

Name	
Student Id	
Signature	

The University of New South Wales
November 2007
Final Examination

COMP3421/COMP9415
Computer Graphics

- Time allowed: three hours
- Number of Questions: 4
- Answer **All** Questions
- Total number of marks: 100
- Calculators permitted.
- You may keep this paper.
- Write your answers using ink.

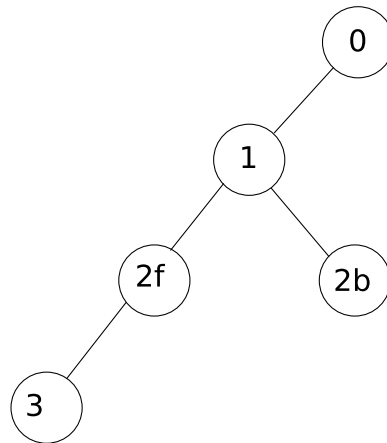
Question 1 (20 Marks)

Give brief (no more than two sentences) definitions of the following

- (a) homogenous coordinates
- (b) Phong exponent
- (c) scene graph
- (d) Snells law
- (e) fragment shader

Question 2 (25 Marks)

- (a) Consider the BSP tree drawn below.



Draw a possible layout of lines 0,1,2, and 3 in 2D space that could create this tree. Don't forget to indicate which is the front side of each line.

If the view position is behind 3 in the BSP tree (i.e. the right child of the node 3), in what order will the objects be drawn?

- (b) Consider a triangle ABC where $A = (0, 0, 0)$, $B = (10, 0, 0)$, $C = (10, 6, 0)$ and a point $X = (5, 3, 0)$. Note that X is the midpoint of AC . The normals at the corners of the triangles are $N_A = (0, 1, 0)$, $N_B = (1, 0, 0)$, $N_C = (0, 0, 1)$.

The triangle has material properties of $k_a = (1, 1, 1)$, $k_d = (0.2, 0.2, 0.2)$, $k_s = (0, 0, 0)$ and $n = 5$.

There is an ambient light of colour $(0.1, 0.1, 0.1)$ and a positional light at $(1, 2, 3)$ with colour $(1, 0.5, 0)$. The light is not attenuated with distance.

The eye position is $(0, 3, 0)$.

Compute the colour at point X using Phong shading.

(c) A point in world space is located at $(1,2,0)$.

We want to first move the point — 5 units on the X axis, 2 units on the Y axis, and 1 unit on the Z axis. Then we want to scale the point by a factor of 2 in the X and Y direction and 3 in the Z direction. Finally we rotate it by 90 degrees around the Z axis.

a. Write out the matrices that perform this series of operations and compute the new point.

Question 3 (25 Marks)

Design a virtual reality simulation for training mine workers. Real coal mines are dangerous places – a mistake can kill you. A virtual reality system lets miners practice and learn in a safe environment. Some of the modules required are:

- rib and roof stability — signs of danger are cracks and bulges in the roof
- unaided self escape — tunnels could be blocked with rubble or filled with smoke and fire
- pre-shift truck inspection — signs of danger are cracks and leaks from hydraulic and fuel tanks and hoses, lights not working, damaged tyres, missing bolts and nuts on wheels and shock absorbers and missing or damaged mud flaps.



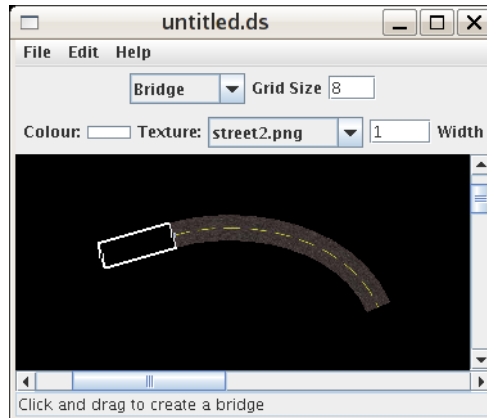
Tell me (using diagrams where appropriate)

- A: what hardware your system would require,
B: what software and algorithms your system would use,
C: how you would store the information needed,
D: and how a user would interact with your system.

Question 4 (30 Marks)

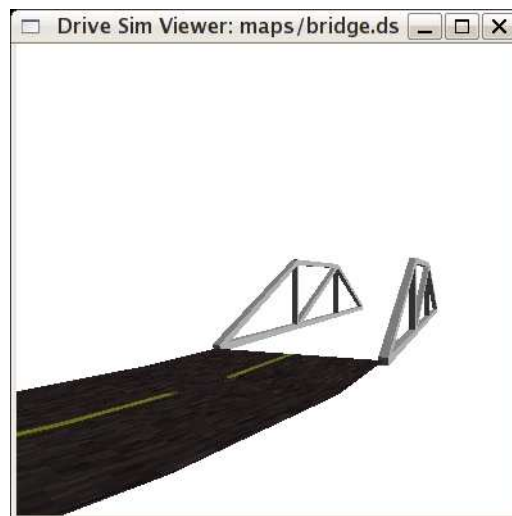
In this question you will modify my solution to the second assignment to add bridges.

A Bridge is defined by two control points giving at the middle of the road at each end of the bridge. In DSEdit it is shown as a rectangle drawn with a line two pixels wide and no texture like this:



The extra field specifies the width of the bridge.

The same map in DSView looks like this:



At the end of the paper you will find a listing of part of my solution. When you want to change one of my files you can indicate it by saying something like “Delete line 63 of DSCanvas.java and replace with the following code.”

If you want to cut and paste code from my solution, don't copy it out again, just write something like “Insert lines 12–16 from Sign.java”.

Each of the parts of this question can be done independently of each other. You can assume that the methods in other parts have been written and work when doing each part.

- Write a BridgeTool that allows interactive creation of a Bridge.
- Write a `paint` method for the Bridge class that draws the representation of the bridge used in DSEdit.

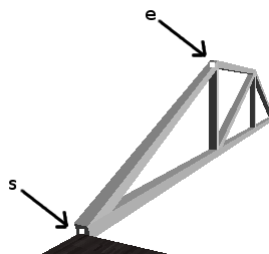
- (c) Write a `contains` method for the Bridge class. It should return true if the user clicks anywhere inside the rectangle.
- (d) Each side of the bridge is made of seven girders, two horizontal, two vertical, and three diagonal.

Write a method

```
public void girder(GL gl, Vector3D s, Vector3D e)
```

that renders one of the girders of the bridge. The girder is a 2 by 2 square in cross-section and the central spine goes from `s` to `e`. It is rotated so that two sides of the square are vertical.

The white dots show the position of `s` and `e` for one of the diagonal girders:.



Hint: Work out vectors \vec{u} , \vec{v} , \vec{n} that define a local coordinate system for the girder with the spine going along the N axis. You'll find the `localToWorld` method in `Vector3D` useful.

- (e) Write the `render3D` method for the Bridge class. The vertical girders are spaced $1/3$ and $2/3$ of the way along the bridge.

Hint: I found this easiest to do by transforming into a convenient local co-ordinate system as I did in Sign.

Building.java

```
1  /** This class represents a building */
2  import java.awt.*;
3  import java.util.*;
4  import javax.media.opengl.*;
5  import com.sun.opengl.util.texture.*;
6
7  public class Building extends Wall {
8
9      public float lineWidth() {
10         return 4.0f;
11     }
12
13     /** paint this building using gl.*/
14     public void paint(GL gl, GLDrawable glc){
15
```

```

16     if (fill != null){
17         setColor(gl,fill);
18         gl.glDisable( GL.GL_TEXTURE_2D );
19     }
20     gl.glLineWidth(lineWidth());
21     gl.glBegin( GL.GL_LINE_LOOP); //draw the building
22
23     Point2D c1 = (Point2D)(pts2d.get(0));
24     Point2D c2 = (Point2D)(pts2d.get(1));
25
26     gl.glVertex2d(c1.x,c1.y);
27     gl.glVertex2d(c1.x,c2.y);
28     gl.glVertex2d(c2.x,c2.y);
29     gl.glVertex2d(c2.x,c1.y);
30     gl.glVertex2d(c1.x,c1.y);
31     gl.glEnd();
32 }
33
34
35 //draw one side of the building
36 void drawSide(GL gl, Point2D start, Point2D end){
37     double wallLength = Math.sqrt((start.x-end.x)*(start.x-end.x) +
38                                     (start.y -end.y)*(start.y-end.y));
39     gl.glBegin(GL.GL_POLYGON);
40     gl.glNormal3d((end.y-start.y)/wallLength, 0, -(end.x-start.x)/wallLength );
41     gl.glTexCoord2d(0, 0);
42     gl.glVertex3d(start.x, 0, start.y);
43     gl.glTexCoord2d(0, 1);
44     gl.glVertex3d(start.x, extra, start.y);
45     gl.glTexCoord2d(scale*wallLength/100, 1);
46     gl.glVertex3d(end.x, extra, end.y);
47     gl.glTexCoord2d(scale*wallLength/100, 0);
48     gl.glVertex3d(end.x, 0, end.y);
49     gl.glEnd();
50 }
51
52
53 public void render3D(GL gl, GLDrawable glc){
54     if (fill != null || texture != null){
55         if (texture == null) {
56             setColor(gl,fill);
57             gl.glDisable( GL.GL_TEXTURE_2D );
58         } else {
59             setColor(gl, Color.white);
60             Texture gltexture = texture.getTexture(glc);
61             gltexture.enable();
62             gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_REPEAT );
63             gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_REPEAT );
64             gl.glTexEnvf( GL.GL_TEXTURE_ENV, GL.GL_TEXTURE_ENV_MODE,
65                           GL.GL_MODULATE );
66             gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER ,
67                                 GL.GL_NEAREST);
68             gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER ,
69                                 GL.GL_NEAREST);
70
71             gltexture.bind();

```

```

72         }
73         gl.glPushMatrix();
74         Point2D c1 = (Point2D)(pts2d.get(0));
75         Point2D c2 = (Point2D)(pts2d.get(1));
76         double left = Math.min(c1.x,c2.x);
77         double right = Math.max(c1.x,c2.x);
78         double top = Math.min(c1.y,c2.y);
79         double bottom = Math.max(c1.y,c2.y);
80         Point2D p1 = new Point2D(left,bottom);
81         Point2D p2 = new Point2D(left,top);
82         Point2D p3 = new Point2D(right,top);
83         Point2D p4 = new Point2D(right,bottom);
84         drawSide(gl,p1,p2);
85         drawSide(gl,p2,p3);
86         drawSide(gl,p3,p4);
87         drawSide(gl,p4,p1);
88         gl.glPopMatrix();
89     }
90 }
91
92
93
94     public boolean contains(int x,int y){
95         Point2D c1 = (Point2D)(pts2d.get(0));
96         Point2D c2 = (Point2D)(pts2d.get(1));
97         double left = Math.min(c1.x,c2.x);
98         double right = Math.max(c1.x,c2.x);
99         double top = Math.min(c1.y,c2.y);
100        double bottom = Math.max(c1.y,c2.y);
101
102        return (x >= left && x <=right && y >= top && y <= bottom);
103    }
104
105 }

```

Point2D.java

```

1    /** This class represents a 2D point. Like Point of java.awt, but
2        uses floats instead of ints.
3        */
4    import java.awt.Point;
5    import java.io.*;
6
7    public class Point2D implements Serializable{
8        double x,y;
9
10       public Point2D(double x, double y) {
11           this.x = x;
12           this.y = y;
13       }
14
15       public Point2D(Point p) {
16           this(p.x,p.y);
17       }
18

```

```

19     public double getX(){
20         return x;
21     }
22
23     public double getY(){
24         return y;
25     }
26
27     //interpolate this + t*(q-this)
28     public Point2D interp(Point2D q,double t){
29         return new Point2D(x + t*(q.x-x), y + t*(q.y-y));
30     }
31
32     //subtract q from this point
33     public Point2D subtract(Point2D q){
34         return new Point2D(x-q.x, y-q.y);
35     }
36
37     //add q from this point
38     public Point2D add(Point2D q){
39         return new Point2D(x+q.x, y+q.y);
40     }
41
42     //multiply by a scalar
43     public Point2D scale(double s){
44         return new Point2D(s*x, s*y);
45     }
46
47     //length of vector
48     public double length(){
49         return Math.sqrt(x*x+y*y);
50     }
51
52     public String toString(){
53         return x+" "+y;
54     }
55
56
57     public Point toPoint(){
58         return new Point((int)Math.round(x),(int)Math.round(y));
59     }
60 }

```

Road.java

```

1     import java.awt.Point;
2     import java.awt.Color;
3     import java.util.*;
4     import javax.media.opengl.*;
5     import com.sun.opengl.util.texture.*;
6
7     public class Road extends DSPolygon {
8         List<Point2D> pol = new ArrayList<Point2D>(); //Polygonal outline of the road
9         double length; //cumulated length along road
10        Point2D lastV; //last vertex on road spine

```

```

11
12  /** name of the extra parameter
13      @returns name of extra parameter (eg "width")
14  */
15
16  public String extraName(){
17      return "Width";
18  }
19
20
21  // the basis function for a Bezier spline
22  static float b(int i, float t) {
23      switch (i) {
24          case 0:
25              return (1-t)*(1-t)*(1-t);
26          case 1:
27              return 3*t*(1-t)*(1-t);
28          case 2:
29              return 3*t*t*(1-t);
30          case 3:
31              return t*t*t;
32      }
33      return 0; //we only get here if an invalid i is specified
34  }
35
36  // derivative of the basis function for a Bezier spline
37  static float bdash(int i, float t) {
38      switch (i) {
39          case 0:
40              return (-3*t + 6)*t - 3;
41          case 1:
42              return (9*t - 12)*t + 3;
43          case 2:
44              return (-9*t +6)*t;
45          case 3:
46              return 3*t*t;
47      }
48      return 0; //we only get here if an invalid i is specified
49  }
50
51  //evaluate a point on the Bezier
52  Point2D p(int i, float t) {
53      float px=0;
54      float py=0;
55      for (int j = 0; j<=3; j++){
56          Point2D p = (Point2D)(pts2d.get(i+j));
57          px += b(j,t)*p.x;
58          py += b(j,t)*p.y;
59      }
60      return new Point2D(px,py);
61  }
62
63  //evaluate a derivative on the Bezier
64  Point2D pdash(int i, float t) {
65      float px=0;
66      float py=0;

```

```

67         for (int j = 0; j<=3; j++){
68             Point2D p = (Point2D)(pts2d.get(i+j));
69             px += bdash(j,t)*p.x;
70             py += bdash(j,t)*p.y;
71         }
72         return new Point2D(px,py);
73     }
74
75     final int STEPS = 12;
76
77     private void addVertex(GL gl, int i, float j){
78         Point2D q = p(i,j);
79         Point2D qdash = pdash(i,j);
80         double len = qdash.length();
81         if (lastV != null){
82             length += q.subtract(lastV).length();
83         }
84
85         if (len == 0) {
86             len = 1;
87         }
88         qdash.x = qdash.x*(extra/2)/len;
89         qdash.y = qdash.y*(extra/2)/len;
90         Point2D p1 = new Point2D(q.x-qdash.y, q.y+qdash.x);
91         Point2D p2 = new Point2D(q.x+qdash.y, q.y-qdash.x);
92         pol.add(p1);
93         pol.add(0,p2);
94
95
96         gl.glTexCoord2d(length*scale/100, 0);
97         gl.glVertex2d(p1.x, p1.y);
98         gl.glTexCoord2d(length*scale/100, 1);
99         gl.glVertex2d(p2.x, p2.y);
100
101         lastV = q;
102     }
103
104
105     // 3D Version of add vertex. Does not have to handle any of the interactive
106     // aspects of the problem.
107     private void addVertex3D(GL gl, int i, float j){
108         Point2D q = p(i,j);
109         Point2D qdash = pdash(i,j);
110         double len = Math.sqrt(qdash.x*qdash.x + qdash.y*qdash.y);
111         if (lastV != null){
112             double dx = q.x - lastV.x;
113             double dy = q.y - lastV.y;
114             length += Math.sqrt(dx*dx + dy*dy);
115         }
116
117         if (len == 0) {
118             len = 1;
119         }
120         qdash.x = qdash.x*(extra/2)/len;
121         qdash.y = qdash.y*(extra/2)/len;
122         Point2D p1 = new Point2D(q.x-qdash.y, q.y+qdash.x);

```

```

123         Point2D p2 = new Point2D(q.x+qdash.y, q.y-qdash.x);
124         gl.glTexCoord2d(length*scale/100, 0);
125         gl.glVertex3d(p1.x, 1, p1.y);
126         gl.glTexCoord2d(length*scale/100, 1);
127         gl.glVertex3d(p2.x, 1, p2.y);
128         lastV = q;
129     }
130
131
132     /** paint this curve into g.*/
133     public void paint(GL gl, GLDrawable glc){
134
135         if (fill != null || texture != null){
136             if (texture == null) {
137                 setColor(gl,fill);
138                 gl.glDisable( GL.GL_TEXTURE_2D );
139             } else {
140                 Texture gltexture = texture.getTexture(glc);
141                 gltexture.enable();
142                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_REPEAT );
143                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_REPEAT );
144                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER, GL.GL_LINEAR );
145                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER, GL.GL_LINEAR );
146                 gl.glTexEnvf( GL.GL_TEXTURE_ENV, GL.GL_TEXTURE_ENV_MODE,
147                             GL.GL_REPLACE );
148                 gltexture.bind();
149             }
150             if(pts2d.size() >= 4){
151                 length = 0.0;
152                 lastV = null;
153                 pol.clear();
154                 gl.glBegin( GL.GL_TRIANGLE_STRIP ); //draw the Bezier
155                 addVertex(gl,0,0);
156                 for (int i = 0; i < pts2d.size()-3; i+=3) {
157                     for (int j = 1; j <= STEPS; j++) {
158                         addVertex(gl,i,j/(float)STEPS);
159                     }
160                 }
161                 gl.glEnd();
162             }
163
164         }
165     }
166 }
167
168
169     public void render3D(GL gl, GLDrawable glc){
170
171         if (fill != null || texture != null){
172             if (texture == null) {
173                 setColor(gl,fill);
174                 gl.glDisable( GL.GL_TEXTURE_2D );
175             } else {
176                 setColor(gl,Color.white);
177                 Texture gltexture = texture.getTexture(glc);
178                 gltexture.enable();

```

```

179         gl.glTexEnvf( GL.GL_TEXTURE_ENV, GL.GL_TEXTURE_ENV_MODE,
180                       GL.GL_MODULATE);
181         gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_REPEAT );
182         gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_REPEAT );
183         gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER,
184                             GL.GL_NEAREST);
185         gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER,
186                             GL.GL_NEAREST);
187         gltexture.bind();
188     }
189     if(pts2d.size() >= 4){
190         length = 0.0;
191         lastV = null;
192         gl.glBegin( GL.GL_TRIANGLE_STRIP ); //draw the Bezier
193         gl.glNormal3d(0,-1,0);
194         addVertex3D(gl,0,0);
195         for (int i = 0; i < pts2d.size()-3; i+=3) {
196             for (int j = 1; j <= STEPS; j++) {
197                 addVertex3D(gl,i,j/(float)STEPS);
198             }
199         }
200         gl.glEnd();
201     }
202
203
204 }
205 }
206
207
208 public boolean contains(int x,int y){
209     return toPolygon(pol).contains(x,y);
210 }
211
212
213 }

```

Sign.java

```

1  /** This class represents a sign */
2  import java.awt.*;
3  import java.util.*;
4  import javax.media.opengl.*;
5  import com.sun.opengl.util.*;
6  import com.sun.opengl.util.texture.*;
7
8  public class Sign extends Wall {
9
10     double signBot = 6;
11
12     public float lineWidth() {
13         return 2.0f;
14     }
15
16     public void render3D(GL gl, GLDrawable glc){
17         if (fill != null || texture != null){

```

```

18     if (texture == null) {
19         setColor(gl,fill);
20         gl.glDisable( GL.GL_TEXTURE_2D );
21     } else {
22         setColor(gl, Color.white);
23         Texture gltexture = texture.getTexture(glc);
24         gltexture.enable();
25         gl.glTexEnvf( GL.GL_TEXTURE_ENV, GL.GL_TEXTURE_ENV_MODE,
26                     GL.GL_MODULATE );
27         gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER ,
28                             GL.GL_NEAREST);
29         gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER ,
30                             GL.GL_NEAREST);
31
32         gltexture.bind();
33     }
34     gl.glPushMatrix();
35     Point2D start = pts2d.get(0);
36     Point2D end = pts2d.get(1);
37     double signLength = Math.sqrt((start.x-end.x)*(start.x-end.x) +
38                                   (start.y-end.y)*(start.y-end.y));
39
40     /* Transform to local coordinate system where bottom of the sign is
41        origin and sign goes along z axis */
42     Vector3D start3D = new Vector3D(start.x,0,start.y);
43     Vector3D end3D = new Vector3D(end.x,0,end.y);
44     Vector3D centre3D = start3D.interp(end3D,0.5);
45     Vector3D n = end3D.subtract(start3D).normalize();
46     Vector3D v = new Vector3D(0,1,0);
47     Vector3D u = v.cross(n).normalize();
48
49     Vector3D.localToWorld(gl,u,v,n,centre3D);
50
51
52     gl.glBegin(GL.GL_POLYGON);
53     gl.glNormal3d(1, 0, 0);
54     gl.glTexCoord2d(0, 0);
55     gl.glVertex3d(0, signBot, -signLength/2);
56     gl.glTexCoord2d(0, scale);
57     gl.glVertex3d(0, extra, -signLength/2);
58     gl.glTexCoord2d(scale, scale);
59     gl.glVertex3d(0, extra, signLength/2);
60     gl.glTexCoord2d(scale, 0);
61     gl.glVertex3d(0, signBot, signLength/2);
62     gl.glEnd();
63
64     setColor(gl,fill);
65     gl.glDisable( GL.GL_TEXTURE_2D );
66     gl.glTranslated(0, signBot/2, 0);
67     gl.glScaled(0.5,signBot,0.5);
68     GLUT glut = new GLUT();
69     glut.glutSolidCube(1);
70
71     gl.glPopMatrix();
72 }
73 }

```

```
74
75
76
77 }
```

SignTool.java

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.util.*;
4
5  /** Polygon tool for graphical editor
6   * @author Tim Lambert
7   */
8
9  public class SignTool extends WallTool{
10
11
12     public SignTool(DSEdit editor) {
13         super(editor);
14     }
15
16     public String getName() {
17         return "Sign";
18     }
19
20     public String getMessage() {
21         return "Click and drag to create a road sign";
22     }
23
24
25     public Wall newWall(){
26         return new Sign();
27     }
28
29 }
```

Viewpoint.java

Vector3D.java

```
1  /** This class represents a 3D vector.
2   */
3  import javax.media.opengl.*;
4
5  public class Vector3D {
6     double x,y,z;
7
8     public Vector3D() {
9         x = 0;
10        y = 0;
```

```

11     z = 0;
12 }
13
14 public Vector3D(double x, double y, double z) {
15     this.x = x;
16     this.y = y;
17     this.z = z;
18 }
19
20 public Vector3D(Vector3D p) {
21     x = p.x;
22     y = p.y;
23     z = p.z;
24 }
25
26 public Vector3D(float a[]) {
27     if (a.length >= 3){
28         x = a[0];
29         y = a[1];
30         z = a[2];
31     }
32 }
33
34 public Vector3D(double a[]) {
35     if (a.length >= 3){
36         x = a[0];
37         y = a[1];
38         z = a[2];
39     }
40 }
41
42 public double getX(){
43     return x;
44 }
45
46 public double getY(){
47     return y;
48 }
49
50 public double getZ(){
51     return z;
52 }
53
54 //interpolate this + t*(q-this)
55 public Vector3D interp(Vector3D q,double t){
56     return new Vector3D(x + t*(q.x-x), y + t*(q.y-y), z + t*(q.z-z));
57 }
58
59 //subtract q from this point
60 public Vector3D subtract(Vector3D q){
61     return new Vector3D(x-q.x, y-q.y, z-q.z);
62 }
63
64 //add q from this point
65 public Vector3D add(Vector3D q){
66     return new Vector3D(x+q.x, y+q.y, z+q.z);

```

```

67     }
68
69     //multiply by a scalar
70     public Vector3D scale(double s){
71         return new Vector3D(s*x, s*y, s*z);
72     }
73
74     //length of vector
75     public double length(){
76         return Math.sqrt(x*x+y*y+z*z);
77     }
78
79     //make it length one
80     public Vector3D normalize(){
81         return scale(1/length());
82     }
83
84     //dot product
85     public double dot(Vector3D q){
86         return x*q.x + y*q.y + z*q.z;
87     }
88
89     //cross product
90     public Vector3D cross(Vector3D q){
91         return new Vector3D(y*q.z - z*q.y,
92                             z*q.x - x*q.z,
93                             x*q.y - y*q.x);
94     }
95
96     public String toString(){
97         return x+" "+y+" "+z;
98     }
99
100    //this does exactly the same thing as gluLookat, except that it takes
101    //three vectors
102    public static void lookAt(GL gl, Vector3D eye, Vector3D centre, Vector3D up) {
103        Vector3D n = eye.subtract(centre).normalize();
104        Vector3D u = up.cross(n).normalize();
105        Vector3D v = n.cross(u);
106        /* OpenGL matrices are in column major order */
107        double[] m = {u.x, v.x, n.x, 0,
108                     u.y, v.y, n.y, 0,
109                     u.z, v.z, n.z, 0,
110                     0, 0, 0, 1};
111
112        /* do the rotation*/
113        gl.glMultMatrixd(m,0);
114
115        /* Translate Eye to Origin */
116        gl.glTranslated(-eye.x, -eye.y, -eye.z);
117    }
118
119    //Apply local to world transform given basis and origin
120
121    public static void localToWorld(GL gl, Vector3D u, Vector3D v, Vector3D n, Vector3D O) {
122        /* OpenGL matrices are in column major order */

```

```

123         double[] m = {u.x, u.y, u.z, 0,
124                        v.x, v.y, v.z, 0,
125                        n.x, n.y, n.z, 0,
126                        0.x, 0.y, 0.z, 1};
127
128         /* apply the transform*/
129         gl.glMultMatrixd(m,0);
130     }
131
132 }

```

Wall.java

```

1  /** This class represents a wall */
2  import java.awt.*;
3  import java.util.*;
4  import javax.media.opengl.*;
5
6  public class Wall extends DSPolygon {
7
8      public float lineWidth() {
9          return 4.0f;
10     }
11
12     /** paint the wall using gl.*/
13     public void paint(GL gl, GLDrawable glc){
14
15         if (fill != null){
16             setColor(gl,fill);
17             gl.glDisable( GL.GL_TEXTURE_2D );
18         }
19         gl.glLineWidth(lineWidth());
20         gl.glBegin( GL.GL_LINES ); //draw the wall
21         for(int i = 0; i < 2; i++) {
22             Point2D p = (Point2D)(pts2d.get(i));
23             gl.glVertex2d(p.x, p.y);
24         }
25         gl.glEnd();
26     }
27
28
29     /** add a control point */
30     public void addPoint(Point p) {
31         if (pts2d.size() < 2) {
32             super.addPoint(p);
33         }
34     }
35
36
37     /** remove specified control point */
38     public void removePoint() {
39         return;
40     }
41
42     public boolean contains(int x,int y){

```

```

43         Edge2D e = new Edge2D((Point2D)pts2d.get(0),
44                               (Point2D)pts2d.get(1));
45         Point2D pe = e.toLineSpace(new Point(x,y));
46         return (pe.getX() > 0 && pe.getX() < 1 && Math.abs(pe.getY()) < EPSILON);
47     }
48
49 }

```

WallTool.java

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import java.util.*;
4
5  /** Polygon tool for graphical editor
6   @author Tim Lambert
7   */
8
9  public class WallTool extends Tool{
10
11
12     public WallTool(DSEdit editor) {
13         super(editor);
14         extra = 2.4; //default height
15     }
16
17     public String getName() {
18         return "Wall";
19     }
20
21     public String getMessage() {
22         return "Click and drag to create a wall";
23     }
24
25     public String extraName(){
26         return "Height";
27     }
28
29     protected Wall p; /* Wall being created */
30
31
32     public Wall newWall(){
33         return new Wall();
34     }
35
36     // create the polygon
37     public void mousePressed(MouseEvent e) {
38         p = newWall();
39         editor.getProperties(p);
40
41         canvas.getShapes().add(p);
42         canvas.select(p);
43         p.addPoint(gridPoint(e));
44         p.addPoint(gridPoint(e));
45         canvas.repaint();

```

```
46     }
47
48
49     public void mouseDragged(MouseEvent e) {
50         if (p != null){
51             p.setPoint(gridPoint(e));
52             canvas.repaint();
53         }
54     }
55
56     public void mouseReleased(MouseEvent e) {
57         mouseDragged(e);
58         p = null;
59     }
60
61 }
```

Local illumination equation

$$I = I_a k_a + \sum_{k=1 \dots n} f_{\text{att}}(d_k) I_k (k_d \vec{N} \cdot \vec{L}_k + k_s (\vec{V} \cdot \vec{R}_k)^n)$$

where

- k_a = ambient reflection coefficient
- k_d = diffuse reflection coefficient
- k_s = specular reflection coefficient
- \vec{N} = surface normal
- I_a = Ambient light intensity
- I_k = Intensity of light source k
- \vec{L}_k = Direction to light source k
- \vec{V} = Direction of viewer
- $\vec{R}_k = 2\vec{N}(\vec{N} \cdot \vec{L}_k) - \vec{L}_k$
- n = Phong exponent
- d_k = distance to light source k
- $f_{\text{att}}()$ = light attenuation function

All the vectors (N, L_k, V) should be normalized.

Bezier curve

$$B(t) = \sum_{i=0 \dots 3} b_i(t) p_i$$

where

- $b_0(t) = (1-t)^3$
- $b_1(t) = 3t(1-t)^2$
- $b_2(t) = 3t^2(1-t)$
- $b_3(t) = t^3$
- p_i = Control point i

B spline curve

$$B_j(t) = \sum_{i=0 \dots 3} b_i(t) p_{i+j}$$

where

- $b_0(t) = (-t^3 + 3t^2 - 3t + 1)/6$
- $b_1(t) = (3t^3 - 6t^2 + 4)/6$
- $b_2(t) = (-3t^3 + 3t^2 + 3t + 1)/6$
- $b_3(t) = t^3/6$
- p_i = Control point i

2D Transformations

Translation (t_x, t_y)	Scaling (s_x, s_y)	Rotation by θ
$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Window-Viewport

$$\begin{bmatrix} a & 0 & c \\ 0 & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

where

$$\begin{aligned} a &= (V_r - V_l)/(W_r - W_l) \\ c &= V_l - aW_l \\ d &= (V_t - V_b)/(W_t - W_b) \\ f &= V_b - dW_b \end{aligned}$$

Vector operations

$$\begin{aligned} \hat{a} &= (a_x, a_y, a_z) \\ \hat{a} \cdot \hat{b} &= a_x b_x + a_y b_y + a_z b_z \\ \|\hat{a}\| &= \sqrt{a_x^2 + a_y^2 + a_z^2} \\ \hat{a} \times \hat{b} &= (a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x) \end{aligned}$$

3D Transformations

$$\begin{array}{ccc} T(t_x, t_y, t_z) & S(s_x, s_y, s_z) & R_X(\theta) \\ \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ R_Y(\theta) & R_Z(\theta) & \text{perspective} \\ \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \end{array}$$

Local to World Transformation

$$M_{UVN \rightarrow XYZ} = \begin{bmatrix} u_x & v_x & n_x & O_x \\ u_y & v_y & n_y & O_y \\ u_z & v_z & n_z & O_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing Transformation

$$\begin{aligned} \vec{n} &= V\vec{P}N / \|V\vec{P}N\| \\ \vec{u} &= V\vec{u}p \times V\vec{P}N / \|V\vec{u}p \times V\vec{P}N\| \\ \vec{v} &= \vec{n} \times \vec{u} \\ M_{XYZ \rightarrow UVN} &= \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} T(-o_x, -o_y, -o_z) \end{aligned}$$

Linear Interpolation

Parametric equation of line from \vec{a} to \vec{b} is

$$L(t) = \vec{a} + (\vec{b} - \vec{a})t, \quad 0 \leq t \leq 1$$

This is also the formula for linear interpolation.