CPSC 414
Assignment 2

Due Tuesday March 9, 2004, 11:59pm

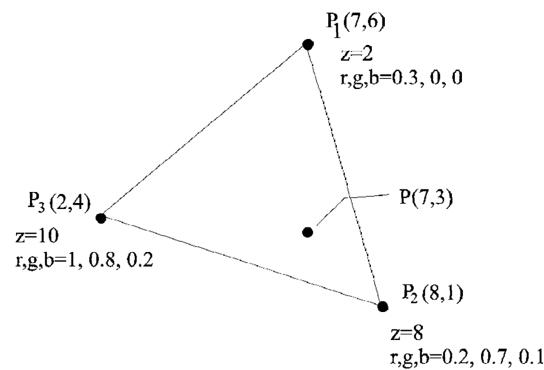
> Answer the questions in the spaces provided on the question sheets. If you run out of space for an answer, use separate pages and staple them to your assignment.

Name: ____________________

Student Number: ____________________

Question 1	/ 13
Question 2	/ 3
Question 3	/ 4
Question 4	/ 7
Question 5	/ 34
TOTAL	/ 61

1. Scan Conversion and Interpolation



(a) (3 points) Give the implicit plane equation for the triangle shown above. I.e., $Ax + By + Cz + D = 0$. Show your work.

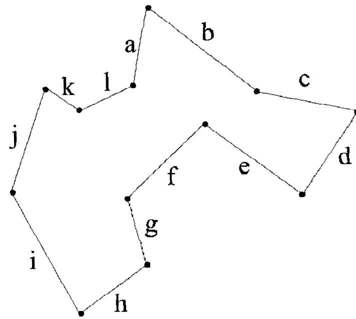
(b) (1 point) Compute the value of z for point P using your plane equation.

(c) (6 points) Compute the barycentric coordinates for point P. Compute z and r, g, b for point P using the Barycentric coordinates.

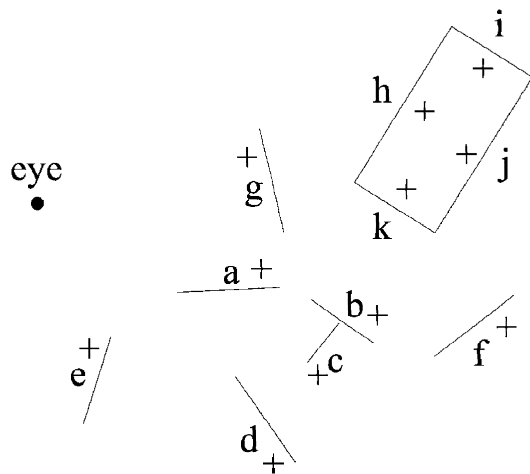
(d) (3 points) We can compute a normalized implicit equation for a plane, $F'(x, y, z) = A'x + B'y + C'z + D' = 0$ such that $|N'| = 1$ where $N' =< A', B', C' >$. Show that $|F'(P)| = d$, where d is the minimal distance from point P to the plane. Hint: evaluate $F'(P')$, where $P' = P + dN'$ and P is a point on the plane.

2. (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.



3. BSP Visibility



   (a) (2 points) The edges in the scene below represent polygons in 3D. In the space to the right of the above figure, draw the BSP tree that is constructed for the scene by inserting the polygons in alphabetical order into the tree. Place the '+' half-spaces on the left side of your tree.

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

4. Clipping

   You are given a VCS perspective viewing frustum which is given by the following parameters: $near = 1, far = 10, left = -0.5, right = 0.5, bot = -0.5, top = 0.5$.

   (a) (3 points) Write the 6 implicit plane equations that define the viewing frustum in VCS. Ensure that $F(P) > 0$ for points inside the view volume.

   (b) (4 points) Clip the VCS triange $P_1(0, 1, -4)P_2(1, 0, -3)P_3(2, 2, -2)$ to the viewing frustum given above.

5. Implementing the Graphics Pipeline

   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 all mostly replacements for the equivalent OpenGL functions. For example, myBegin() can be thought of as a replacement for glBegin().

   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.

   (a) (6 points) Implement myMatrixMode(), myLoadIdentity(), 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 mplement 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 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.

   (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. 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 file that includes your name, your login ID, and any information you would like to pass on the marker.

Create a folder called 'assn2' under your cs414 directory and put all the source files, your makefile, and your README 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 cs414 assn2
```