15-462 Project 2: Geometry

Release Date: Thursday, February 04, 2010

Due Date: Thursday, February 18, 2010, 23:59:59

Starter Code: http://www.cs.cmu.edu/afs/cs/academic/class/15462-s10/www/proj/p2.tar.gz

1 Overview

In this project, you will have the opportunity to learn about and implement texture mapping and to learn and implement a subdivision algorithm. Specifically, you will use the loop subdivison algorithm. You will be adding code to texture map meshes and will be adding a method which will subdivide the mesh and modify the mesh you are rending accordingly. This document, the subdivision lecture and chapter 9 of the Red Book will be extremely helpful as they should cover everything 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 texture map on 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 based 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 class bboard, which can be found at `cmu.cs.class.cs462`. For this particular assignment, the subdivision lecture and this document will be the most helpful resources.

3 Submission Process and Handin Instructions

Failure to follow submission instructions will negatively impact your grade.

1. Your handin directory may be found at `/afs/cs.cmu.edu/academic/class/15462-s10-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.
   - Makefile and all `*.mk` files
   - `writeup.txt`
   - Any models/textures/shaders needed to run your code.

3. Please **do not** include:

   - The `bin/` folder or any `.o` or `.d` files.
   - Executable files
   - Any other binary or intermediate files generated in the build process.

   Run `make clean` before submitting. If you were using Visual Studio, be sure to clean the solution before submitting.

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

5. We will enter your handin directory, and run `make clean && make`, and it should build correctly. **The code must compile and run on the GHC 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 seperate 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.

4 Required Tasks

See the end of the document for some sample outputs.

Input: The input is a virtual camera, a loaded mesh, and a texture filename. Sometimes the texture filename is null, in which case no texture should be rendered.

Output: The output should be a rendering of the mesh using the given camera and texture. With each press of the 'r' key, the mesh should be subdivided using the loop subdivision algorithm.

For your program, you must:

- Correctly set the projection and modelview matrices based on the camera's values. The user should be able to move the camera around with the mouse and keyboard.
- Correctly render the mesh (very similar to the previous project).
- Use OpenGL fixed-functionality to add lights to the scene and correctly shade the scene with materials.
- Render the mesh with textures when a texture filename is provided.
- Subdivide each face of the mesh using to the loop subdivision algorithm. This includes calculating the new positions, normals and texture coordinates.
- 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 `project.cpp` and `project.hpp` in `geoemtry/` and `writeup.txt`, 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.

4.1 Starter Code

It is recommended that you begin by first reviewing the starter code is provided. Though the amount you must edit is small, we are providing you with a large code base to get you started and handle more mundane application tasks. Most of this is identical to the previous lab, only with bug fixes and necessary changes for the lab. The `README` gives a breakdown of each source file.

4.2 Building and Running the Code

The code is designed to run and build on the SCS Linux machines and comes with a makefile. Consult the `README` for more detailed build instructions.

We have also provided a Visual Studio 2008 solution, though it will take a bit of effort to get working since the programs have required command-line arguments. More details are in the `README`. If you use Windows, your project still **must** build and run on GHC Linux machines, so you will still have to test it on them before submitting. There are some differences in the compilers, so **code that compiles and works with Visual Studio may not compile or run correctly with GCC**. Make sure you test it well before the deadline. Be sure not to submit Windows binaries, either.

The program takes 1 required argument and 1 optional argument. The first required one is the filename of the mesh that will be subdivided. The second optional argument is the filename of a texture to use. The starter code will load the mesh for you, then give you the loaded mesh and the texture filename. If the mesh does not contain texture coordinates or no texture was given, then no texture filename.

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

4.3 What You Need to Implement

`geometry/project.cpp` and `geometry/project.hpp` are the two primary files that should be modified for your implementation, but you may add/modify other files if necessary. The specification for each function you need to write is given in `geometry/project.cpp`. These closely resemble the first project, though there is a new function you must implement, `subdivide`, which is invoked when the 'r' key is pressed. We handle they key press for you; you merely have to implement `subdivide` to perform one iteration of the loop subdivision algorithm.

We give you and handful of models in the `models` folder and some textures in the `textures` folder. Only some models have texture coordinates and therefore not all models can use textures. A text file in the `models` folder details which models have texture coordinates. `initialize` function, and should use some arbitrary material (like project 1) when there is no texture.

5 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.

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. 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.

6 Implementation Details

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

6.1 Mesh and Scene Details

The initialization function provides all data you will need about the scene:
`bool initialize( const Camera* camera, const MeshData* mesh, const char* texture_filename )`.

`camera` is the camera for the scene. It behaves exactly like the previous project.

`mesh` contains all the attribute data for the mesh. Meshes are in a similar format to the previous lab, with an array of vertices and an array of triangles with indices. The structs are defined in `geometry/project.hpp`. For this assignment, normals will be given to you for each vertex, along with position data. You **should not** compute the normals yourself. Some of the meshes also have texture coordinate data for each vertex. In those cases, you should render the mesh with a texture. Not all meshes we provide have texture-coordinate data, in which case the mesh we give you will have the zero vector for every texture coordinate.

`texture_filename` is the filename of the texture to use. This will be null if no texture was specified or if the mesh has no valid texture coordinates. Only use texturing if the texture filename is not null.

Note: Conveniently, you can still pass in texture coordinate data even if texturing isn't enabled, in which case OpenGL will simply ignore it. So you can

use the same rendering code for both textured and untextured meshes.

6.1.1 Transformations and Shading

We still provide with a camera that you must obey, and it should still move around with the keys and mouse similarly to the previous project. You must use texturing with the given texture filename when the filename is not null. Place at least one light in the scene, and use the same Blinn-Phong lighting used in the previous lab.

Everything else with transformations and lighting is up to you, though here are a few suggested guidelines. Note these aren't requirements, just suggestions.

- Leave the object without additional transformations besides the camera.
- Set the light color to whatever you like.
- When texturing is not used, set the material properties to whatever you like.
- When texturing is being used, leave the ambient and diffuse material completely white.

6.2 Texture Mapping

Texture mapping essentially allows you to glue a 2D image onto some set of polygons that are being rendered. The texture (specified as a 2D array of 4-byte pixels) is assigned a coordinate system in two dimensions, from $(0, 0)$ on the bottom left to $(1, 1)$ on the top right. Similar to specifying normals or other attributes, you will specify texture coordinates for every vertex of the mesh and OpenGL will then sample the texture based on these coordinates. Carefully look through Chapter 9 in the Red Book so that you understand texture mapping.

OpenGL supports 2, 3, or 4 dimensional texture coordinates for various uses. In our case, we'll only need 2-dimensional textures, so you only need read sections pertaining to 2D textures.

6.2.1 Loading in the PNG for texturing

We provide you a routine to load PNG images into a suitable format. Within `appliction/imageio.hpp`, there is a function named `imageio_load_image`. It returns array of bytes in RGBA format, along with width and height information. That is, the first 4 values are the RGBA values of the first pixel, the next four values are the RGBA values of the second pixel and so on.

Note: `imageio_load_image` uses `malloc` to allocate space for the data, so be sure to free it. Once the texture is loaded into OpenGL, it is safe to free the loaded image data.

6.2.2 Creating an OpenGL Texture

Consult the Red Book for more details on this section. This is just an overview.

OpenGL manages textures (as well as several other kinds of objects) using a handle-based system. You can create a "texture object" using `glGenTextures` which will give you an integer handle for that texture.

Like everything else in OpenGL, the current texture is part of the state, so you must bind your texture handle using `glBindTexture`. Then, all texture functions affect the current texture. You can change various setting for the current texture and, most importantly, load in texture data using `glTexImage2d`. One of the arguments for this texture is the input format. For images loaded using `imageio`, use the format `GL_RGBA`. You must also specify a filtering mode using `glTexParameteri`. Consult the Red Book for details.

6.2.3 Specifying Texture Coordinates

Texture coordinates are specified in the same manner as other vertex data. For example, you can use `glTexCoord` in begin/end pairs or `glTexCoordPointer` with vertex arrays. The interface is very similar to that for normals.

6.3 Loop Subdivision Algorithm

6.3.1 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 an 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. *Interior edges* are the complementary set: any edge shared by at 2 triangles. An interior vertex lies on only interior edges.

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 are 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 is does for all meshes we provide.

6.3.2 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.

6.3.3 Even Vertices

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

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

6.3.4 Normals and Texture Coordinates

For the purpose of this assignment, you can use the same algorithm to calculate the new normals and the new texture coordinates 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.

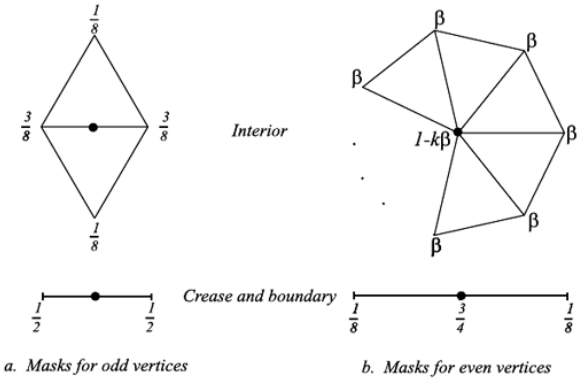

Figure 1: Illustration of the algorithm

7 Advice

7.1 Suggested Sequence

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

1. Familiarize yourself with the changes to the given mesh structure.
2. Add in code for camera transformations, mesh rendering, and some lighting (you can probably just recycle this from the last assignment).
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 / crease case for odd and even vertices. At this point, you should be able to subdivide all of the meshes (including the openg cube and the stegosaurus).
6. Add texture mapping. You should now be able to render `bunnytex.obj`.

7.2 Words of Advice

- As always, start early.
- Make sure you understand the loop subdivision algorithm completely before you try to implement it.
- Remember (from 15213) that because of the way floats are stored, comparing floats directly could be a bad idea. To check if two floats (or vectors of floats) are equal, you may need a routine that checks if their distance is wihitn some tolerance of one another.

- 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`
- 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.

7.3 Additional Resources

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

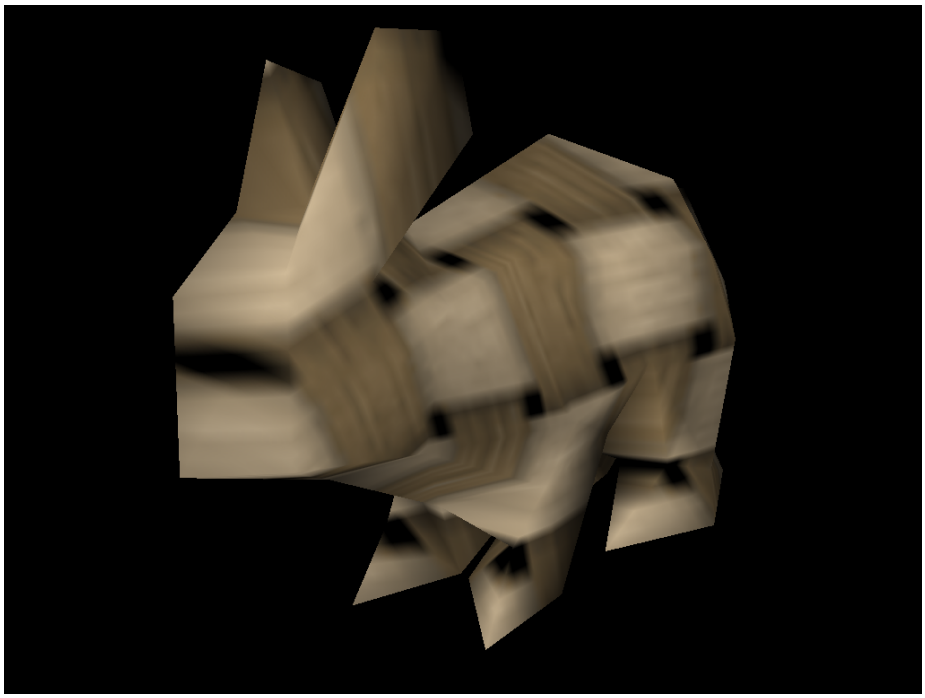

Figure 2: Textured bunny with 0 subdivisions

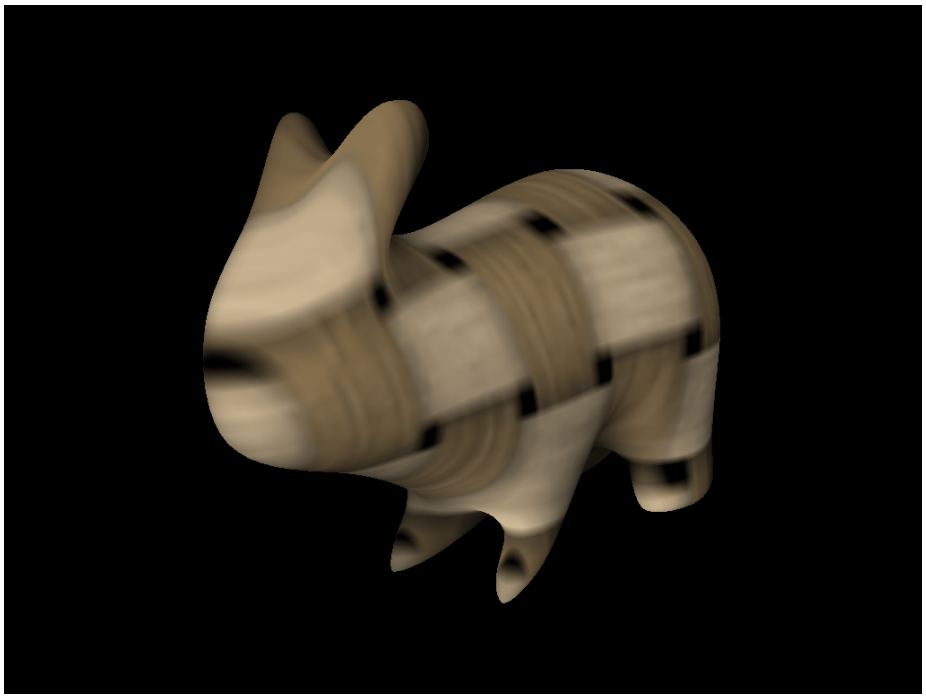

Figure 3: Textured bunny with 3 subdivisions

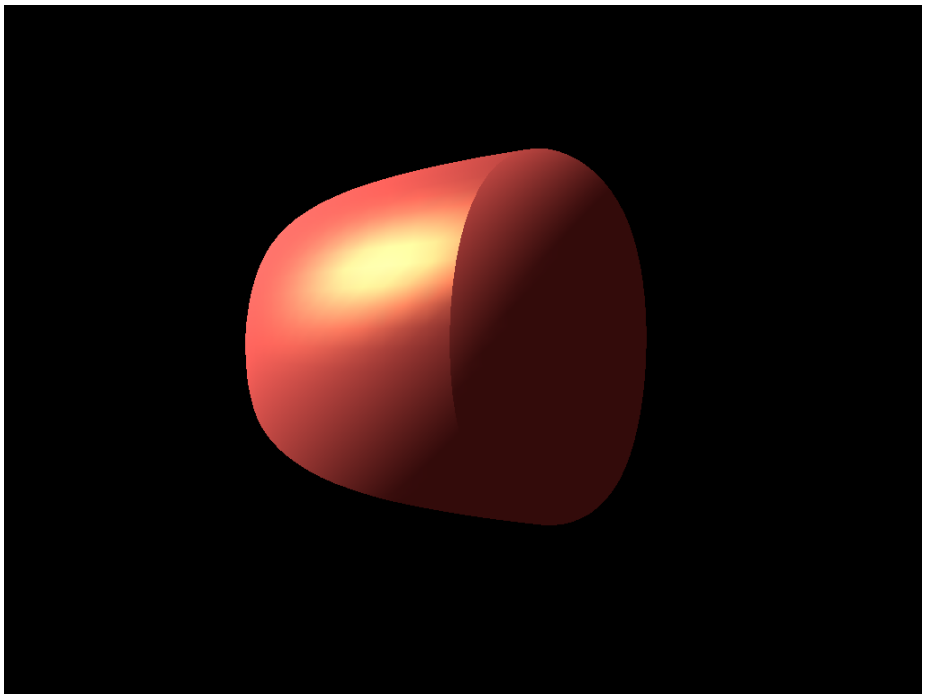

Figure 4: open_cube with 4 subdivisions

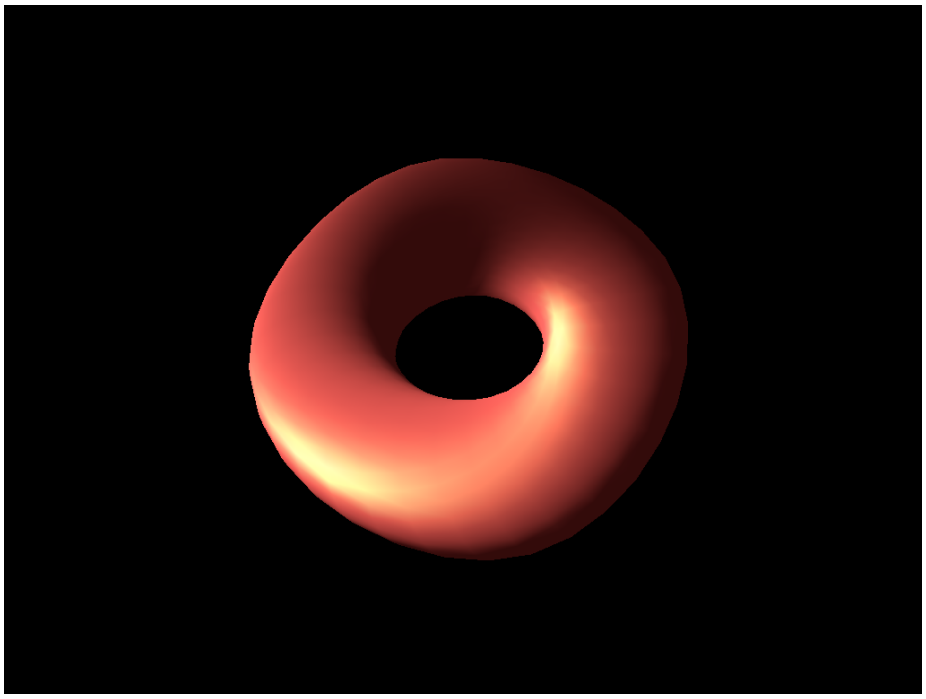

Figure 5: Torus with 3 subdivisions

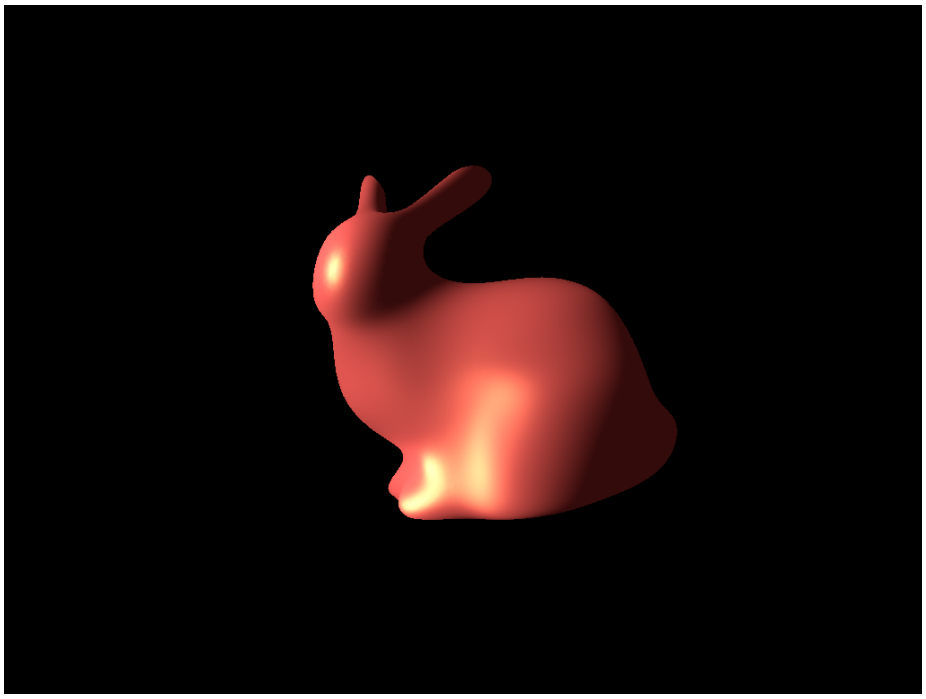

Figure 6: Bunny with 3 subdivisions

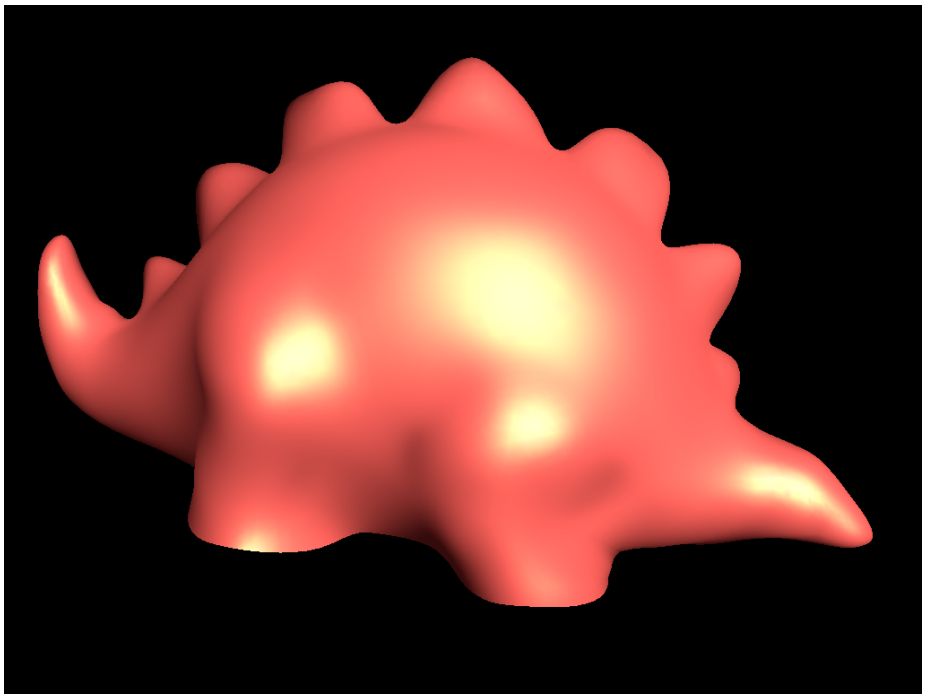

Figure 7: Stegosaurus with 2 subdivisions