CPSC 314 Assignment 2

Term: September 2007, Instructor: Wolfgang Heidrich, heidrich@cs.ubc.ca, http://www.ugrad.cs.ubc.ca/~cs314

Due: Programming portion: Oct 25, 5pm; Theory: Oct 25 in class

Assignment 2.1: Line Clipping (5 Points)

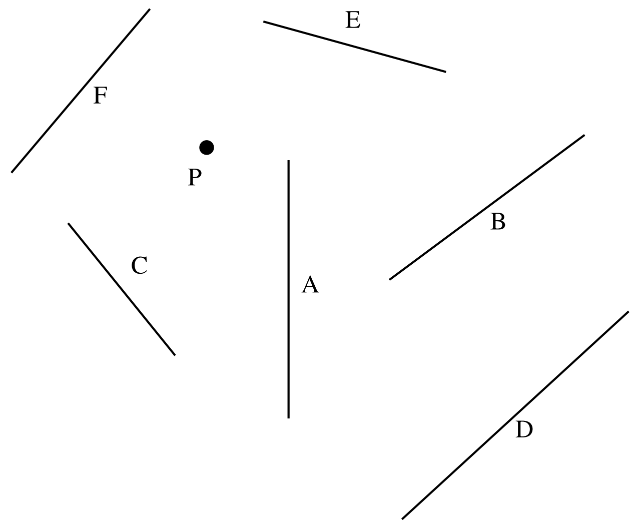
a) Briefly describe the meaning of outcodes and of window-edge coordinates (WEC) in line clipping.

b) Outline the general principle of the α-clipping algorithm for clipping lines to convex polygons.

c) Given two line segments, one from $(-2, 3)$ to $(2, -2)$, and the other from $(-2, 3)$ to $(2, 2)$, write down the steps of the α-clipping algorithm for a window of $x, y \in [-1 \ldots 1]$.

Assignment 2.2: BSP Trees (5 Points)

a) The diagram below shows a top-down-view of vertical walls in a “maze”. Construct the BSP tree that results from inserting the individual walls in alphabetical order. Use the convention where the right subtree (child) is located on the side that the normal points to. The positive halfspace of a wall is indicated by the side on which the letter is located.



b) What is the traversal order for a viewing position located in P?

Assignment 2.3: The Rendering Pipeline (30$_{(+10)}$ Points)

In this question, you will be implementing your own version of the geometric transformations involved in the graphics pipeline, as well as implementing the scan-conversion of smoothly-shaded polygons. You can earn bonus points for also implementing texture mapping and/or modeling a nice scene. These bonus points will carry over (i.e. they can be used to balance lost points in quizzes and the final).

Use the template code given online as a starting point for your code. You will not be using any OpenGL functions – all image pixels can be set using the `SetPixel(x,y,r,g,b)` function call. The functions you will be implementing are mostly replacements for the equivalent OpenGL functions. For example, myBegin() can be thought of as a replacement for `glBegin()`.

The file `libMath.cpp` provides many basic math functions, such as cross-products and matrix-vector multiplications. See `libMath.h` for a list of the available functions. The following is a suggested order for implementing and testing your code. After completing each part, ensure that the prior parts still work. The markers will be testing your code by running scenarios A through H without restarting your program. You should only make changes to the file `mygl.cpp`. A reference solution Linux executable `myglref` is provided for comparison.

(a) (6 points) Implement `myBegin()`, `myVertex()`, and `myEnd()` functions. You will be testing these functions using **scenario A**, which uses these functions to set a few points on the screen. The template code runs scenario A when "a" is typed on the keyboard. You will find it useful to implement a vertex data structure and create an array of these to hold all required information about vertices. In order to make things simple, you may assume that there are never more than 10 vertices specified between a `glBegin()` and a `glEnd()`. Implement myVertex() so that it stores the untransformed coordinates as well as the current color. In your `myBegin()` function, you will need to remember the current type of primitive being drawn. Your code only needs to handle `GL_POINTS` and `GL_TRIANGLES`. Your `myEnd()` function should call other functions of your own creation to transform all the points in the vertex list to viewport coordinates and then to draw them as points for the `GL_POINTS` mode. Test your code with scenario A. This sets the Model/View and Projection matrices to be identity matrices, and so object coordinates will effectively be the same normalized-device coordinates.

(b) (5 points) Implement triangle scan conversion using solid shading and no Z-buffer. Use the color assigned to the first vertex as being the color used for the triangle. Begin by computing the bounding box and making sure that this scan-converts correctly. Then make use of the implicit line equations of the triangle to only set those pixels that are interior to the triangle. Note that the bounding box should be correctly clipped to the window before scan conversion. Test this with Scenario B.

(c) (4 points) Implement smooth shading by linearly interpolating the colors for each pixel from the colors given for the vertices. Do this by computing barycentric coordinates for each rendered pixel. Test this with Scenario C.

(d) (5 points) Implement `myTranslate()`, `myRotate()`, and `myScale()` functions. Test this with Scenario D.

(e) (5 points) Implement the `myLookAt()` and `myFrustum()` functions, which will alter the Model/View and Projection matrices, respectively. Test this with Scenario E.

(f) (5 points) Implement a Z-buffer by interpolating Z-values from the vertices and by doing a Z-buffer test before setting each pixel. Test this with Scenario F. Note that a Z-buffer has already been declared for you in the template code. Use the `init_zbuf()` function to initialize it – this function is called for you every time a redraw is started.

(g) (Optional: 6 bonus points) Implement texture mapping by doing scan-converting the texture coordinates. Note that the perspective-correct scan-conversion of texture coordinates is slightly more complex than simply using the barycentric coordinates to produce a weighted combination of the vertex texture coordinates. When the `PCTexMap` boolean flag is true then perspective-correct texture coordinate interpolation should be applied.

Otherwise, linear texture coordinate interpolation should be applied. The file `libImage.cpp` provides the function `getTexel(image, u, v, r, g, b)`, which you can use to get back the RGB color of point (u, v), where $u, v \in [0, 1]$ in the image that serves as the current texture map. Test this with Scenario G.

(h) (Optional: 4 bonus points) Model and render a small-but-interesting scene with your rendering system, such as a stack of texture-mapped cubes, or a simple room with texture-mapped walls. Test this as Scenario H. Use animation and your own texture images if you like. Texture images should be in the PPM format.

Hand-in Instructions

You do not have to hand in any printed code. Create a README.txt file that includes your name, student number, and login ID for yourself, and any information you would like to pass on the marker. Create a folder called "assn2" under your cs314 directory and put all the source files, your makefile, and your README.txt file there. Also include any images that are used as texture maps. Do not use further sub-directories. The assignment should be handed in with the exact command:

```
handin cs314 assn2
```

Collaboration Policy

In this course, graded assignments are intended to be solved by individual students. Each student must submit their own individual solution.

If you have collaborated with other students on the solution to this assignment, this fact needs to be disclosed in the form below. **Note that disclosing a collaboration may result in reduced marks**. Likewise, if external resources (other than the course web pages and text book) were used for solving the assignment, they need to be listed below. **Failure to disclose a collaboration or external resources constitutes an act of academic misconduct, and will be reported to the dean**.

Declaration

I hereby declare that I have read and understood the above statement.

Name:

Student ID:

Signature and Date:

I have used the following external resources:

I have collaborated with the following individuals (explain degree of collaboration):