CPSC 314
Assignment 2

Due Monday November 3, 2014

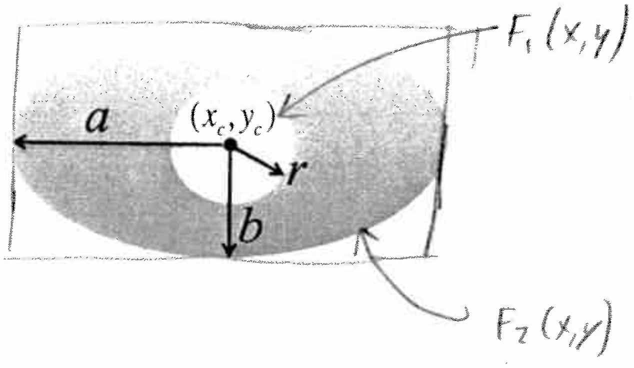
Answer the questions in the spaces provided on the question sheets. If you run out of space for an answer, use separate pages and staple them to your assignment.

Name: Solution

Student Number: ____

Question 1	/ 5
Question 2	/ 9
Question 3	/ 6
Question 4	/ 10
Question 5	/ 30
TOTAL	/ 60

1. (5 points) Scan Conversion
   Give the pseudocode for scan converting the ellipse shown below. It is centred at x_c, y_c, has a major axis length of $2a$, a minor axis length of $2b$, and contains a circular hole of radius r. Use implicit equations to develop your solution.



$$r^2 = (x - x_c)^2 + (y - y_c)^2$$

$$F_1(x,y) = (x - x_c)^2 + (y - y_c)^2 - r^2$$

$= 0$ on circle

> 0 outside

< 0 inside.

$$1 = \frac{(x - x_c)^2}{a^2} + \frac{(y - y_c)^2}{b^2}$$

$$F_2(x,y) = 1 - \frac{(x - x_c)^2}{a^2} - \frac{(y - y_c)^2}{b^2}$$

$= 0$ on circle

> 0 inside

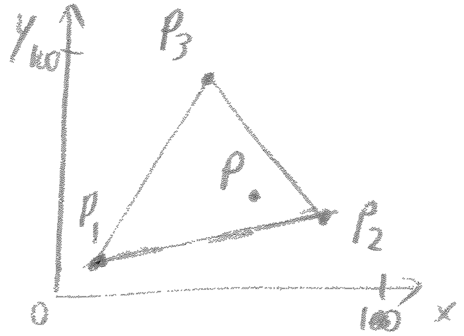
< 0 outside.

```
For (x = x_c - a; x ≤ x_c + a; x++) {
    for (y = y_c - b; y ≤ y_c + b; y++) {    // for each pixel in bounding box
        if (F_1(x,y) ≥ 0  and  F_2(x,y) ≥ 0)
            SetPixel(x,y)
    }
}
```

2. Scan Conversion and Interpolation

A triangle has device coordinates $P_1(10,10)$, $P_2(80,20)$, and $P_3(40,90)$. You wish to interpolate a value v for point $P(60,30)$, given the value of v at the vertices: $v_1 = 40, v_2 = 20, v_3 = 30$.

(a) (1 point) Sketch the triangle and the point P.



$$P(\alpha,\beta,\gamma) = \alpha P_1 + \beta P_2 + \gamma P_3$$

(b) (4 points) Develop a plane equation for v as a function of x and y. You can use Matlab or an online linear equation calculator (Google this) to solve a set of linear equations for your plane parameters. Compute v for point P using the plane equation.

Plane equation: $Ax + By + C = V$

P_1: $10A + 10B + C = 40$

P_2: $80A + 20B + C = 20$

P_3: $40A + 90B + C = 30$

Solve for A,B,C: $A = -0.2830$, $B = -0.0189$, $C = 43.02$

for point P:

$A \cdot 60 + B \cdot 30 + C = \boxed{25.5}$

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

$$F_{12}(x,y) = x(y_2 - y_1) + y(x_1 - x_2) + x_2y_1 - y_1y_2$$
$$= x(10) + y(-70) + 800 - 200$$
$$= 10x - 70y + 600$$

$$F'_{12}(x,y) = F_{12}(x,y) / F_{12}(x_3,y_3)$$
$$= F_{12}(x,y) / ((10)(40) - (70)(90) + 600)$$
$$= F_{12}(x,y) / (-5300)$$

$$F_{23}(x,y) = x(y_3 - y_2) + y(x_2 - x_3) + x_3y_2 - x_2y_3$$
$$= x(70) + y(40) + 800 - 7200$$
$$= 70x + 40y - 6400$$

$$F'_{23}(x,y) = F_{23}(x,y) / F_{23}(x_1,y_1)$$
$$= F_{23}(x,y) / ((70)(10) + (40)(10) - 6400)$$
$$= F_{23}(x,y) / (-5300)$$

$$\alpha = F'_{23}(60,30) = ((70)(60) + (40)(30) - 6400) / (-5300) = \frac{-1000}{-5300} = 0.1887$$

$$\gamma = F'_{12}(60,30) = ((10)(60) - (70)(30) + 600) / 5300 = \frac{-900}{-5300} = 0.1698$$

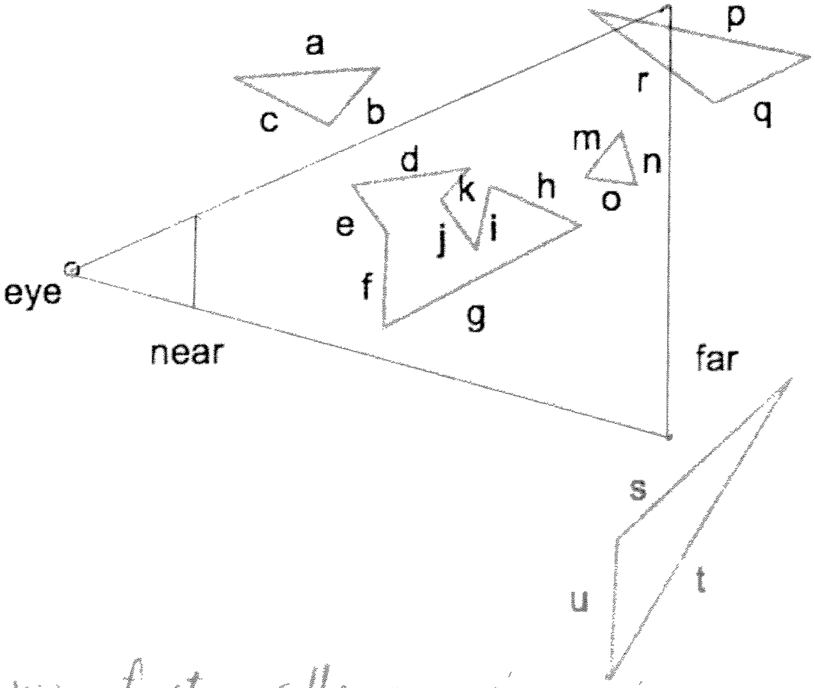
$$\beta = 1 - \alpha - \gamma$$
$$= 0.6415$$

CPSC 314 Assignment 2

$V = \alpha V_1 + \beta V_2 + \gamma V_3$
$= 25.5$

3. (6 points) Culling

For the following scene, shown as a side-view of VCS, list which polygons would be culled by (a) view frustrum culling, and (b) back-face culling. Assume that each segment with a letter represents a face. Consider both types of culling independently of each other.



(a) view frustum culling: cull if both vertices of a given face are "outside" with respect to one of the frustum planes.

cull: [a, b, c, q, u]

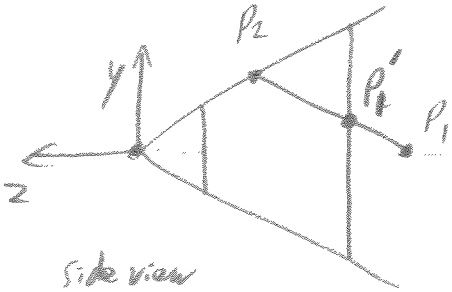
(b) back face culling: cull if the eye is <u>below</u> the plane that embeds the given polygon.

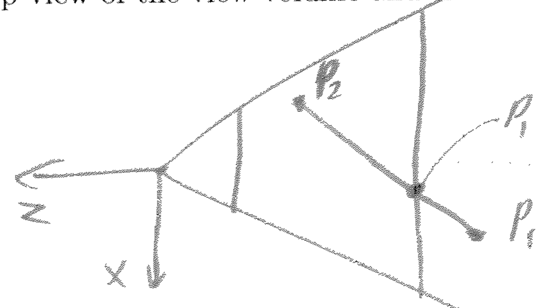
cull: [a, b, d, g, h, j, k, n, p, q, t]

4. Clipping

Suppose that a perspective view-volume is defined by `near=1, far=4, bot=-1, top=1, left=-1,` ~~`far`~~ right `=1`. Consider the line defined by the VCS coordinates $P_1(2,0,-5)P_2(-1,2,-2)$.

(a) (3 points) Sketch a side-view and top-view of the view-volume and the line.





(b) (3 points) Determine if view-frustum culling can be applied to the line, i.e., if both vertices are "outside" with respect to any one of the six view frustum planes. Use the implicit plane equations.

Implicit view volume eqns (from p.6 of visibility notes)

Left: $x + \frac{left}{near} \cdot z = x - z$ $P_1: 2-(-5)=7$ (inside) $P_2: -1-(-2)=1$ (inside)

right: $-x - \frac{right}{near} \cdot z = -x - z$ $P_1: -2-(-5)=3$ (inside) $P_2: -(-1)-(-2)=3$ (inside)

top: $-y - \frac{top}{near} \cdot z = -y - z$ $P_1: -0-(-5)=5$ (inside) $P_2: -2-(-2)=0$ (on plane $\rightarrow$ inside)

bottom: $y + \frac{bottom}{near} \cdot z = y - z$ $P_1: 0-(-5)=5$ (inside) $P_2: 2-(-2)=\cancel{4}$ (inside)

front: $-z-1$
$= 4$ for P_1
$= 1$ for P_2 } inside

back: $z+4$
$= -1$ for P_1 (outside)
$= 2$ for P_2 (inside)

$\boxed{\text{Cannot cull}}$

(c) (1 point) Based on your work for the question above, determine the view-frustum planes that the line intersects.

We need to clip with respect to the $\boxed{\text{back}}$ plane (i.e., far): P_1 is inside, P_2 is outside

(d) (3 points) Compute the final clipped version of line P_1P_2 in VCS. Show your work.

$$P(t) = P_1 + t(P_2 - P_1)$$

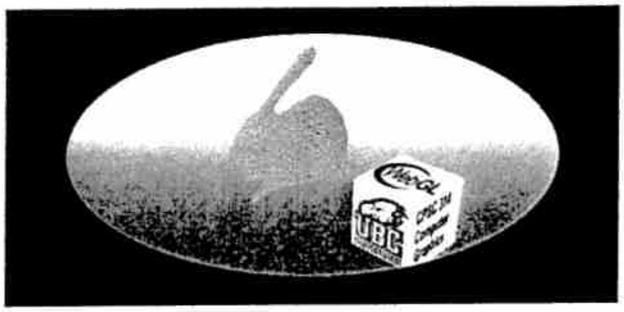
$$t \text{ for intersection: } t = \frac{-F(P_1)}{F(P_2)-F(P_1)} = \frac{-(-1)}{2-(-1)} = \frac{1}{3}$$

$$P_1'(t) = \begin{bmatrix} 2 \\ 0 \\ -5 \end{bmatrix} + \frac{1}{3}\begin{bmatrix} -3 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2/3 \\ -4 \end{bmatrix} = P_1'$$

Final clipped segment is $P_1'P_2$

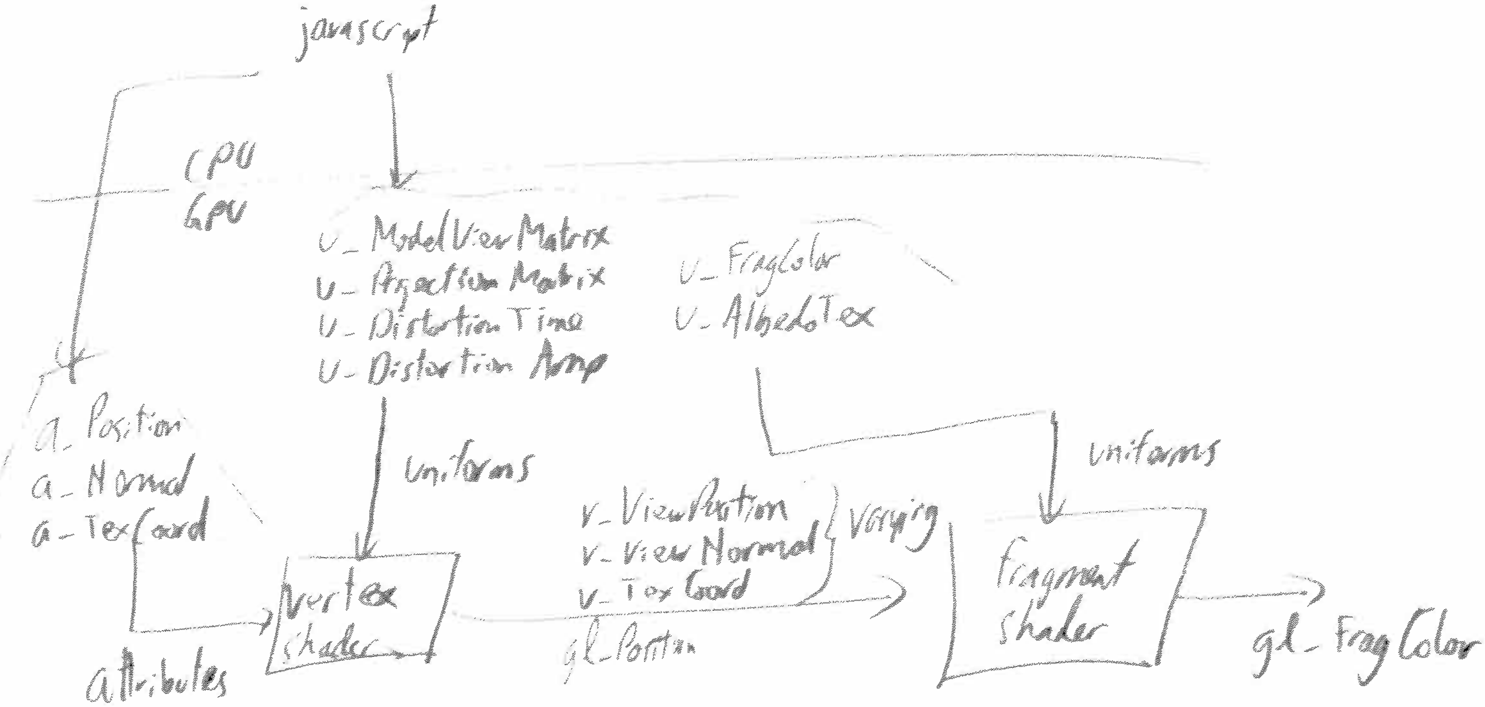
5. Coding: Texture mapping, vertex shaders, and fragment shaders

   The objective of this coding question is to gain some hands-on experience with using texture mapping, vertex shaders, and fragment shaders. You will be starting with template code (see the lectures web page for the link) and making a number of changes to it. A screen shot of what the final solution might look like is shown below.



   Complete the following modifications to the template code. You need not exactly follow the given order, as most of the modifications are independent of each other.

   (a) (5 points) Examine the vertex shader (`mesh_vs.glsl`) and the fragment shader (`mesh_fs.glsl`) for the available input variables, which are listed at the beginning of the shader. In the space below, sketch a diagram with illustrated blocks for the javascript application code, the GPU memory, the vertex shader, the fragment shader, and the final image buffer. On this diagram, list the attribute, uniform, and varying variables and illustrate between which blocks they are communicated. Also illustrate the default output variables for the vertex shader and fragment shader. Include your diagram below as part of your paper assignment handin.



(b) (2 points) The ground texture is modeled using a textured plane, constructed with four vertices, which is defined in `a2.js`. Change the texture coordinates assigned to the four vertices such that the texture becomes much finer than it currently is, i.e., so that it repeats many more times.

(c) (2 points) Change the texture map that is used for the bunny from the UBC logo to something more appropriate. See the `loadTextures()` function for where the texture map file names are specified. Note that the dimensions of images used for texture mapping must by powers of 2, i.e., `256x256`, `512x512`, etc. If you like, use `psychedelic.png`. Or, better yet, take any image you like, and resize it using your favorite image resizing application. This is optional, but it can count as one of the "extras" you do for the last step.

(d) (4 points) Now you will be changing the texture mapping for the faces of the cube. First, view the image `ubcTexture.png`, which is currently used to texture map the cube. You should change the texture coordinates for each of the cube faces (see `a2.js`) in order to map the four individual sub-images in that texture map to individual faces, in a way that makes sense, i.e., upright and legible. See the solution image at the start of this question for an example.

(e) (3 points) Now take a look inside the fragment shader, (`mesh_fs.glsl`). The ultimate output of the fragment shader is `gl_FragColor`. Comment out the last line and add a new line that assigns `gl_FragColor = u_FragColor;`. Observe what this does, i.e., produce a simple flat-shaded rendering without textures. Now the goal for this step is to give your bunny a 'colour tinted' texture. Look for the code in `drawScene()` where `u_FragColor` is used before drawing. Change the colour assignments to something distinct and observe the resulting change in your 'flat shaded' rendered version. Now compute the product of the default colour with the texture-map colour and use that as the colour for the fragment. Note that the shading language allows for component-wise multiplication using a statement such as: `vec4 c = a*b;`, where `a` and `b` are also of type `vec4`. You should now be able to produce a texture mapped bunny with a desired color tint.

(f) (3 points) We'll now use the fragment shader to produce an elliptical viewing window. In the supplied fragment shader, you are already given the NDCS coordinates of the fragment being rendered. Use this to evaluate an implicit function, `f`, for a circle, which will appear as an ellipse on screen because of the aspect ratio. Replacing the default `f=1.0;` line with your function should result in the fragment being assigned the colour black when it is outside the elliptical border.

(g) (3 points) We'll now be making changes to the vertex shader. The goal in this step is to distort the geometry in the vertex shader. First, you could simply move the geometry as a function of time, given by `u_DistortionTime`. Thus adding an offset to one of the `x,y,z` components, in model space, according to `DistortionAmp*sin(c2*DistortionTime)` will result in the vertex shader adding a sinusoidal translation to the entire model over time. While all the objects are drawn using the same vertex shader, only the bunny should move because `DistortionAmp`

is set to zero before the cube and ground plane are drawn. Now, further change the distortion so that the offset also varies as a function of space,
i.e., `DistortionAmp*sin(c2*DistortionTime + c3*x)`, where `x` could be any of the point's `x,y,z` model coordinates. Hitting the spacebar will start and stop the advancement of `DistortionTime`.

(h) (3 points) We'll now introduce a simple model of colored fog using the fragment shader. You will see some template code in the fragment shader that blends the fog colour, white by default, with the computed fragment colour in order to compute a final pixel colour. First, understand the visual effect by setting `fogAmount=0.5` and also experimenting with fog colours other than white. Now compute fogAmount as a linear function of the viewing distance, which you have access to because `v_ViewPosition` is provided as a varying variable, and this represents the VCS coordinates for the current fragment. You will need to further clamp `fogAmount` to within the range `[0,1]`. This will mean that beyond some distance, the fragment will be completely fog coloured, while closer than some distance, the fragment will have no fog colour at all.

(i) (5 points) Develop your ideas of your own for augmenting the scene. You could add more bunnies, make the fog color vary with time and space, add some motion, etc.

Submit your code using `handin cs314 a2`.
Include a README.txt file that contains: (a) your name; (b) your student number; (c) any comments and explanations that you wish to include.