

# CPSC 314

## Assignment 2

Due Friday November 3, 2006, 4pm

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	/ 6
Question 2	/ 10
Question 3	/ 3
Question 4	/ 4
Question 5	/ 34
TOTAL	/ 57

## 1. Perspective View Volumes

- (a) (2 points) Using a side view and a top view, sketch the perspective view volume that is defined by the following frustum parameters:  
 $left = -2, right = 2, bot = -1, top = 1, near = 4, far = 200$ .

- (b) (2 points) 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?

- (c) (2 points) Digital cameras are different than film cameras in that the imaging area is smaller in size than standard 35mm film. Suppose that you take a photograph with a digital camera using a  $20\text{mm}$  lens, and having a digital imaging area of  $8.8 \times 6.6\text{mm}$ . These are thus the dimensions of the imaging area of the light-sensitive CCD chip in the camera.  $20\text{mm}$  lens is the focal length of the lens, meaning that parallel rays of light entering the lens will cross at a point  $20\text{mm}$  behind the lens. Give the horizontal and vertical field of view for the above scenario, as measured in degrees. Note that light passes through the center of an ideal lens without having its direction altered, irrespective of its incoming direction.

## 2. Scan Conversion and Interpolation

A triangle has device coordinates  $P_1(20, 30)$ ,  $P_2(100, 20)$ , and  $P_3(60, 90)$ . You wish to interpolate a value  $v$  for point  $P(50, 40)$ , given the value of  $v$  at the vertices:  $v_1 = 5$ ,  $v_2 = 8$ ,  $v_3 = 20$ .

- (a) (1 point) Sketch the triangle and the point  $P$ .
- (b) (3 points) Compute an interpolated value  $v$  for point  $P$  by developing an implicit plane equation for the given triangle of the form  $Ax + By + Cv + D = 0$ . Compute the value of  $v$  for point  $P$  using your plane equation. Show your work.

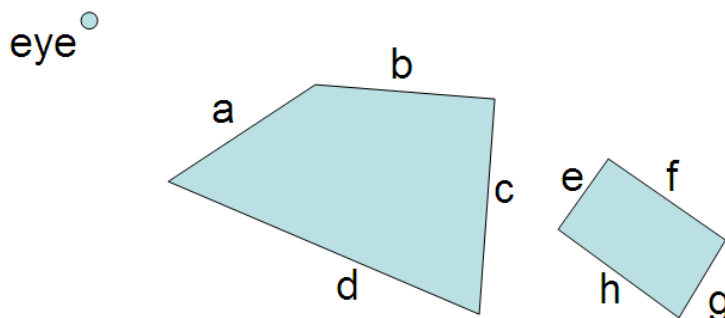
- (c) (4 points) Compute the barycentric coordinates for point  $P$ . Compute  $v$  for point  $P$  using the Barycentric coordinates.

(d) (1 point) Are there any triangles that would cause problems for the above interpolation procedures? If so, describe these cases.

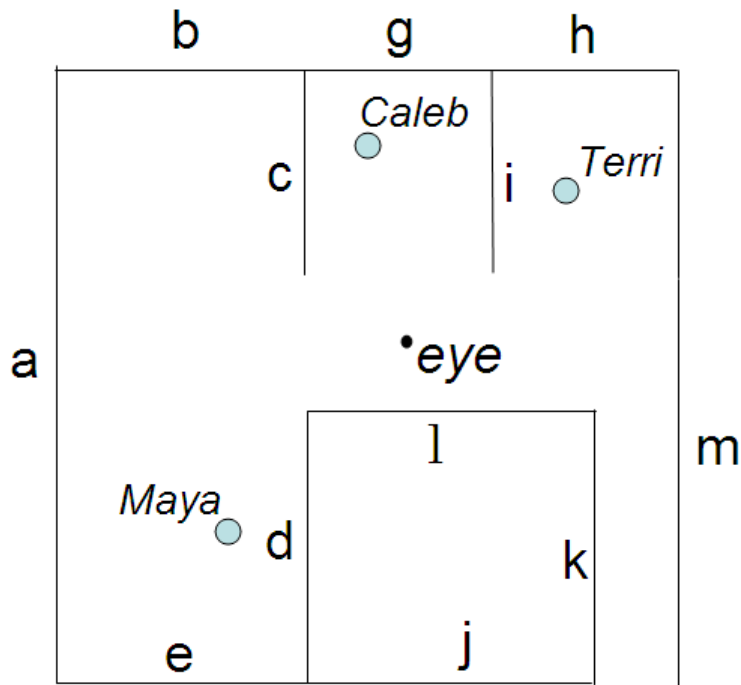
(e) (1 point) Sketch the worst-case rendering scenario for the rendering scheme that walks through all the pixels in the axis-aligned bounding box of a triangle. I.e., what type of triangle will cause the largest fraction of pixels that are evaluated to be 'outside'?

3. (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 on the diagram.



4. 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 as nodes, after having added all polygons. If two polygons are co-planar, i.e., *b* and *g*, you can have the polygon that is inserted later into the tree, i.e., *g*, share the the node for the earlier polygon in the BSP tree, i.e., *g*.
- (b) (2 points) Give the back-to-front ordering that is produced for the given eye view-point. Show your work.

## 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 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. The `init_zbuf()` function is already called for you to initialize it everytime a redraw is started, although you are free to add extra functionality there.
- (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. Typing 't' will toggle this flag on and off. 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
```