

Name	
Student Id	
Signature	

**The University of New South Wales**  
**November 2008**  
**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) flat shading
- (b) specular reflection
- (c)  $G^1$  continuity
- (d) Z buffer
- (e) HSV

### Question 2 (25 Marks)

- (a) While a Bezier curve cannot make an exact circle, it can make a close approximation. Find the four control points for a Bezier that approximates the quarter of the unit circle that goes from  $(1, 0)$  to  $(0, 1)$ . The tangents at the endpoints of the Bezier  $((1, 0)$  and  $(0, 1))$  should be the same as for the circle. The midpoint of the Bezier curve should be the point halfway along the quarter circle (i.e.  $(\sqrt{2}/2, \sqrt{2}/2)$ ). And the Bezier curve should be symmetric about the line  $x = y$ .
- (b) The view direction is  $(3, 4, 12)$  and the up vector is  $(0, 1, 0)$ . Calculate the orthonormal basis  $(u, v$  and  $n)$  for camera space. If the view reference point is  $(0, 0, 10)$  what is the matrix that transforms from world coordinates to camera coordinates?

### Question 3 (25 Marks)

A company that sells flat pack furniture (furniture that the customer must assemble) wants a computer system so that it can show customers how to assemble the furniture using 3D graphics. For example, here is a picture of one of their desks.



This desk comes as a lot of individual pieces of wood, together with the screws and bolts need to hold it together.

They want to be able to show the customers what the desk will look like at each stage of assembly and show animations of what is required at step of the procedure. Customers need to be able to view the desk from different directions, and control speed and direction of the animations.

Design such a system for them.

Tell me (using diagrams where appropriate)

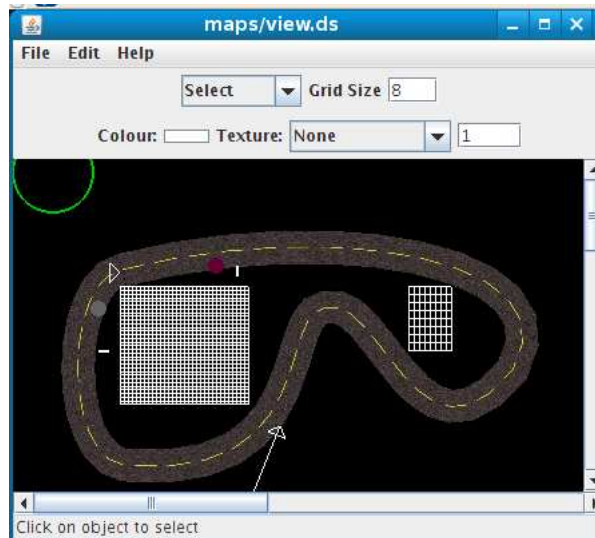
- A: what hardware your system would require,
- B: what software and algorithms your system would use,
- C: how you would store the information needed,
- D: and how a user would interact with your system.

### Question 4 (30 Marks)

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

A Terrain is defined by two control points giving two diagonally opposite corners of a rectangular area to be covered by fractal terrain. Just like a Building, the sides of the rectangle are parallel to the coordinate axes. The extra field is the level of the fractal – if the level is 16, the rectangle is divided up into a 16x16 grid of smaller rectangles.

In DSEdit it is shown by drawing the grid the grid of rectangles like this:



This is the sample map from the assignments with the Buildings replaced by Terrains. The bigger Terrain is level 32, while the smaller one is level 8.

The same map in DSView looks like this:



At the end of the paper you will find a listing of part of my solution. I've also included the code for FractalTerrain from the ToyFinal example. 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.

- (a) Write a `TerrainTool` that allows interactive creation of a `Terrain`.
- (b) Write a `paint` method for the `Terrain` class that draws the representation of the terrain used in `DSEdit`.
- (c) Write a `render3D` method for the `Terrain` class that draws the representation of the terrain used in `DSView`. You can ignore textures.
- (d) We'd like roads to go up and down over a `Terrain`, rather than straight through it. To make this work, `Terrain` needs a method that returns the height of an arbitrary point on the terrain:

```
public double height(double x, double z);
```

Write this method. If the query point is outside the rectangle that defines the `Terrain`, `height` should return 0.

- (e) Modify `Road` so that `Roads` go over a `Terrain`. To make things simple, the `Terrain` that it must follow is available as `DSView.terrain`.
- (f) Describe you you would make the car go over a `Terrain`. You don't have to write any code, just tell me how you would do it.

## 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) + (start.y -end.y)*(start.y
38     gl.glBegin(GL.GL_POLYGON);
39     gl.glNormal3d((end.y-start.y)/wallLength, 0, -(end.x-start.x)/wallLength );
40     gl.glTexCoord2d(scale*wallLength/100, 1);
41     gl.glVertex3d(start.x, 0, start.y);
42     gl.glTexCoord2d(scale*wallLength/100, 0);
43     gl.glVertex3d(start.x, extra, start.y);
44     gl.glTexCoord2d(0, 0);
45     gl.glVertex3d(end.x, extra, end.y);
46     gl.glTexCoord2d(0, 1);
47     gl.glVertex3d(end.x, 0, end.y);
48     gl.glEnd();
49 }
50
51
52 public void render3D(GL gl, GLDrawable glc){
53     if (fill != null || texture != null){
54         if (texture == null) {
55             setColor(gl,fill);
56             gl.glDisable( GL.GL_TEXTURE_2D );
57         } else {
58             setColor(gl, Color.white);
59             Texture gltexture = texture.getTexture(glc);
60             gltexture.enable();
61             gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_REPEAT );
62             gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_REPEAT );
63             gl.glTexEnvf( GL.GL_TEXTURE_ENV, GL.GL_TEXTURE_ENV_MODE,
64                 GL.GL_MODULATE );
65             gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER ,
66                 GL.GL_NEAREST);
67             gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER ,
68                 GL.GL_NEAREST);
69
70             gltexture.bind();
71         }
72         gl.glPushMatrix();
73         Point2D c1 = (Point2D)(pts2d.get(0));
74         Point2D c2 = (Point2D)(pts2d.get(1));
75         double left = Math.min(c1.x,c2.x);
76         double right = Math.max(c1.x,c2.x);
77         double top = Math.min(c1.y,c2.y);
78         double bottom = Math.max(c1.y,c2.y);

```

```

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

```

## BuildingTool.java

```

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

```

```

27     }
28
29 }

```

## FractalTerrain.java

```

1  /** Draw some fractal Terrain
2  */
3
4  import javax.media.opengl.*;
5
6  public class FractalTerrain{
7
8      static double[][] y;
9      static int oldlevel=-1; //level last time we were called
10
11     /*      Return a random double between -1.0 and 1.0*/
12     static double myrand(){
13         return(Math.random()*2-1);
14     }
15
16
17     /* Determine the mid point values of the given square,
18     then sub-divide into 4 smaller squares and do the
19     same for each of them (till no midpoints left)
20     square abdc is divided into four smaller squares as
21     below
22     a-f-c
23     | | |
24     e-g-i
25     | | |
26     b-h-d
27     We only want to generate height values at each point once, since the
28     surface will crack otherwise. So heights at e and f should be generated
29     only if we are on boundary.
30     */
31     static void generate(int minx, int minz, int maxx, int maxz, double jag)
32     {
33         int     midx, midz;
34
35         if (maxx <= (minx + 1)) /* Since it is a square this covers both x & z */
36             return;          /* they would both fail here at the same time. */
37         /* i.e. recursion ends when no points between. */
38
39         midx = (minx + maxx) / 2;
40         midz = (minz + maxz) / 2;
41
42         /* generate each of the mid points (top, left only if on boundary) */
43         if (minx == 0)
44             y[minx][midz] = (y[minx][minz] + y[minx][maxz]) / 2.0 + jag * myrand(); //e
45         if (minz == 0)
46             y[midx][minz] = (y[minx][minz] + y[maxx][minz]) / 2.0 + jag * myrand(); //f
47
48         y[midx][midz] = (y[minx][minz] + y[minx][maxz] + y[maxx][minz] + y[maxx][maxz]) / 4.0 + jag * myrand();
49         y[midx][maxz] = (y[minx][maxz] + y[maxx][maxz]) / 2.0 + jag * myrand(); //h

```

```

50     y[maxx][midz] = (y[maxx][minz] + y[maxx][maxz]) / 2.0 + jag * myrand(); //i
51
52     /* Preserve self similarity for sub squares */
53     jag /= 2.0;
54
55     /* generate each new sub-square */
56     generate(minx, minz, midx, midz, jag); /* a-e-f-g */
57     generate(minx, midz, midx, maxz, jag); /* e-b-g-h */
58     generate(midx, minz, maxx, midz, jag); /* f-g-c-i */
59     generate(midx, midz, maxx, maxz, jag); /* g-h-i-d */
60 }
61
62
63 //normal to a triangle
64 public static Vector3D normal (Vector3D a, Vector3D b, Vector3D c){
65     return b.subtract(a).cross(c.subtract(a));
66 }
67
68
69 public static void surface(GL gl, int level, double jag){
70     Vector3D a,b,c,n;
71
72     if (level != oldlevel){
73         //generate new array of y values
74         y = new double[level+1] [];
75         for (int i = 0; i <= level; i++){
76             y[i] = new double[level+1];
77         }
78         generate(0,0,level,level,jag);
79         oldlevel = level;
80     }
81
82     float h = 1.0f/level;
83     float hh = h/10.0f;
84     for (int i = 0; i < level-1; i++){
85         float x = i*h;
86         gl.glBegin(GL.GL_TRIANGLE_STRIP);
87         for (int j = 0; j < level; j++){
88             float z = j*h;
89             a = new Vector3D(x+h, y[i+1][j], z);
90             b = new Vector3D(x+h+h, y[i+2][j], z);
91             c = new Vector3D(x+h, y[i+1][j+1], z+h);
92             n = normal(a,c,b);
93             gl.glTexCoord2d(x+h,z);
94             gl.glNormal3d(n.x,n.y,n.z);
95             gl.glVertex3d(a.x,a.y,a.z);
96
97             a = new Vector3D(x, y[i][j], z);
98             b = new Vector3D(x+h, y[i+1][j], z);
99             c = new Vector3D(x, y[i][j+1], z+h);
100            n = normal(a,c,b);
101            gl.glTexCoord2d(x,z);
102            gl.glNormal3d(n.x,n.y,n.z);
103            gl.glVertex3d(a.x,a.y,a.z);
104
105        }

```

```

106         gl.glEnd();
107     }
108 }
109
110 }

```

## MyCar.java

```

1  import java.awt.*;
2  import java.util.*;
3  import javax.media.opengl.*;
4  import com.sun.opengl.util.*;
5
6  public class MyCar {
7      final float increment = 0.002f;
8      float currentSpeed = 0.02f;
9      final float maxSpeed = 0.1f;
10     int i = 0;
11     Point2D cPos;
12     Point2D cDir;
13
14     float t = 0;
15     public MyCar(Point2D pos, Point2D dir){
16         cPos = pos;
17         cDir = dir;
18     }
19
20
21     public void decreaseSpeed(){
22         currentSpeed -= increment;
23         if(currentSpeed < 0) currentSpeed = 0;
24     }
25
26     public void increaseSpeed(){
27         currentSpeed += increment;
28         if(currentSpeed > maxSpeed) currentSpeed = maxSpeed;
29     }
30
31     public void stop(){
32         currentSpeed = 0;
33     }
34
35     public void turn(double angle){
36         double ang = angle + Math.atan2(cDir.y, cDir.x);
37         cDir.x = Math.cos(ang);
38         cDir.y = Math.sin(ang);
39     }
40
41     public void fullSpeed(){
42         currentSpeed = maxSpeed;
43     }
44
45     public void advancePos(){
46         cPos.x += currentSpeed*cDir.x;
47         cPos.y += currentSpeed*cDir.y;

```

```

48     }
49
50     public Point2D currentPos(){
51         return cPos;
52     }
53
54     public Point2D currentDir(){
55         return cDir;
56     }
57
58
59     public void setColor(GL gl, Color c){
60         gl.glColor3ub((byte)c.getRed(),(byte)c.getGreen(),(byte)c.getBlue());
61     }
62
63     public void render3D(GL gl, GLDrawable glc){
64
65         setColor(gl,Color.white);
66         gl.glDisable( GL.GL_TEXTURE_2D );
67         // System.out.println("cDir = " + cDir.toString());
68         double angle = Math.atan2(cDir.y, cDir.x);
69         gl.glPushMatrix();
70         gl.glTranslated(cPos.x, 4, cPos.y);
71         gl.glRotated(-angle*180.0/Math.PI, 0, 1, 0);
72         gl.glScaled(2,0.6,1);
73         GLUT glut = new GLUT();
74         glut.glutSolidCube(10);
75         gl.glPopMatrix();
76     }
77 }

```

## 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.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_REPEAT );
182             gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_REPEAT );
183             gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER,
184                                 GL.GL_NEAREST);
185             gl.glTexParameteri( 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 }

```

## 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                  O.x, O.y, O.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 }
```

## 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 $\theta$
$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Window-Viewport

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

where

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

## 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.