CS 148 - Homework 5 - Meshes & Laplacian Smoothing

Overview

When you run the program, you'll see the infamous Stanford bunny hanging out atop his favorite ball. He's cute, but don't get too emotionally attached or you risk trauma when you see what this algorithm does to him. You can click the mouse and drag around to rotate the scene.

The starter code contains an OpenGL skeleton that should look familiar. We've also provided you with a simple graphics library, libst ("st" for "Stanford graphics library"). It contains lightweight data structures and routines for common graphics tasks. Take a look through the header files to see what the library affords you. An important class for this assignment is STShape, which we use to model our mesh. It includes our Vertex and Face classes and also allows us to import 3D models as meshes from disk as .obj files. This assignment uses the "bunny.obj" file we've provided. Feel free to play with other files you find online, but be aware that .obj files can vary considerably in format, and STShape's .obj loading routine works correctly on only a subset of them.

You will not be writing huge volumes of code for this assignment. Instead, much of your work will involve familiarizing yourself with the STShape files and classes used inside of them. Make sure you understand it inside and out before you begin coding. If you don't, you'll struggle to implement the correct additions to the starter code's data structure to help you with the smoothing routine.

Also, before you begin coding, make sure you understand the mathematics of Laplacian smoothing. Issues with mesh algorithms can be hard to debug analytically; it's hard to inspect a massive number of vertices and faces one-by-one. So, understanding what you're doing up front is crucial.

In Laplacian smoothing, recall that the main idea is to reposition vertices such that at each iteration, every new vertex position is defined as:

$$x_i = \frac{1}{N} \sum_{j=1}^{N} x_j, \text{ where each } x_j \text{ is one of the N neighboring vertices of } x_i \text{ prior to the smoothing iteration.}$$

Crucially, note that the computation of each vertex at each iteration must happen “in parallel”; that is, it is incorrect Laplacian smoothing to alter vertex (i+1) of a mesh whose vertex i has already been altered within the same iteration. Think of the cleanest way to avoid this problem.

Also note that in the above formula, we are implying the “lambda” value from the algorithm provided in lecture equals 1. Note that setting lamba to 1 may not work in the most general case for arbitrary meshes, but it’s good enough for us in this case. See extra credit option #1 below for information about how you might go deeper here.

Suggested Steps

(1) First, you’ll need to understand how to use the mesh data structure for your smoothing routine. Begin by making sure you understand how STShape stores the information from the .obj file. Next, given STShape’s lists of vertices and faces, make sure you understand what’s happening when you look up which vertices neighbor any given vertex.

(2) Next, implement a single iteration of Laplacian smoothing. It may be difficult to verify that it’s working perfectly on complex, high-resolution shapes like the bunny. A recurring theme in computer graphics is imagining clever test scenes. So, if you like, think of a simpler situation you can use as a sanity check. Note that the STShape class provides helper methods for creating some simple polygons. Beyond these, see if you can imagine any other clever scenarios that prove your smoothing is flawless.

(3) Lastly, allow for an arbitrary number of smoothing iterations. The starter code provides you with a helper variable to toggle between iteration counts. Use the keyboard’s number keys (1-7) to alter the iteration count and re-render the scene. The number of current iterations is printed out and is equal to 2^(“number key” - 1). Each time you press a number key, your program may begin iterating from the original, unaltered mesh. In other words, if you press the “2” key to perform 2 iterations, and then press the “3” key to perform 4 iterations, your program does not need to smooth the mesh by starting from step 2; it can simply start from the original mesh each time you select a new count.

Deliverables and Evaluation

1) We'll compare your Stanford bunny meshes with the correct ones for each number key 1 through 7. Please submit front-facing images of your bunny for each of these iteration counts.

2) We'll also evaluate your code, It shouldn't take a long time to perform the smoothing, so make sure not to do anything needlessly slow. For reference, our solution takes only a split second to perform the maximum iteration count of 64 without doing any clever optimization beyond using correct data structures.

Extra credit

Notice that (especially at higher iterations) Laplacian smoothing deforms both the shape and size of your mesh. The overall size may shrink quite a bit, and finer features may disappear altogether (those poor ears).

Implementing the below routines is optional, but we'll reward you if you complete either of them (or both, you overachiever, you). Note that the second is much more involved than the first. If you complete either, please make a note in your submission's README file along with images and/or other instructions for us to examine your glorious work.

(1) Even simple modifications of Laplacian smoothing can help preserve the mesh's original features. There are many ways to do this, and each has different use cases.

A well-known extension of laplacian smoothing is the **Taubin Smoothing** routine we mentioned in lecture. You can find the original research paper here:

https://graphics.stanford.edu/courses/cs468-01-fall/Papers/taubin-smoothing.pdf

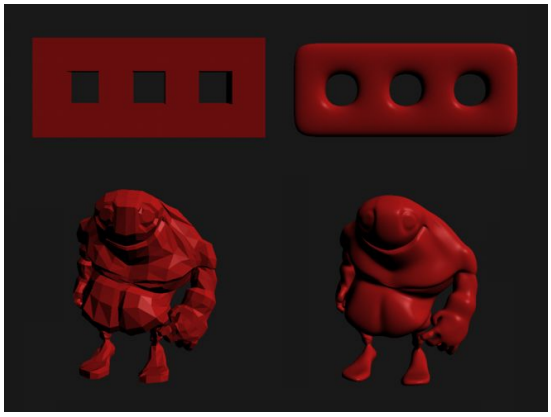
Note that this algorithm implies you will need to alter your Laplacian formula to include varying values of lambda as described in lecture.

You can consider implementing the **HC-algorithm** described in section 4 of this paper, which aims to help reduce artifacts of noisy meshes, like those originating from digital scans.

http://www.joerg-vollmer.de/downloads/Improved_Laplacian_Smoothing_of_Noisy_Surface_Meshes.pdf

You'd likely want to modify your solution to the original assignment and simply include a keyboard toggle which shifts between the default Laplacian smoothing routine and any optional smoothing algorithm you implement here.

(2) A more complex method for smoothing involves re-thinking the problem altogether. Rather than simply re-locate existing vertices, we can imagine adding new vertices at each step of smoothing. Doing so increases the overall resolution of the object without distorting its overall size, polishes existing details, and can even introduce new details. Consider implementing one of the most classic and beautiful algorithms in all of computer graphics, Catmull-Clark subdivision.



See here for a description:
https://graphics.stanford.edu/wikis/cs148-09-summer/Assignment3Description

To keep things clean for this one, please create an entirely separate program from the original assignment, and submit us images of any models you choose to smooth at various counts of catmull-clark subdivision. A good test case for this is an icosahedron turning into a sphere like below.