

CPSC 314

Assignment 2

Due 4PM, Friday, Nov 12, 2004

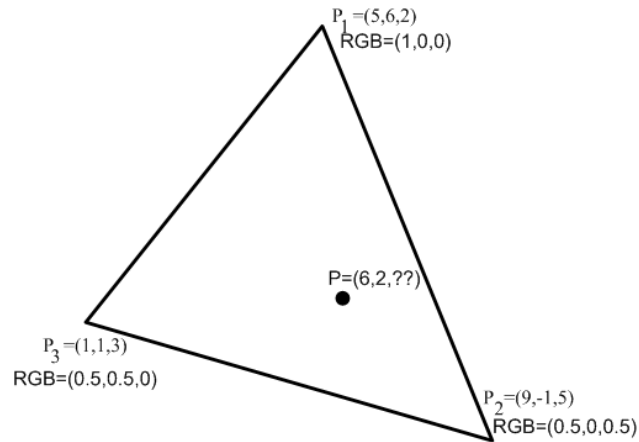
Answer the questions in the spaces provided on the question sheets. If you run out of space for an answer, use separate pages and staple them to your assignment.

Name: _____

Student Number: _____

Question 1	/ 12
Question 2	/ 7
Question 3	/ 8
Question 4	/ 73
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) (3 points) Compute the barycentric coordinates for point P . Compute z and r, g, b for point P using the Barycentric coordinates.

- (d) (3 points) Transform the triangle using the following perspective transformation ($d = .5$ and $\alpha = .1$). Compute the new coordinates for P_1, P_2, P_3 and a new plane equation $Ax + By + Cz + d = 0$. Show your work.

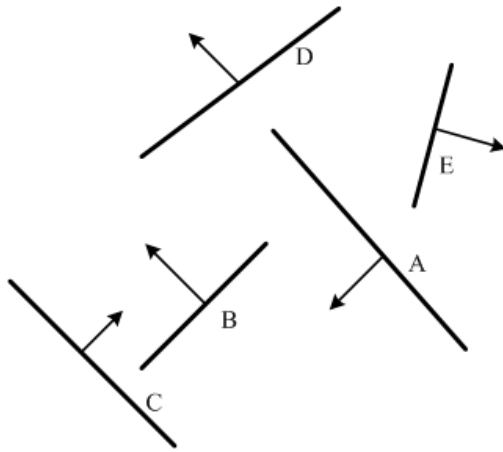
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{.5}{.5-.1} & \frac{1}{.5} \\ 0 & 0 & \frac{-.1+.5}{.5-.1} & 0 \end{pmatrix}. \quad (1)$$

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

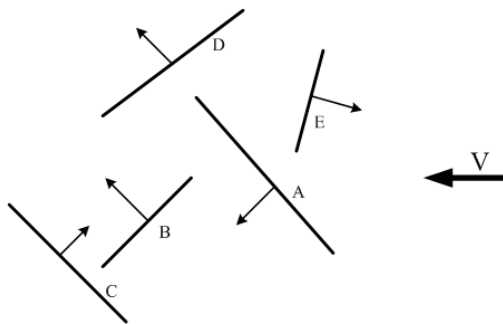
- (f) (2 points) Compute the barycentric coordinates for point P after transformation.

2. 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.

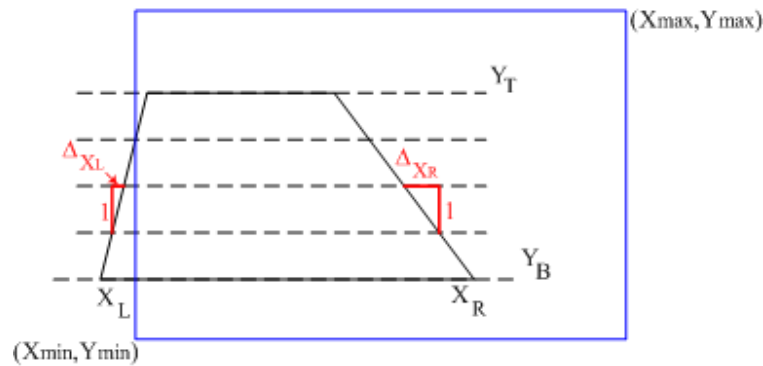


- (b) (3 points) Given the view direction as shown below, describe the complete traversal order of your BSP tree during rendering.



3. (8 points) Clipping

Instead of clipping the triangles in a continuous setting as shown in class, it is possible to clip them directly during scan-conversion. Write the pseudo-code for the “scanTrapezoid” function which does clipping as part of the conversion. Hint: use the bounding box of the trapezoid. The arguments passed to “scanTrapezoid” are the parameters of the trapezoid - $X_L, X_R, Y_B, Y_T, \Delta_{X_L}, \Delta_{X_R}$ and the window parameters $X_{min}, X_{max}, Y_{min}, Y_{max}$ as shown in the figure below



4. 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 (www.ugrad.cs.ubc.ca/~cs314/Vsep2004/a2/a2_templ.zip) as a starting point for your code. 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 J without restarting your program.

- (a) (8 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 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`, `GL_LINE_LOOP`, 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.

- (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 scenario B.
- (c) (15 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 edgeWalking 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. Use your pseudocode from question 4 as the basis for the scan conversion code. Test this with scenario C.
- (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. Test this with Scenario D.
- (e) (5 points) Implement `myTranslate()`, `myRotate()`, and `myScale()` functions. Test this with Scenario E.

- (f) (5 points) Implement the `myLookAt()` and `myFrustum()` functions, which will alter the `ModelView` and `Projection` matrices, respectively. Test this with Scenario F.
- (g) (15 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 G. 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) (bonus) Implement texture mapping by 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 (hint: Question 1). Test this with Scenario H.
- (i) (5 points) Use scenario I to test your code on real-life models (obj format). 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 and your own texture images if you like. Texture images should be in the PPM format. Code your new scene as scenario J.

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 cs314 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 cs314 assn2
```