Lab 6: The World Ain't Flat! [Textures]

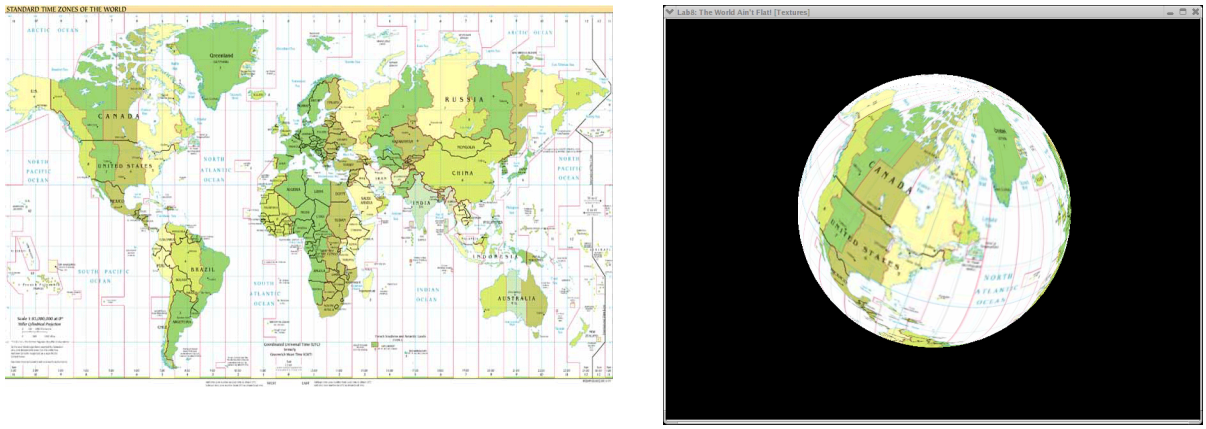

Fig 1: Screenshot showing the Mercator map used as texture and the result of the mapping. Note the relative sizes of USA and Greenland in both. This has been one of the strongest points against the Mercator projection.

You must have heard of the book by Thomas L. Friedman titled "The World is Flat". This is a controversial book on globalizing forces like off-shoring, in-shoring, out-sourcing, open-sourcing etc. and on how they have revolutionized the concept of business and economy. This lab will give us a chance to prove Thomas Friedman wrong! A Mercator[1] (flattened) world map obtained from Google search will be used as a texture and our task will be to map it onto a sphere thereby un-flattening the world!

Steps in applying any texture:

1. Generate/obtain and pre-condition the texture image.
2. Load the texture image from file and import it into memory.
3. Specify the image as texture using glTexImage2D()
   Specify texture-mapping options:
   1. glTexParameter()
   2. glTexEnvf() [decal / modulate / blend / replace]
4. While constructing the 3D/2D object, specify what texture coordinate each vertex should have. For this lab we will construct a sphere from first principles like we did in the lab on lighting.

After completing the world-map assignment, you can take a shot at implementing something similar to Google's street-view (see Sec. 5.2 below).

1 Obtaining and Preconditioning the Texture Image

A political world map in Mercator projection has been downloaded for you and is available as world-map-country-names.tga. To specify any image as a texture one must call glTexImage*D() (*=1,2,3) and since we will work with 2D textures we will use glTexImage2D(). A quick examination of its prototype reveals that the last parameter to be provided must be the address of an array of pixels. Exactly what this array contains and how the information is organized can be

1 http://en.wikipedia.org/wiki/Mercator_projection

specified using the other parameters for this function, but first we must ensure that we have such an array. For simplicity, we will process the TGA file to give us this array.

1.1 The TGA file format and GIMP

The Truevision TGA raster graphics file format is a simple way of encoding color information in images. The RGBA components of color are each stored using 1-32 bits of *depth,* and the pixels are stored contiguously. An optional RLE (run-length encoding) compression can be employed but this would work against our purpose of creating an array of uncompressed pixel data.

If you want to use any other image you must first convert it to the TGA format (for this lab) which can be achieved in two ways:

1. Using ImageMagick (Linux/Mac)
2. Using the GIMP or Photoshop (Linux/Mac/Windows)

1.2 ImageMagick

ImageMagick[1] is a suite of free programs for creation, modification and display of images in a very wide range of formats. Run the following on the command line to see what you can do:

```
csh-prompt% man ImageMagick
```

We are interested in the convert utility. To convert between image formats, run the following on the command line:

```
csh-prompt% convert <filename1>.<ext1> <filename2>.<ext2>
```

For example,

```
csh-prompt% convert world-political.gif world-political.tga
```

will convert the GIF image here to the TGA file format, also renaming the file. You can provide many options to process the image during conversion, like resizing, rotating, adding effects etc.

2 Loading the Image

The LTGA class (in ltga.h and ltga.cpp) has been provided so that you can easily work with TGA images. You will encounter this class again in PA3 on ray-tracing. Create an instance of the LTGA class and provide the path to the TGA image file in the constructor.

The main file you need to work on for this lab is globe.cpp which, upon compilation, will result in an executable named `globe`. This program will require the path to the TGA image file as a command line parameter. So include this information while instantiating the LTGA class. This will obviate the need to recompile each time a texture is to be changed.

Study the LTGA class members to see what information you can gain. Based on what these return, you will need to adjust parameters to the glTexImage2D() call in the init() function.

A simple program named tgainfo has been provided as an example of LTGA class usage. It creates an LTGA instance from the file provided and prints out information using the LTGA public members. The source code is in tgainfo.cpp. You can run this as

```
csh-prompt% tgainfo  <imagefile>.tga
```

1 http://www.imagemagick.org/

3 Specifying the Image as a Texture

The texture must be initialized before any OpenGL drawing calls using it. Make a call to glTexImage2D() in the function init() and use the info returned by the LTGA class to enter the parameters. Consult the glTexImage2D man page for documentation on its parameters.

3.1 Specifying Texture Parameters

Texture images can be stretched to fit or be tiled. This is specified using calls to glTexParameterf(). The following parameters are of use:

1. GL_TEXTURE_MIN_FILTER
2. GL_TEXTURE_MAG_FILTER
3. GL_TEXTURE_WRAP_S
4. GL_TEXTURE_WRAP_T

Also, once the texture is stretched or tiled, the texels can be copied to replace pixels on the surface, or can be blended or be used to modulate the surface so that lighting and textures are combined. Selection of this mode is achieved using the glTexEnvf() function with the GL_TEX_ENV_MODE parameter. Consult the man page or online documentation for help.

4 3D Object Construction and Texture Coordinate Calculation

Now we are ready to map various points on the texture image to vertices on the 3D object at the time of construction. Construct a sphere of radius 1.0, centered at the origin, and map the Mercator world map onto the spherical surface. Texel coordinates, *s* and *t*, lie in [0,1] in each dimension so decompose the spherical surface into quadrilaterals (or triangles) and calculate what portions of the globe image should map to each quadrilateral.

The use of display lists is a requirement. They can optimize texture rendering based on hardware capability because the list is compiled beforehand.

5 More Fun with Textures

5.1 Antarctica on the world map

A couple of things are just wrong with the map given. For example, where (on earth?) is Antarctica? Another Mercator map with Antarctica is provided and is titled Mercator-proj.jpg. View this image first using gimp, xv, or eog. Try applying this map to the world and see how Antarctica scales.

A texture for an orange is provided in orange-texture.jpg. Apply this to the sphere and see what you get. Try tiling the texture instead of stretching it. How would you achieve this?

5.2 Google's street-view

Create a cubical box larger than and encompassing the sphere and paste the images titled `cm_*.jpg` in the `images/` folder onto the walls. Implement rotation in your keyboard callback so that you get the illusion of standing in the environment approximated by the 6 images. This is a

basic step in how Google probably implements street-view – a car with cameras mounted on the top drives through the streets in an area photographing the front, back, left, right, top and bottom views at regular intervals. At any point on the street-view, it creates this simple viewing system letting you to rotate the view.

6 Grading

For the grading of this lab the final product should be a textured Earth in side the cube described in 5.2. The other tasks are for thought and for fun.

7 Postscript

The exercise here has been simple: we have preconditioned an image and mapped that onto a 3D surface. We have merely scratched the tip of the iceberg. The scope of all that can be done with textures is a book unto itself. What happens when the texture image and the primitive are of vastly different sizes? Can texture elements (texels) simply be copied? What would you do if the multiple texels map to one pixel? What would you do if a single texel maps to multiple pixels? If the object being viewed can be scaled widely, then textures would have to be scaled with the object. How would you prevent aliasing? Think *mipmaps*.

The OpenGL Red Book chapter on Textures is a very good source of information. Also, the man pages for all OpenGL functions extensively document the various parameters you can provide, and their effects.