the project above the minimum requirements can be cause for extra credit. Extra credit is generally awarded for impressive achievements beyond the project requirements, at the discretion of the graders.

Ideas may include but are not limited to:

- Implement some more interesting shaders.
- Fast subdivision (e.g. 6th iteration of stegosaurus in under 2 seconds).
- Adaptive loop subdivision. Subdividing a flat mesh should not change the mesh in any way. A subdivision algorithm that adapts based on the gradient of the mesh would be more efficient

than subdividing every polygon. Note that if you reduce the triangle count you will also be much better equipped to get very fast subdivision.

Hint: Be careful not to introduce cracks in the mesh. Specifically, there should not be more than one level of subdivision between adjacent triangles.

- An Interpolating subdivision scheme, such as Butterfly subdivision.

10 Advice

- As always, start early.
- Make sure you understand the loop subdivision algorithm completely before you try to implement it.
- When debugging possible mistakes with interior vertices, try `cube.obj` as it is the simplest obj we have provided you.
- When debugging possible mistakes with boundary vertices, try `open_cube.obj`
- Debugging shaders can be difficult and frustrating, as you can't do any of the normal debugging functions (print statements, break point, etc.). However, you do have colors. If you need to see what sort of data is coming through your shader, try rendering colors that represent your data. For example, if your normals are acting weirdly, set the color of the fragment to the normal to visualize the normals across the mesh.
- Be careful with memory allocation, as too many or too frequent heap allocations will severely degrade performance.
- Make sure you have a submission directory that you can write to as soon as possible. Notify course staff if this is not the case.
- While C has many pitfalls, C++ introduces even more wonderful ways to shoot yourself in the foot. It is generally wise to stay away from as many features as possible, and make sure you fully understand the features you do use.

10.1 Additional Resources

http://www.glprogramming.com/red/chapter09.html
http://www.mrl.nyu.edu/~dzorin/sig00course/