CSCI 4611: Programming Interactive Graphics and Games

Assignment 3: Earthquake Visualization

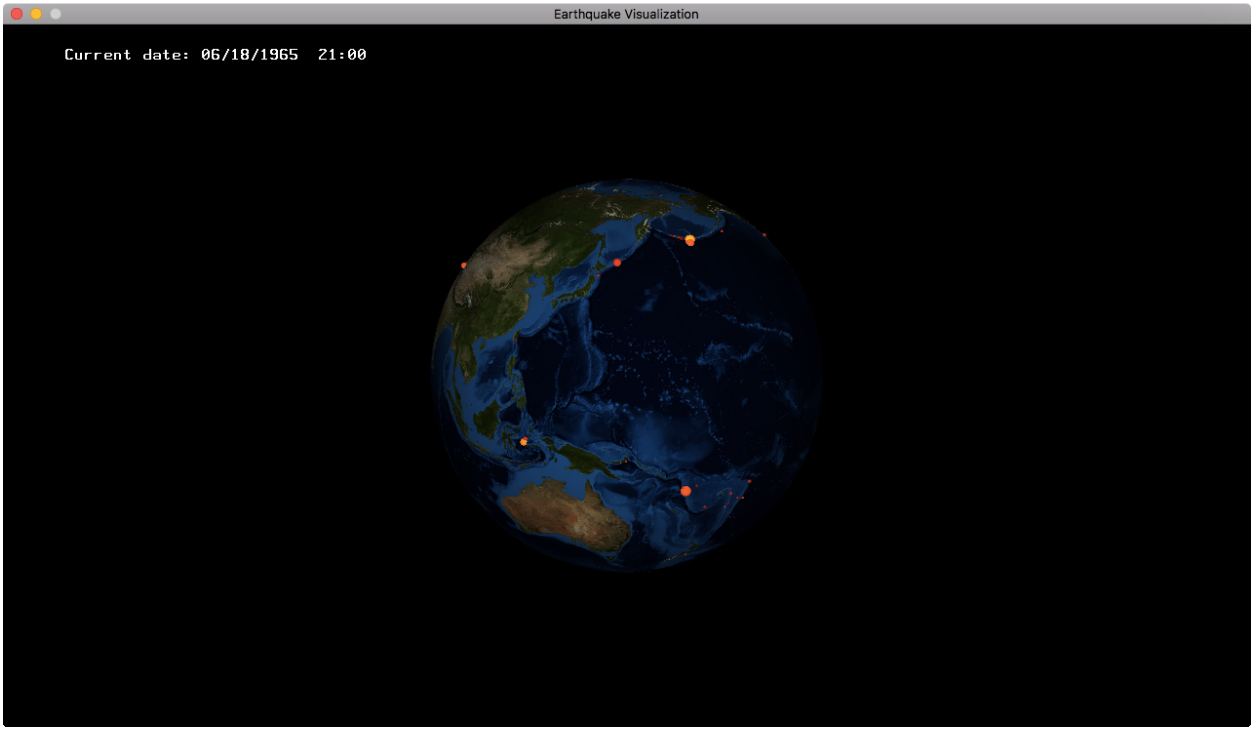
Handed out: Monday, February 27

Due: Monday, March 20

Introduction

For this assignment, you'll be working with data from NASA and the USGS to visualize on a globe the locations where earthquakes happened between 1905 and 2007. Visualizations incorporating geospatial data are used and analyzed in many different contexts, including navigating a city (as seen in GPS devices), commercial development, and setting governmental policy. This area also receives a significant amount of research attention. For example, Prof. Vipin Kumar and others at the University of Minnesota are working on visualizing and understanding global warming datasets as part of an NSF project (you can find more information at http://climatechange.cs.umn.edu/).

Your assignment should look like the image below when you've finished it.



The earthquake dataset you'll be using includes 13,540 different earthquakes. The Earth texture dataset from NASA is available in resolutions down to 500m/pixel (although getting this to display on your graphics card is well beyond the scope of this assignment).

In this assignment, you will learn to:

- Visualize real-world geographical data on a 3D textured globe
- Apply textures to 3D objects
- Algorithmically create a deforming 3D mesh and display it using vertex buffers
- Define normal vectors and texture coordinates for a sphere
- Convert from spherical coordinates (latitude and longitude) to 3D Cartesian coordinates.

Earth and Earthquake Data

We have included multiple scaled-down versions of this Earth texture with the code distributed on the website, since you need a fairly powerful computer to render even the lowest-quality image from the NASA page. In order of decreasing quality, the following images are provided:

- `earth.jpg`: Full-resolution (8 km/px, 5400×2700) image from NASA page
- `earth-2k.bmp`: 2048×1024 scaled-down version of image
- `earth-1k.bmp`: 1024×512 version of image
- `earth-512.bmp`: 512×256 version of image
- `earth-256s.bmp`: 256×256 version of image

Almost any graphics card should be able to use the 256s version of the Earth texture. If your computer isn't able to use this texture, you will need to use the computers available in the various CSE Labs. The Earth textures are stored in a equirectangular projection, which simply means that the *x* coordinate corresponds directly to longitude and *y* directly to latitude.

The earthquake dataset contains information about the earthquake's magnitude (a measure of how severe the earthquake is) and its longitude and latitude. You'll be required to display this information on the globe in a meaningful manner. More information on the earthquakes is available in the data file, and if you are interested, you can try to figure out ways to integrate additional data variables into your visualization.

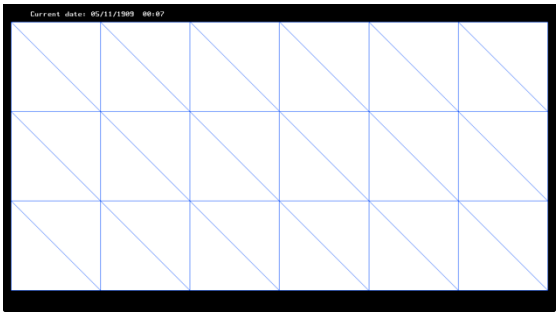
Requirements and Grading Rubric

You will be required to write the code that displays the Earth on screen and animates it between flat rectangular map and a three-dimensional globe. You will also need to write the code to display the earthquakes on the Earth. The starter code defines a single light source that coincides with the camera, so that surfaces seen head-on (i.e. whose normal points

towards the viewer) will be bright while those that are slanted will be darker. Pressing Space will pause the current visualization time, while the left and right arrow keys will make it slow down and speed up respectively. Once you have the mesh being drawn, pressing M will visualize its triangles.

Work in the "C" range will:

Use vertex buffers and element buffers to draw a rectangle subdivided into multiple triangles, representing a flat map of the Earth. You can use the M key to view the mesh structure.



- The rectangle should lie in the *xy* plane, with x going from $-\pi$ to π and *y* going from $-\pi/2$ to $\pi/2$. It must be divided into *slices* divisions horizontally and *stacks* divisions vertically. Therefore, the mesh will have (*slices*+1)×(*stacks*+1) vertices and 2×*slices*×*stacks* triangles. On the right is an example with 6 slices and 3 stacks.
- Fill in the necessary parts of the `Earth::getPosition()` and `Earth::getNormal()` functions and use them to populate the vertex and normal arrays. You should assume that the input latitude lies between −90° to 90°, and longitude lies between −180° to 180°.
- Populate the element buffer with the appropriate vertex indices. It may help to work out the indices by hand for a small example, say *slices* = 4, *stacks* = 2, and generalize from there.

Work in the "B" range will:

Apply a texture of the Earth to the rectangle so that it looks like the original image.

- Remember to call `unsetTexture()` at the end of the Earth drawing function, so that the texture does not get applied to things drawn afterward.

Display on the Earth all the earthquakes that have happened within the past one year of the current visualization time.

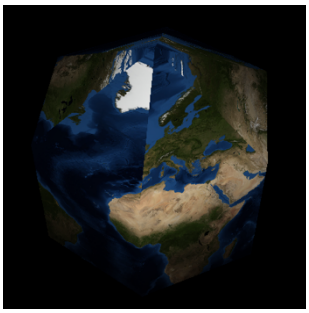
- Use `earth.getPosition()` to obtain the earthquake positions such that they match those of the sphere textures (i.e., an earthquake occurring in California must be displayed in the same location as the California of the Earth texture). One way to tell if your mapping is correct is if you see a lot of earthquakes along the "Ring of Fire" in the Pacific Ocean off the coast of Asia.
- You may use the `Draw::` functions to draw the earthquake markers or come up with your own custom geometry. Earthquake sizes and colors should be based on the

earthquake data in some meaningful way. Explain and justify the mapping you choose in your `README`.

Work in the "A" range will:

Change the vertex positions and normals to draw the Earth as a sphere instead of a rectangle.

- In the `QuakeVis` constructor, set `isSpherical=1`. Fill in the necessary parts of `Earth::getPosition()` and `Earth::getNormal()`.
- You will need to convert latitude and longitude into the three-dimensional Cartesian coordinates of the corresponding point on the sphere, using the formulas
  $x = \cos(lat)\sin(lon)$,
  $y = \sin(lat)$,
  $z = \cos(lat)\cos(lon)$.
  Be careful that the input latitude and longitude are in degrees, not radians.
- The texture is permitted to look slightly "cut off" *only* at the top and bottom stack of the Earth mesh. The image to the right shows what this looks like when *slices* = 6 and *stacks* = 3. (See below for why this happens.) Increasing the number of stacks and slices will make this problem harder to see.
- The earthquakes must appear on the sphere in the correct geographical locations (though they may not lie exactly on the mesh if it has too few slices and stacks).



Smoothly transform the mesh between a rectangle and a sphere based on user input. The program should start with the Earth displayed as a rectangle. When the user presses the S key, it should continuously deform to a sphere. If S is pressed again, it should deform back to a rectangle, and so on.

- To start with, you may call `earth.setSpherical()` whenever the S key is pressed to update the state of the mesh.
- Immediately toggling the state creates an abrupt transition. Instead, it is better to smoothly transition between the rectangular and spherical shapes, by gradually changing the *spherical* parameter from 0 to 1 or vice versa over multiple frames. One way to do this is to add a variable to `QuakeVis` to keep track of the target value of *spherical*. Only change the target value on user input, and adjust the state of the mesh towards the target at each frame.
- Fill in the remainder of `Earth::getPosition()` and `Earth::getNormal()` to smoothly interpolate between the rectangular and spherical values. (Technically, the correct normal of the interpolated shape is not simply the interpolation of the normals, but we may pretend it is for this assignment.)

If you reduce the value of *slices* and *stacks* and then watch the mesh structure as it interpolates between a rectangle and a sphere, you should be able to see why the texture appears to be cut off near the poles.

Useful Math

Here are a few mathematical operations that are very common in graphics and may be useful for this assignment:

- Linear interpolation: One way to blend smoothly between two values x and y (which could be reals, or vectors, or matrices, etc.) is to define a function whose output varies continuously from x to y as a scalar parameter a goes from 0 to 1. This function is traditionally abbreviated "lerp":
  `lerp`$(x, y, a) = x + a(y - x)$.
  Thus, for example, `lerp`$(x, y, 0) = x$, `lerp`$(x, y, 1) = y$, and `lerp`$(x, y, ½) = (x + y)/2$.
- Clamping: A concise way to constrain a value to lie in a specified interval $[a, b]$ is to define a "clamp" function `clamp`(x, a, b) which returns a if $x \le a$, returns b if $x \ge b$, and returns x otherwise.
- Rescaling: Suppose you have a value x in the range $[xmin, xmax]$, and you want to find the corresponding value in $[ymin, ymax]$. Observe that $x - xmin$ lies in $[0, xmax - xmin]$, and $(x - xmin)/(xmax - xmin)$ lies in $[0, 1]$, so the desired value is
  $y = ymin + (ymax - ymin)(x - xmin)/(xmax - xmin)$.

Above and Beyond

All the assignments in the course will include great opportunities for students to go beyond the requirements of the assignment and do cool extra work. We don't offer any extra credit for this work — if you're going beyond the assignment, then chances are you are already kicking butt in the class. However, we do offer a chance to show off… While grading the assignments the TAs will identify the best 4 or 5 examples of people doing cool stuff with computer graphics. After each assignment, the selected students will get a chance to demonstrate their programs to the class!

There are great opportunities for extra work in this assignment. For example, the source website for the Earth texture (see Data Credits below) has images for each month of the year. You could animate between the textures based upon the current time of year. We have also included some `height` images that contains the elevation and bathymetry data for the Earth, with sea level being 50% gray, and black and white indicating 8 km below and above sea level respectively. You could use this image to visualize the shape of the Earth's surface by loading it with `SDL_LoadBMP()` and using its pixel data to displace mesh vertices along their normals. Be creative!

Support Code

The webpage where you downloaded this assignment description also has a download link for support code to help you get started. The support code for this assignment is a simple program using the SDL-based engine, similar to the ones we have used before. You should also download separately the zip file containing the texture and earthquake data files.

The support code defines a program structure and everything you need to read and parse the earthquake data file. **To make locating data files simpler, we have provided a header file called `config.hpp` that contains absolute paths to your data files. You should edit this file with the full path to your data files (e.g. `"C:\Users\Turing\a3\data"` or `"/home/turing/a3/data"`).** We will modify this file appropriately when grading your assignment.

Handing It In

When you submit your assignment, you should include a `README` file. This file should contain, at a minimum, your name and descriptions of design decisions you made while working on this project. If you attempted any "above and beyond" work, you should note that in this file and explain what you attempted.

When you have all your materials together, zip up the source files and the `README`, and upload the zip file to the assignment hand-in link on our Moodle site. You don't need to include the data files, which are pretty large and which we have a copy of anyway. Any late work should be handed in the same way, and points will be docked as described in our syllabus.

Data Credits

The earthquake data comes from http://earthquake.usgs.gov/data/centennial/. As per http://www.usgs.gov/laws/info_policies.html, this data is in the public domain.
Credit: U.S. Geological Survey,
Department of the Interior/USGS

The Earth texture comes from http://visibleearth.nasa.gov/view_cat.php?categoryID=1484. As per http://visibleearth.nasa.gov/useterms.php, this data is freely available for re-use.
Credit: NASA Earth Observatory