Lab 1 – CPE 471 – Starting software rasterizer with the computation and drawing of a bounding box

Due Thursday (1/9)

Please work individually but feel free to chat with your neighbors and discuss plans and ideas, etc.

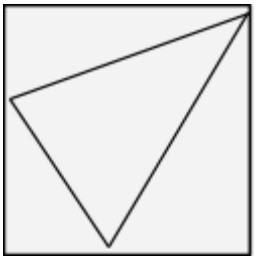
Objective: Introduction to rasterization (drawing via computer program)

Overview and context:
Over the next 2 weeks, we will be writing a program to render (draw) an indexed face set (aka polygonal mesh of triangles) as an image via software rasterization (i.e. the code will be entirely written in C/C++ with no graphic libraries. The software will render a static scene). In general the required steps for the program will be:

- Read in triangles
- Convert triangles to window coordinates
- Rasterize each triangle (using barycentric coordinates for linear interpolations and in-triangle test)
- Write color values per pixel (using a z-buffer test to resolve depth)

We will discuss the window coordinate transform, barycentric coordinates and z-buffers in the next few lectures.

Before we can draw triangles, we will want to compute a 2D bounding box for any triangle we might need to draw. *The bounding box of the triangle is the box with the extents that will exactly bound that triangle*. (For the method of rasterization we will use, we will test if any pixel within the bounding box is "inside" the triangle or not to decide if we should color the pixel.)



A triangle and its 2D bounding box

We will start with very simple provided base code that writes out an image file - this is where we will be "drawing" rasterizing for this first assignment. (static render). The base code is only intended to demonstrate the use of the image writing library (you will not need to modify: stb_image_write.h, Image.h and Image.cpp - they are there just to be helpful.

Step 1 - test and understand the base code

We'll be using CMake and make throughout this quarter to try to support cross platform development. We'll also be using "out of source" builds (to make it easier to manage CMake across platforms). You also need a decent C++ compiler (that supports at least C++11).

If you're using the lab machines, g++ and CMake should already installed. To compile and run your code from the command line, follow these steps from the folder containing CMakeLists.txt.

```
> mkdir build
> cd build
> cmake ..
> make -j4
> ./Lab01
```

You should see some output on the console. Look at main.cpp to see what the command line arguments mean. Once you enter the correct arguments, you should see the following output file in the build directory.

If you're using your own computer, follow the steps at the end of the lab to install g++ and CMake.

Once you are set up and can see the above output go onto **Step 2.**

Step 2 - user defined data to control bounding box location in image

For today, we will just be playing with the provided image code to build up functionality that will be useful in your rasterizer. Your lab assignment today is very simple in general you will want to modify the code to

- read in three vertices which represent a triangle (see below for details)
- color all the pixels in the bounding box the color of your choice
- write out those pixels as a tga image (base code provided)

Starting with the provided code, make the list of command line arguments take the following options (in exactly this order)

- Output filename (e.g., foo.png)
- Image width (e.g., 512)
- Image height (e.g., 512)
- Vertex 1 x-coord (e.g., 100)
- Vertex 1 y-coord (e.g., 100)
- Vertex 2 x-coord
- Vertex 2 y-coord
- Vertex 3 x-coord
- Vertex 3 y-coord

Usage: *Lab01 filename width height vax vay vbx vby vcx vcy*
Thus for example – a valid program execution would look like:

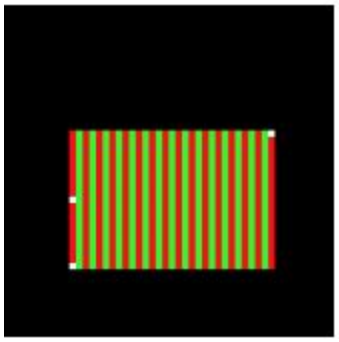
```
./Lab01 out.png 200 200 2 2 50 60 150 170
```

For today only, we will assume the triangle's vertices are in "window coordinates."

In other words, the data values are integer values that range from 0 to the value of width-1 of the window/image (or 0 to the height-1 value of the window/image). You may choose to have the vertices be 2D (although for the Assignment 1, they will need to be 3D). You will want to design a data structure to represent the triangle and its vertices. Draw the three vertices into the image, using three calls to image->setPixel().

The bounding box of the triangle is the box with the extents that will exactly bound that triangle. You will need to design a data structure to represent the bounding box. At minimum such a structure needs xmin, xmax, ymin, ymax. Using the provided image code, modify the color of all the pixels in the bounding box to be a color of your choice (which is different then the background).

You can also try a pattern based on the row and/or column. The three vertices and the bounding box should look something like this:



Write out the modified image and confirm your code is working as expected. Be sure to try various test cases with vertex positions in various orders. Make sure the bounding box completely covers the three vertices.

Lab check

For lab credit you will need to demonstrate your working code on several vertex tuples specified by the instructor or the TA by the end of lab on Thursday of this week.

Some directions setting up your own system

Here are some general ideas of how to get set up (each system can really vary so use your knowledge, your neighbors and online resources as needed). In general you need CMake and some (modern) c++ compiler to work (at least c++11).

Linux

Use a package manager to download cmake. For example, with ubuntu,

```
> sudo apt-get update
> sudo apt-get g++
> sudo apt-get install cmake
```

Follow the steps above to build and run the program from the command line.

OSX

You can use homebrew/macports or install these manually. You can either use Xcode or just compile from the terminal it is up to you. If you don't have the Xcode developer tools, you'll need to log in with your Apple ID and download them.

Then download CMake (http://cmake.org/download/)
Make sure the commands g++ and cmake work from the command prompt.
If you want to use Xcode as your IDE, then do the following from the folder containing CMakeLists.txt .

```
> mkdir build
> cd build
> cmake -G Xcode ..
```

This will generate Lab01.xcodeproj project that you can open with Xcode.
To run, change the target to Lab01 by going to Product -> Scheme -> Lab00. Then click on the play button or press
Command+R to run the application.
Edit the scheme to add command-line arguments or to run in release mode.

Windows

You'll need to download these manually.

- Visual Studio. Any version should work (but results will vary these instructions were for 2015, so use them as a general guide)
- CMake (http://cmake.org/download/).

Make sure to add CMake to the system path when asked to do so.

Make sure the command cmake works from the command prompt. On Windows, you'll be using Visual Studio as the IDE. Run the following from the folder containing CMakeLists.txt . to generate a solution file.

```
> mkdir build
> cd build
> cmake ..
```

By default on Windows, CMake will generate a Visual Studio solution file, Lab01.sln , which you can open by double-clicking. If you get a version mismatch, you may have to specify your visual studio version, for example:

```
> cmake -G "Visual Studio 14 2015" ..
```

Other versions of Visual Studio are listed on the CMake page (http://cmake.org/cmake/help/latest/manual/cmakegenerators.7.html).

To build and run the project, right-click on Lab01 in the project explorer and then click on "Set as Startup Project."

Then press F7 (Build Solution) and then F5 (Start Debugging).

To add a command line argument, right-click on Lab01 in the project explorer and then click on "Properties" and then click on "Debugging."