Computer Science

COMPSCI 372 S2 C – Exercise Sheet 3
Sample Solution

Q1: Download the "OpenGL Examples" demo at
http://www.cs.auckland.ac.nz/compsci372s2c/resources/OpenGL_ExamplesNET.zip

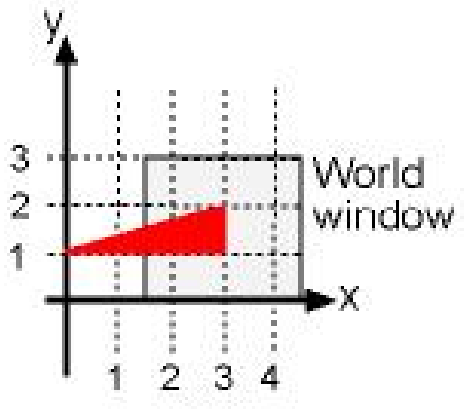
(a) Run the project "fog" which demonstrates three different types of distance attenuation to simulate smoke/fog covering an object. Which type of attenuation makes a far away object look most blurry?

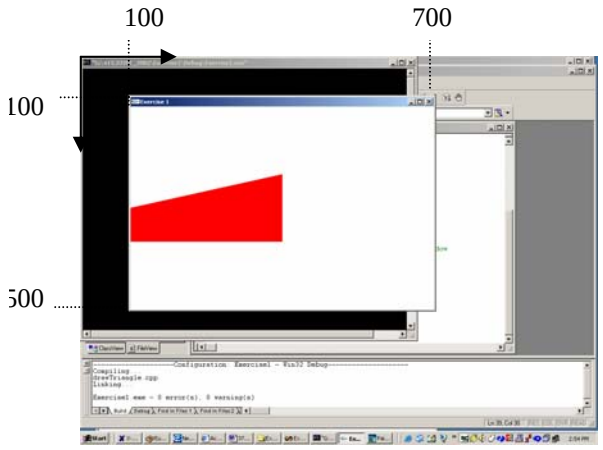
Solution:
OpenGL For blends a fog colour with the pixel colour of a rasterised object. The blending factor depends on the attenuation function used which can be linear, exponential or exponential squared (http://www.cs.auckland.ac.nz/compsci372s2c/resources/manpagesOpenGL/glFog.html). The user can specify parameters for these attenuation functions. In our example the exponential squared function results in the largest blending factor for fag when applied to the furthest away object, i.e. the object appears most blurry in this case.

(b) Run the project "dof" which simulates the out-of-focus depth-of-field effect you have when using a real camera. For what type of applications would such an effect be useful? Find an example for this on the Internet.

Solution:
Depth-of –field effects are a great way to emphasize objects by focusing on them and blurring the foreground and background. This effect is also sometimes used in games to give the player a hint on where to go next or which object to pick.
Here are some nice examples generated with Photoshop: http://www.photoshopsupport.com/photoshop-blog/06/10/index2.html

Q2: **(a)** Compute the world-to-viewport mapping which maps the world window with the coordinates (1.5, 4.5, 0.0, 3.0) [(left, right, bottom, top)] onto a window on the screen (the viewport). The window has a width of 600 pixels, a height of 400 pixels and its left-top corner is at the pixel (100,100) on the screen.





The image above shows on the left hand side the world coordinate system and the world window we want to display on the screen. The image on the right shows the resulting output on the screen. Note that the drawing window has a width and a height of 600 and 400 pixels, respectively, and that its top-left corner is at the position (100,100).

The coordinates of the world window are w.l=1.5, w.r=4.5, w.b=0, w.t=3 .
The coordinates of the viewport are: v.l=100, v.r=700, v.b=500, v.t=100

Inserting these values into the formulas from the lecture notes gives:

$$A = \frac{v.r - v.l}{w.r - w.l} = \frac{700-100}{4.5-1.5} = \frac{600}{3} = 200 \quad , \quad C = v.l - A * w.l = 100 - 200 * 1.5 = -200$$

$$B = \frac{v.t - v.b}{w.t - w.b} = \frac{100-500}{3-0} = \frac{-400}{3} \quad , \quad D = v.b - B * w.b = 500 - \frac{-400}{3} * 0 = 500$$

(b) Assume you draw a triangle with the world coordinate vertices (0.0,1.0), (3.0, 1.0), and (3.0,2.0) using the world-to-viewport mapping from (a). What are the pixel positions of the triangle corners on the screen? Which part of the triangle is inside your drawing window (i.e. the viewport)?

Using the solution from part (a) the screen coordinates of the triangle vertices are computed by using the world-to-viewport mapping

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} C \\ D \end{pmatrix} = \begin{pmatrix} 200 & 0 \\ 0 & -400/3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} -200 \\ 500 \end{pmatrix}$$

For vertex 1: $\begin{pmatrix} 200 & 0 \\ 0 & -400/3 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} -200 \\ 500 \end{pmatrix} = \begin{pmatrix} -200 \\ 1100/3 \end{pmatrix} \approx \begin{pmatrix} -200 \\ 367 \end{pmatrix}$

For vertex 2: $\begin{pmatrix} 200 & 0 \\ 0 & -400/3 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \end{pmatrix} + \begin{pmatrix} -200 \\ 500 \end{pmatrix} = \begin{pmatrix} 400 \\ 1100/3 \end{pmatrix} \approx \begin{pmatrix} 400 \\ 367 \end{pmatrix}$

For vertex 3: $\begin{pmatrix} 200 & 0 \\ 0 & -400/3 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \end{pmatrix} + \begin{pmatrix} -200 \\ 500 \end{pmatrix} = \begin{pmatrix} 400 \\ 700/3 \end{pmatrix} \approx \begin{pmatrix} 400 \\ 233 \end{pmatrix}$

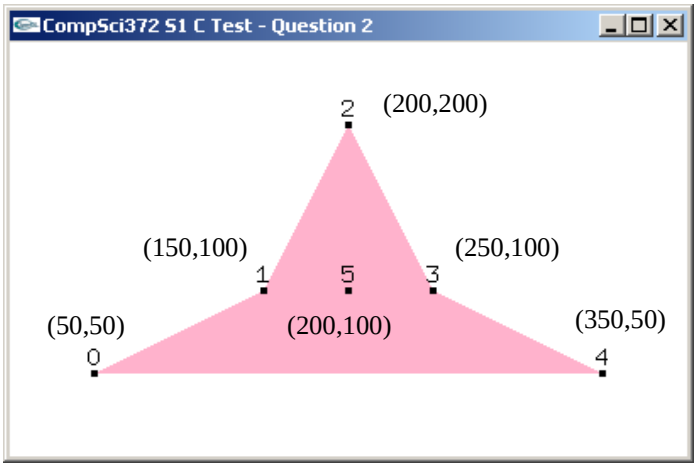
Looking at the screen image on the previous page it can be seen that these values are indeed correct.

Since the x-coordinates of the triangle vertices lie between 0 and 3, but the world window has its left boundary at the x-coordinate 1.5, only half the triangle is displayed in the drawing window (the removal of invisible parts of an object is called *clipping*)

(c) Write a program drawing the triangle from (b) using the world-to-viewport mapping from (a).

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glut.h>

void display(void)
{
      // clear all pixels in frame buffer
      glClear(GL_COLOR_BUFFER_BIT);

      // draw a red triangle
      glColor3f (1.0, 0.0, 0.0);    // (red,green,blue) components
      glBegin(GL_TRIANGLES);
      glVertex2f(0.0, 1.0);
      glVertex2f(3.0, 1.0);
      glVertex2f(3.0, 2.0);
      glEnd();

      // start processing buffered OpenGL routines
      glFlush ();
}

void init (void)
{
      // choose white as background colour (used in glClear)
      glClearColor (1.0, 1.0, 1.0, 0.0);

      // initialize view (simple orthographic projection)
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(1.5, 4.5, 0.0, 3.0);
}

// create a single buffered 600x400 pixels big colour window
int main(int argc, char *argv[])
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (600, 400);
   glutInitWindowPosition (100, 100);
   glutCreateWindow("Exercise 1");
   init ();                    // initialise view
   glutDisplayFunc(display);  // draw scene
   glutMainLoop();
   return 0;                   // ANSI C requires main to return int
}
```

Q3: Draw the 2D object shown on the right. The vertices 0 to 5 of the object and the corresponding co-ordinates are shown in the image on the right. For each answer to the questions below use the most efficient representation. [This question is from the 2004 COMPSCI 372 Test]



(a) Complete the code fragment below so that it defines a global array containing the six 2D vertices of the above shape in the given order:

```
const int numVertices=6;
const float v[numVertices][2] =
```

{ {50, 50}, {150, 100}, {200, 200}, {250, 100},
{350, 50}, {200, 100}};

(b) Complete the display function below so that it draws the given shape using the GL_TRIANGLE_FAN mode and the glVertex2fv command. The vertex numbers and the black dots in the above image have been inserted afterwards for clarity and you don't have to draw them.

```
void display(void)
{
      // clear all pixels in frame buffer
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(1.0, 0.1, 0.2);     // reddish colour
      glBegin(GL_TRIANGLE_FAN);
```

glVertex2fv(v[1]);
glVertex2fv(v[0]);
glVertex2fv(v[4]);
glVertex2fv(v[3]);
glVertex2fv(v[2]);

```
      glEnd();
      glFlush();
}
```

(c) Complete the display function below so that it draws the given shape using the GL_TRIANGLE_STRIP mode and the glVertex2fv command.

```
void display(void)
{
      // clear all pixels in frame buffer
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(1.0, 0.1, 0.2);     // reddish colour
      glBegin(GL_TRIANGLE_STRIP);
      glVertex2fv(v[0]);
      glVertex2fv(v[4]);
      glVertex2fv(v[1]);
      glVertex2fv(v[3]);
      glVertex2fv(v[2]);
      glEnd();
      glFlush();
}
```

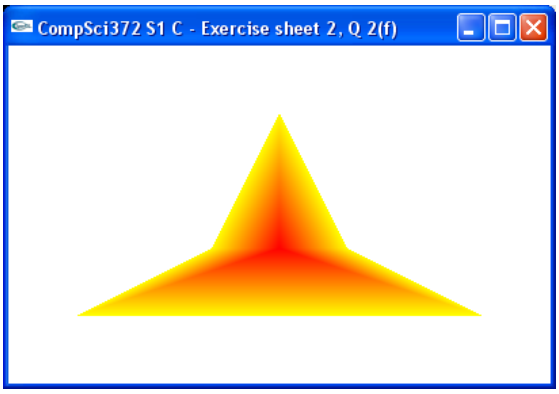
(d) Complete the display function below so that it draws the given shape using the GL_QUAD_STRIP mode and the glVertex2fv command.

```
void display(void)
{
      // clear all pixels in frame buffer
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(1.0, 0.1, 0.2);     // reddish colour
      glBegin(GL_QUAD_STRIP);
      glVertex2fv(v[0]);
      glVertex2fv(v[4]);
      glVertex2fv(v[1]);
      glVertex2fv(v[3]);
      glVertex2fv(v[2]);
      glVertex2fv(v[2]);
      glEnd();
      glFlush();
}
```

(e) Write a program which displays your solutions to (b)-(d).

Solution: use the above code ;-)

(f) Write a program which draws the above shape using an OpenGL primitive such that its colour varies smoothly from red at its centre to yellow at its boundary.




```
void display(void)
{
  // clear all pixels in frame buffer
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_TRIANGLE_FAN);
  glColor3f(1.0, 0.0, 0.0);
  glVertex2fv(v[5]);
  glColor3f(1.0, 1.0, 0.0);
  glVertex2fv(v[1]);
  glVertex2fv(v[0]);
  glVertex2fv(v[4]);
  glVertex2fv(v[3]);
  glVertex2fv(v[2]);
  glVertex2fv(v[1]);
  glEnd();

  // start processing buffered OpenGL routines
  glFlush ();
}
```