THE UNIVERSITY OF AUCKLAND

FIRST SEMESTER, 2001
Campus: City

COMPUTER SCIENCE

Graphics and Graphical User Interface Programming

(Time allowed : TWO hours)

Answer **ALL** questions in the space provided. Extra space for answers is available at the end of this combined question and answer book, just before the Appendix. Additional paper is available on request.

Calculators are not permitted, so you are not expected to numerically evaluate non-trivial arithmetic answers, such as those involving square roots.

The Appendix at the end of this exam lists various classes and methods used in the graphics section of the paper that might be relevant to part B of the examination.

Indicated marks are out of a total of 100 marks.

Surname:	
Forenames:	
ID Number:	
Signature:	

Part	*Possible Marks*	*Awarded Marks*
A	35	
B	65	
TOTAL	100	

CONTINUED

Part B. Graphics. [Part B is worth a total of 65 marks]

B1. Complete each of the following statements by filling in the underlined blank spaces. Each *labelled part* (i.e. a, b, c etc) is worth 1 mark.

[13 marks]

(a) An OpenGL implementation is usually comprised of *three* separate components: GL, ________________ and ________________.

(b) If **v** = (1,5,0) and **u** = (3,2,1) then the dot product **v.u** = ______________________.

(c) The cross product (1,0,0) × (0,-1,0) = ___________________________.

(d) The normal to a planar convex polygon can usually be calculated from the cross product of any two adjacent edge vectors, for example $\overrightarrow{AB} \times \overrightarrow{BC}$, where A, B and C are three consecutive vertices. However, this calculation will fail to yield a useful result if the edges are ___________________________ or if either of them ____.

(e) The homogenous coordinate point (3,4,5,6) corresponds to the normal three-space point _____________________________.

(f) The distance of the plane $2x + y - z = 6$ from the origin is ____________________.

(g) The vector (_________ , _________ , ________) is the unit normal to the plane $2x + y - z = 6$.

(h) Given any three points A, B, C in three-space, the area of triangle ABC can be calculated using vector operations as _______________________________.

(i) If **n** is the normal of a face of a polyhedron and **L** is the vector to the light from a point on the face, then the light illuminates the surface only if the condition _______________________________ is true.

(j) An OpenGL program that provides a trackball for zooming in or out would normally use the "zoom factor" provided by the trackball to adjust the ____________________ parameter in the call to the ______________________ method.

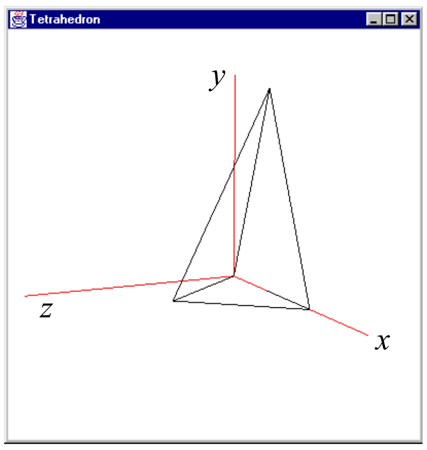
(k) When a polygon is being scan converted with depth buffering enabled, the pixel value *frameBuffer[i][j]* is overwritten with the colour of the current polygon only if the computed depth *d* of the current polygon at the pixel location (i, j) is ____________________________________ *depthBuffer[i][j]*.

(l) Each VRML *shape* node contains a ________________________ field to specify the physical shape and an ________________________ field to specify the "material" of the sphere.

(m) For clipping primitives to the view volume, the _________________________ clipper is used for *lines* and the ___________________________________ clipper is used for filled polygons.

CONTINUED

B2. The incomplete code below and on the oppposite page is meant to draw the wire-frame tetrahedron shown in the figure to the right. The vertices of the tetrahedron are at (0, 0, 0), (6, 0, 0), (3, 0, 4) and (3, 8, 0). The axes and their labels are not drawn by OpenGL but have been added later for clarity.



Fill in the missing code in the answer boxes provided. The code must be written in such a way that *any* polyhedron could be drawn just by changing the values of the *vertices* and *faces* arrays. Also, all four tetrahedron faces must be included in the *faces* array, and their definitions must follow the usual "right hand" rule.

[10 marks]

```
import com.hermetica.magician.*;

public class Tetrahedron extends java.awt.Frame
                         implements GLEventListener {
    private GLComponent glc = null;
    private GL gl_ = new ErrorGL(new CoreGL());
    private GLU glu_ = new ErrorGLU(new CoreGLU());
```

`private float[][] vertices =`
`private int[][] faces =`

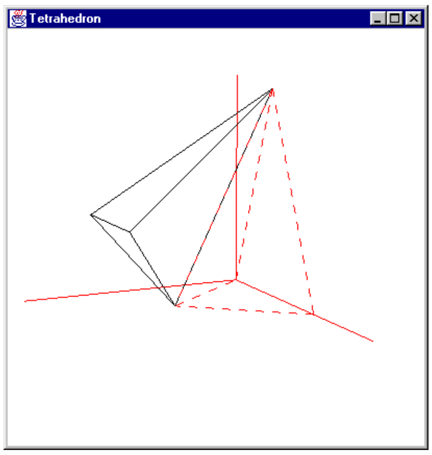
```
    public static void main( String argv[] ) { new Tetrahedron(); }

    public Tetrahedron() {
        super("Tetrahedron");
        glc =
          (GLComponent) GLDrawableFactory.createGLComponent(400,400);
        add( glc );
        pack(); show();
        glc.addGLEventListener( this );
        glc.initialize();
    }

    public void initialize( GLDrawable component ) {
        gl_.glClearColor( 1.0f, 1.0f, 1.0f, 1.0f );
    }

    public void reshape( GLDrawable component,
            int x, int y, int width, int height ) {
        int size = Math.min(width, height);
        gl_.viewport(component,0, 0, size, size);
    }
```

CONTINUED

```
    public void display( GLDrawable component ) {
        gl_.glMatrixMode ( GL.GL_PROJECTION );
        gl_.glLoadIdentity();
        glu_.gluPerspective(35, 1, 5, 25);
        gl_.glMatrixMode( GL.GL_MODELVIEW );
        gl_.glLoadIdentity();
        glu_.gluLookAt(10,6,13,1,1,1,0,1,0);
        gl_.glClear( GL.GL_COLOR_BUFFER_BIT );
        gl_.glColor3f(0,0,0);
        gl_.glPolygonMode(GL.GL_FRONT_AND_BACK, GL.GL_LINE);
        for (int i = 0;
            int[] face = faces[i];
            gl_.glBegin(GL.GL_POLYGON);




             gl_.glEnd();
        }
    }

    public GL getGL() { return gl_; }
}
```

CONTINUED

B3. The figure to the right shows the result of rotating the tetrahedron of question B2 around its longest edge, through an angle of 180°. The dotted lines show the position of the original tetrahedron.



(a) Write down an *expression* for the transformation required in terms of primitive transformations T(*x*,*y*,*z*) and R(*a*,*x*,*y*,*z*). T(*x*,*y*,*z*) represents a translation by the vector (*x*,*y*,*z*) and R(*a*,*x*,*y*,*z*) represents a rotation of *a* degrees about the vector from the origin through the point (*x*,*y*,*z*).

[4 marks]

(b) Write down the code that should be added to the program of question B2 to achieve the required rotation. Also, state where in the program the extra code should be placed. You are not expected to draw the dotted original tetrahedron.

[4 marks]

B4. Complete the following method so that it behaves as specified in the comment at the start of the method.

```
public Transformation viewTransformation( Point3f eye,
                                          Point3f lookAt) {
  // Returns the view transformation, given the position of the
  // eye and the "look at point" (or "centre") of the view.
  // You should assume that the "up" vector is vertical.
```

[6 marks]

```
  float[][] m = {  {u[0], u[1], u[2], -r.dot(u)},
                   {v[0], v[1], v[2], -r.dot(v)},
                   (n[0], n[1], n[2], -r.dot(n)},
                   {0,    0,    0,     1} };
  return new Transformation(m);
}
```

B5. (a) Write down the parametric equation for a straight *line segment* between points P_0 and P_1, including the range of values that the parameter can take.

[2 marks]

(b) Find the point at which the straight line passing through points (0,0,1) and (1,1,-1) meets the plane that passes through the point (0,0,-5) and has a normal in the direction (0,1,1).

[8 marks]

CONTINUED

B6. A shiny sphere, approximated by a polygon mesh with only a few hundred polygons, is rendered with OpenGL. The highlight, which should be approximately circular, is found to have a rather irregular shape that varies as the light source is moved around the sphere.

(a) What shading method does OpenGL use, and why does it cause the above problem?

[3 marks]

(b) What alternative shading method would largely avoid the problem?

[1 mark]

(c) *How* does the alternative shading method largely avoid the problem?

[3 mark]

(d) Why do you think OpenGL does not use that alternative shading method?

[1 mark]

CONTINUED

B7. (a) The illumination equation for a surface might be written as

$$I = k_d (I_a + I_p \mathbf{L.N}) + k_s I_s (\mathbf{H.N})^n.$$

Indicate briefly what each of the symbols k_d, I_a, I_p, I_s and **H** denote.

[5 marks]

(b) A non-specular unit sphere is centred on the origin. It is illuminated by a light source in the direction (1,1,1). There is no ambient light. The sphere is viewed orthographically with a view vector (0,0,1), so that point (x, y) on the screen corresponds to point $\left(x, y, \sqrt{1-x^2-y^2}\right)$ on the sphere surface. What condition do x and y have to satisfy for the point (x, y) on the screen to *not* be black?

[5 marks]

CONTINUED

APPENDIX

A summary of some graphics-related classes and methods

Note: the following list of classes and methods is very incomplete. In answering questions, you may use other *existing* methods not listed here if you wish.

```
package Geometry;

public class Point3f  {           // A point in 3-space
    public Point3f (float x, float y, float z) ; // Constructor, given coordinates
    public Point3f(Point3f p1, Point3f p2, float alpha);  // Constructor of convex sum of p1 and p2
    public Vector3f minus(Point3f p);    // this – p
    public Point3f plus(Vector3f v);  // this + v
    public Point3f transformedBy(Transformation m);   // m * this
    public float x();                  // x coordinate
    public float y();                  // y coordinate
    public float z();                  // z coordinate
    public float[] get3fv()            // coordinates as a 3-element array of floats
 }

public class Vector3f {     // A vector in 3-space
    public Vector3f (float dx, float dy, float dz)                    // Constructor
    public Vector3f(Point3f p)// Constructor of a point's position vector
    public Vector3f minus(Vector3f v)   // this – v
    public Vector3f plus(Vector3f v) // this + v
    public Vector3f times(float f) // this * f
    public Vector3f normalised() // a unit vector in the direction of this
    public float dot(Vector3f v)    // this.v
    public Vector3f cross(Vector3f v)    // this x v
    public float dx();                 // x coordinate
    public float dy();                 // y coordinate
    public float dz();                 // z coordinate
    public float[] get3fv()            // coordinates as a 3-element array of floats
}

public class Transformation {  // A linear affine transformation on points in 3 space
    public Transformation(float[][] matrix)  // Constructor, given matrix
    public static Transformation shift(float dx, float dy, float dz)  // a translation by (dx, dy, dz)
    public static Transformation scaling(float sx, float sy, float sz) // a scaling by (sx, sy, sz)
    public static Transformation rotation(float angle, Vector3f axis)
        // a rotation of "angle" degrees around axis "axis" through origin
    public Transformation times(Transformation m)        // this * m
    public Transformation then(Transformation m)         // m * this
    public Transformation inverse()      // the inverse of this
    public Transformation transposed()                // the transpose of this
    public float[] getGLMatrix()        // OpenGL matrix representation of this
}
```

CONTINUED

package com.hermetica.magician

```
public interface GL {            // Standard OpenGL methods
    public void glVertex3f (float x, float y, float z)            // Output vertex (x,y,z) to OpenGL
    public void glVertex3fv(float[] v)      // Output the vertex (v[0], v[1], v[2])
    public void glNormal3f(float x, float y, float z)             // Output normal (x,y,z) to OpenGL
    public void glNormal3f v(float[] n)        // Output the normal (n[0], n[1], n[2])
    public void glMatrixMode(int matrix)  // Make the given matrix the current one to  manipulate
    public void glLoadIdentify()        // Load current matrix with the identity
    public void glClear( int bitMask)      // Clear all buffers specified in bitMask
    public void glColor3f(float r, float g, float b)              // Current colour = (r,g,b)
    public void glColor3fv(float[] c)   // Current colour = (c[0], c[1], c[2], c[3])
    public void glPolygonMode(int face, int mode) // Render specified polygon faces (GL_FRONT,
        // GL_BACK or GL_FRONT_AND_BACK)  in the specified mode (GL_FILL or GL_LINE)
    public void glBegin(int primitiveType)   // Start output of a primitive of "primitiveType" ...
        // ... such as GL_LINES, GL_LINE_STRIP, GL_QUAD_STRIP or GL_POLYGON
    public void glEnd()              // End of current primitive
    public void glScalef(float sx, float sy, float sz)
        // Multiply current matrix on right by a scaling matrix with scale factors (sx, sy, sz)
    public void glTranslatef(float dx, float dy, float dz)
        // Multiply current matrix on right by a translation matrix with shifts (dx, dy, dz)
    public void glRotatef(float angle, float ax, float ay, float az )  // Multiply current matrix ...
        // ... on right by a rotation matrix for a rotation of angle degrees around axis (ax,ay,az)
    public void glRotatefv(float angle, float[] axis)   // Multiply current matrix on right by ...
        // ... a rotation matrix for a rotation of angle degrees around given axis
}

public interface GLU {          // Standard GLU methods from OpenGL
    public void gluLookAt(float eyeX, float eyeY, float eyeZ, float lookAtX, float lookAtY,
        float lookAtZ, float upX, float upY, float upZ)// Multiply current matrix by ...
                    // ... a view matrix with given eye and lookAt points and "up" vector.
    public void gluPerspective(float fieldOfViewAngle, float aspectRatio, float near, float far)
        // Multiply current matrix by a perpective projection matrix with given fieldOfViewAngle
        // ... at apex of frustum, given viewplane aspectRatio and with given distances along
        // ... negative z axis to "near" and "far" clipping planes.
}
```