CPSC 641 – Computer Graphics
Assignment 1 due 9/28/17 (class time)

The goal of this assignment is to give you practice with the concepts of sampling, antialiasing, and texture mapping. The general idea is to demonstrate anisotropic antialiasing of texture maps, and to look at multiple sampling techniques when doing so. There are multiple parts to the assignment, which will be described below.

Note that you may choose to incorporate all parts of this assignment into a single program, or to write multiple programs to handle the individual pieces.

1. You should plan to provide at least 3 test images in some format for demonstrating your system:
    a. A high-contrast checkerboard pattern of two colors. It should have a sufficient number of squares to provide good
    b. An image (can be a photo or CG image, or whatever) with some sharp contrast edges within it. The edges should not be strictly horizontal/vertical.
    c. A photograph of some sort (image of the real world)

   Note that you may work with others to get good images – this is not directly part of your grade, and I just want to make sure you have good images to test later parts with.
2. You should write code that reads in an image that will be used for a texture and generates at least 5 different mipmaps for that texture, each one produced by using a different filtering technique. You should write the code to perform the filtering yourself. The three filters should include:
    a. A box filter using a square footprint
    b. A box filter using a circular footprint
    c. A triangular filter using a square footprint
    d. A triangular filter using a circular footprint
    e. A filter of your own choosing (I might suggest something Gaussian-based)

   Please note the following:
    - You are welcome to use outside libraries to load in the data, store it in an array or some internal data structure, and write out the data. The image format is up to you, but it should be compatible with the 3 images you provide in part 1.
    - I am expecting you to determine the filter area and then to determine the relative weighting of sample points from the image for that filter, in order to generate pixels in all the various mipmap levels. Do not just use a library to do this.
3. There should be a way to view the mipmap levels on their own. It can be a static image or an interactive program, but it should be possible to examine the entire mipmap, looking at every level.

4. Create a system that will display a polygon mapped with a given image (and its associated mipmaps). You should support anisotropic filtering on the displayed image, AND should support two displays that just uses a single texture sample.
    a. For the single texture sample (no mipmapping), just pick the pixel center and calculate its texture coordinates. You should support an option where you use that value to bilinearly interpolate the associated texels, and one where you just use the nearest texel.
    b. For the mipmapped levels, you should implement an anisotropic filter as described in class – one that maps the pixel corners to 4 texture values, uses these to determine the pixel projection in texture space, determines a line of anisotropy, a sample ratio to use, and a sample value to use, and then finally blends together multiple samples to get the anisotropically filtered approximation of the pixel coverage for display.
    c. You should allow users to use any of the 5 (or more) mipmaps you created. This could be by generating images with all mipmaps at once, letting them choose between them, letting them specify which filter they want to use at load time, or any other mechanism you wish. Coupled with the 2 single-sample options, there should be at least 7 different ways to display the texture
    d. Your image should show a range of levels of anisotropy and orientation of anisotropy.
        i. A relatively simple way to do this is to map the image to a large quad that is viewed nearly on edge. This will ensure that nearby pixels have small areas, while those farther away have more stretched areas (more anisotropy). To ensure that anisotropy is not biased to the texture directions only, it would help to have the ability to have the quad oriented at an angle to the viewer.
        ii. The details of how you do this are up to you. You can have an interactive system or a static one. You could have a fixed “flight path” for the textured polygon. You could stretch and squash the polygon itself to change the anisotropy, etc.
        iii. You do not need to include any lighting effects – the only consideration here is determining the texture value to display.
    e. I expect you to write the code to determine pixel corner coordinates, map them to the texture domain, determine the correct samples to use from the mipmap and blend them to determine the pixel color that should be displayed. Feel free to use any libraries to handle the display, any interaction, etc., but not to do the filtering calculations.
5. Turn in (in class, in hard copy form) a short writeup. Details are given below.

You should turn in your code to ecampus by the deadline. Please zip all source code and your images into one file. You may be expected to demo your code to the instructor in the following week.

Your writeup should be turned in in class and should be about 1 page in length (try to keep it to 1 page). The writeup should include the following:

1. What additional filter you tried to implement, and why.
2. A brief description of how your program(s) operate – how do you need to interact with them.
3. A list of any outside libraries used, and what they were used for.
4. If there are any non-standard techniques that you made use of that require further explanation.

Notes:
1. For loading the texture from a file, you probably want to use existing code (external libraries are OK) for loading image files. You can probably find code to load .bmp or .tga images on the web, or ask around to see if someone will let you borrow theirs. Since you will be providing the input files, you can use any file format and reading routine you want (as long as it compiles). You should note in your writeup and source code the original source for any such external libraries.

2. To calculate a depth value to use in the mip-map, you will need to determine the line of anisotropy. For the purposes of this assignment, you may determine this line by connecting the midpoints of opposite edges of the pixel's projection into texture space. The longer of these two lines will be the line of anisotropy, and the ratio between the two lines will be the anisotropy factor.

3. You may use a linear interpolation between levels, bilinear filtering within a level, and simple averaging of samples along the anisotropy line, regardless of the filter originally used. You can filter in each color domain (red, green, blue) independently.

4. I am expecting you to write your own code to do the following:
Filtering (i.e. do summing and averaging yourself, don't call another routine to do it for you).
Creating the mip-maps based on the filters
Using the mip-maps to texture the polygons, pixel by pixel (including calculating the depth value and using that depth value to index into the mipmap).
You are welcome to use other code to perform the other functions you need, such as file loading, rasterization (though this is probably simpler to do yourself since you have to back project the pixels into the texture domain), display of the final image, any interactive elements you want to include, etc. If you use any code that you did not write, state in your writeup what routines you used and from where you got them.

5. You should expect to have to demonstrate your code to the instructor. Appointments will be set up to allow this.