CPSC 314
Assignment 2

Due 4PM, Friday, Nov 4, 2005

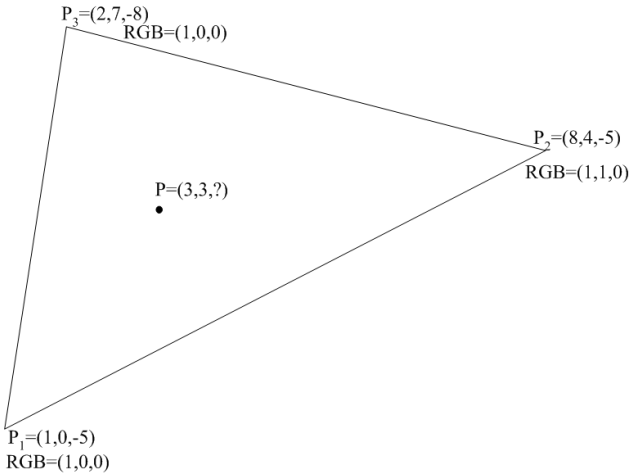
Answer the questions in the spaces provided on the question sheets. If you run out of space for an answer, use separate pages and staple them to your assignment.

Name: _____

Student Number: _____

Question 1	/ 12
Question 2	/ 3
Question 3	/ 3
Question 4	/ 7
Question 5	/ 5
Question 6	/ 70
TOTAL	/ 100

1. Scan Conversion and Interpolation



(a) (2 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) (1 point) Compute the barycentric coordinates for point P using the standard areas formula.

(d) (1 point) Compute the barycentric coordinates for point P using the alternative (bilinear) formula. Show your work.

(e) (1 point) Compute z and r, g, b for point P using the Barycentric coordinates.

(f) (1 point) Write down the OpenGL perspective matrix for the following frustum dimensions: $right = 2$,$left = -2$,$top = 2$,$bottom = -2$, $near = .1$, $far = 10$.

(g) (1 point) Compute the coordinates for P_1, P_2, P_3 after the perspective transformation is applied. Show your work.

(h) (1 point) Compute a new plane equation $Ax + By + Cz + d = 0$ for the triangle after the transformation. Show your work.

(i) (1 point) Compute the value of z for point P using the new plane equation.

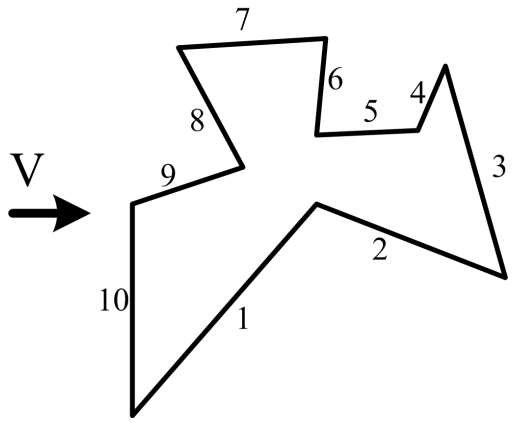
(j) (2 points) Compute the barycentric coordinates for point P after transformation.

2. (3 points) Clipping

   Use the definition of convexity to prove that the intersection of two convex objects is convex.

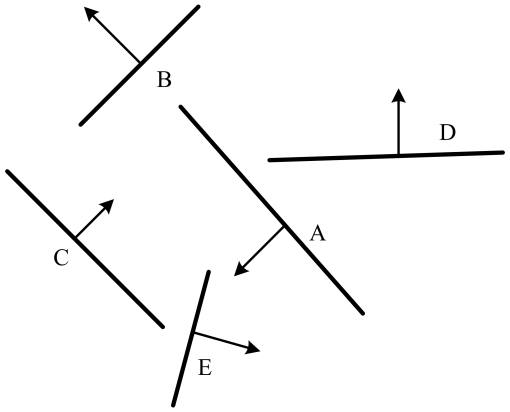
3. (3 points) Backface Culling

   The edges shown below represent faces forming a closed solid. Given the view direction V which faces will be removed by backface culling? Show your work.

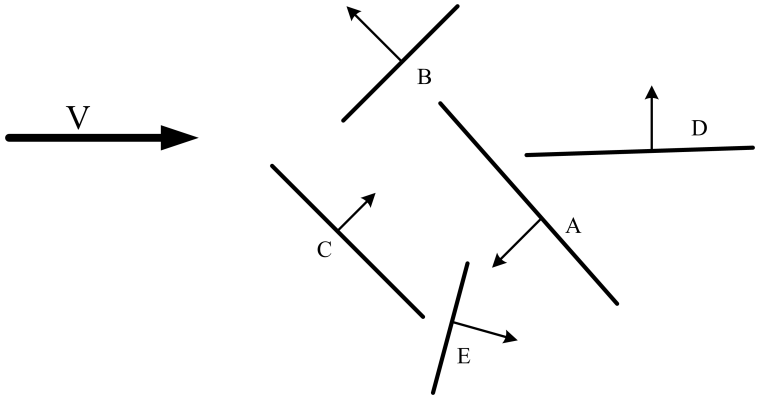


4. BSP Trees

   (a) (4 points) Construct the BSP tree for the segments shown below (use alphabetical insertion order for the segments, when possible). Use the convention where the right subtree (child) is located on the side that the normal points to. Show your work.



(b) (3 points) Given the view direction as shown below, describe the complete traversal order of your BSP tree during rendering.



5. (5 points) Bresenham

Write the Bresenham algorithm for rasterizing a line from (x_1, y_1) to (x_2, y_2) where $x_1 \geq x_2$, $y_2 > y_1$ and $x_1 - x_2 < y_2 - y_1$.

6. 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. This question of the assignment (and only this one) can be done in pairs (see submission procedure below). Use the template code given online (www.ugrad.cs.ubc.ca/∼cs314/Vsep2005/a2/a2_templ.zip) as a starting point for your code. To see how your results should look (up to roundup errors) check (www.ugrad.cs.ubc.ca/∼cs314/Vsep2005/a2/a2_refsol.zip). Note that the reference solution does not implement the bonus (transparency). You WILL NOT be using any OpenGL functions – all image pixels should 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 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 L without restarting your program.

   (a) (5 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, GL_LINES, and GL_POLYGON. 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.

   (b) (15 points) Implement line drawing using Bresenham algorithm. Note that you need to take care of eight slightly different cases, based on slope. Clip the line to the window. Test your code with scenarios B and C (B tests the ability to draw multiple lines within GL_LINES and clipping, C tests your Bresenham implementation).

   (c) (15 points) Rasterize your polygons. You should first triangulate each poligon and then rasterize the resulting triangles. You can assume that the polygons are always convex and the vertices are oriented counter-clockwise around the normal. To debug this part, you might want to draw the triangles you get in different color or draw their edges using Bresenham. 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 scan-conversion algorithm to set the pixels which are interior to the triangle. Note that the bounding box should be correctly clipped to the window before scan conversion (this way you do not need to explicitly clip the traingle itself). Test this with scenarios D and E (D for triangles, E for polygons).

(d) (5 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. Note that different triangulations of the polygons can result in slightly different interpolated colors. Test this with Scenario F.

(e) (5 points) Implement myTranslate(), myRotate(), and myScale() functions. Test this with Scenario G.

(f) (5 points) Implement the myLookAt() and myFrustum() functions, which will alter the ModelView and Projection matrices, respectively. Test this with Scenario H.

(g) (10 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 I. 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.

(h) (5 points) Use scenario J to test your code on real-life models (obj format).

(i) (5 points) Model and render a small-but-interesting scene with your rendering system, using provided obj models or other models you find on the web. Use animation if you like. Code your new scene as scenario K.

(j) (15 points) (bonus) Implement transparency by replacing your z-buffer, with alpha-buffer. This will require you to replace calls to myColor(float r, float g, float b) with calls to myColor(float r, float g, float b, float alpha). You will also need to fill in the code in myColor(float r, float g, float b, float alpha) and provide all the necessary data structures. Most significantly, this will requre you to ADD your own structure on top of the z-buffer to store alpha and depth values. Do NOT change the structures given in the template. Test this with Scenario L. Do not try to implement this part before ALL the rest of your assignment is working.

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. If you do the programming assignment in pairs, write both your names in the README.

If you submit the assignment in pairs, you must do and submit the theory part of the assignment individually. However, for programming part the following submission procedure should be done by ONLY one of you.

Create a folder called 'assn2' under your cs314 directory and put all the source files, your makefile, and your README file there. Do not use further sub-directories.

The assignment should be handed in with the exact command:

```
handin cs314 assn2
```