CPSC 314
Assignment 2

Due Tuesday March 7, 2006, 11:59pm

Answer the questions in the spaces provided on the question sheets. If you run out of space for an answer, use separate pages and staple them to your assignment.

Name: _____

Student Number: _____

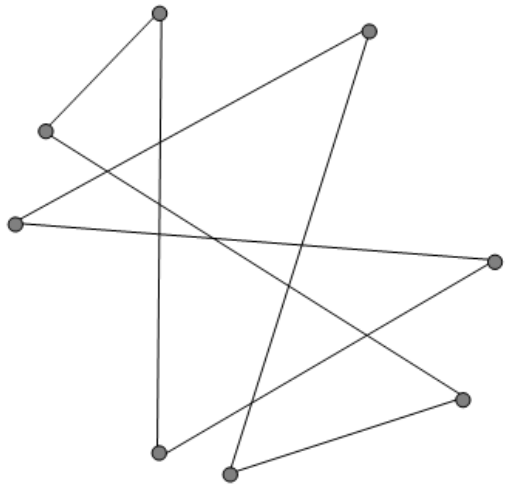
Question 1	/ 4
Question 2	/ 3
Question 3	/ 10
Question 4	/ 3
Question 5	/ 4
Question 6	/ 34
TOTAL	/ 58

1. (4 points) Perspective View Volumes

   Using a side view and a top view, sketch the perspective view volume that is defined by the following frustum parameters:
   $left = -1, right = 1, bot = -0.5, top = 0.5, near = 2, far = 100.$
   Also answer the following three questions. In what coordinate system is the viewing frustum defined ? What is the horizontal and vertical field of view, as measured in degrees? What should the aspect ratio ($width/height$) of the final on-screen window be so that the rendered world will appear undistorted?

2. (3 points) General Scan Conversion

   Arbitrary polygons are often scan converted by intersecting each scanline with all polygon edges, sorting the intersection points, and then using the parity of the intersection count in order to determine which regions to fill. For the following non-simple polygon, indicate which regions the parity-based scan conversion would fill in by shading them.



3. Scan Conversion and Interpolation

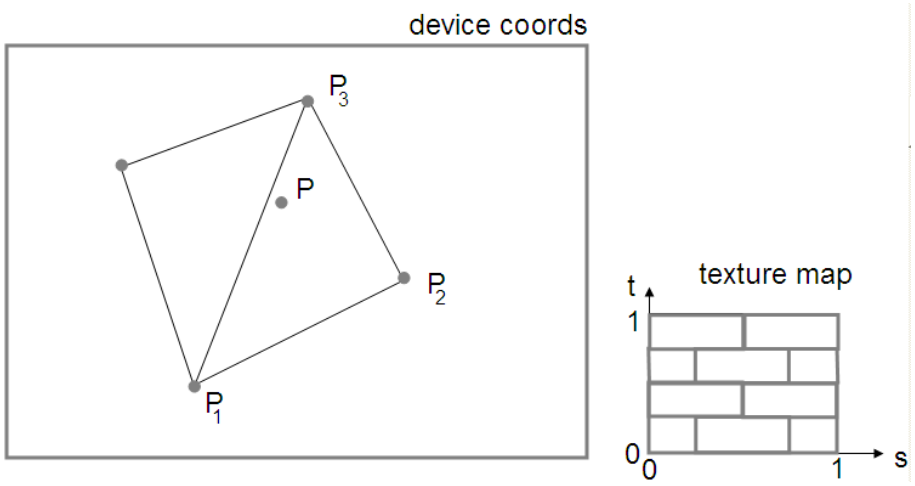
The figure below shows a triangle in device coordinates, with vertices

$P_1(100, 20), P_2(200, 100), P_3(150, 220)$

and a point $P(140, 160)$ inside the triangle. A set of values $V(z, s, t)$ that are associated with each vertex and are to be interpolated across the triangle are

$V_1(1.5, 0.2, 0), V_2(1, 0.7, 0), V_3(1.2, 0.7, 0.8)$.

As shown below, (s, t) are the *texture coordinates*, which indicate what parts of the *texture map* should be mapped onto the triangle.



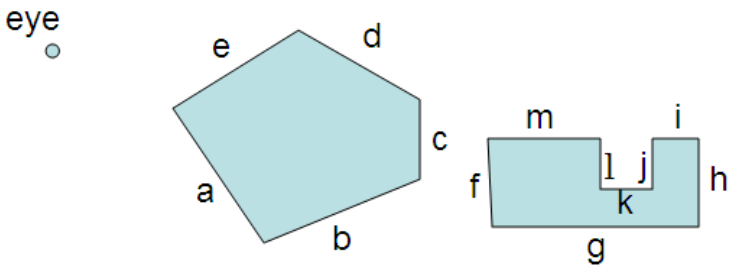
(a) (1 point) By sketching on the brick texture map, indicate which portion of the brick texture will get mapped to the triangle and which point in the texture map will be used to colour point P. By sketching on the final on-screen triangle, indicate how the brick pattern will appear on the on-screen triangle.

(b) (3 points) Give the implicit plane equation for the given triangle. I.e., $Ax + By + Cz + D = 0$. Compute the value of z for point P using your plane equation. Show your work.

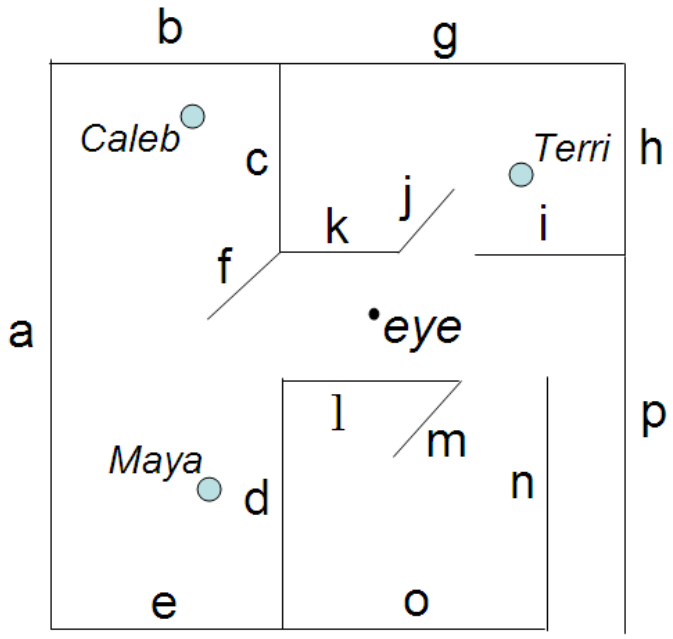
(c) (6 points) Compute the barycentric coordinates for point P. Compute z and s, t for point P using the Barycentric coordinates.

4. (3 points) Backface Culling

The edges shown below represent faces forming a closed solid. Which faces would be removed by backface culling? Show your work.



5. BSP Visibility



(a) (2 points) The edges in the scene above represent 3D polygons in an environment which consists of several rooms and several characters (Maya, Caleb, and Terri). In the space to the right of the figure, draw the BSP tree that is constructed for the scene by inserting the polygons in alphabetical order into the tree. Assume that the letter that labels each polygon sits in the '+' half space of the polygon. Place the '+' half-spaces on the right side of your tree. Add the characters into your BSP tree after having added all the other polygons. If two polygons are co-planar, i.e., i and k, you can have the polygon that is inserted later into the tree, i.e., k, share the the node for the earlier polygon in the BSP tree, i.e., i.

(b) (2 points) Give the back-to-front ordering that is produced for the given eye viewpoint. Show your work.

6. Implementing the Graphics Pipeline

   You may work on this question in pairs. The theory portion of the assignment must still be completed individually.

   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 and texture-mapped polygons. Use the template code given online (follow the assignment 2 links) 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 file `libImage.cpp` provides the function `getTexel(image, u, v, r, g, b)`, which you can use to get back the RGB colour of point $(u, v), u, v \in [0, 1]$ in the image that serves as the current texture map.

   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 colour. 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 ModelView and Projection matrices to be identity matrices, and so obj 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 which 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 colours for each pixel from the colours given for the vertices. Do this by computing barycentric coordinates for each rendered pixel. Test this with Scenario C.

(d) (3 points) Implement myTranslate(), myRotate(), and myScale() functions. Test this with Scenario D.

(e) (4 points) Implement the myLookAt() and myFrustum() functions, which will alter the ModelView and Projection matrices, respectively. Test this with Scenario E.

(f) (4 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 everytime a redraw is started.

(g) (4 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. Test this with Scenario G.

(h) (4 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 for your partner), 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
```