

Name	
Student Id	
Signature	

The University of New South Wales
November 2010
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.

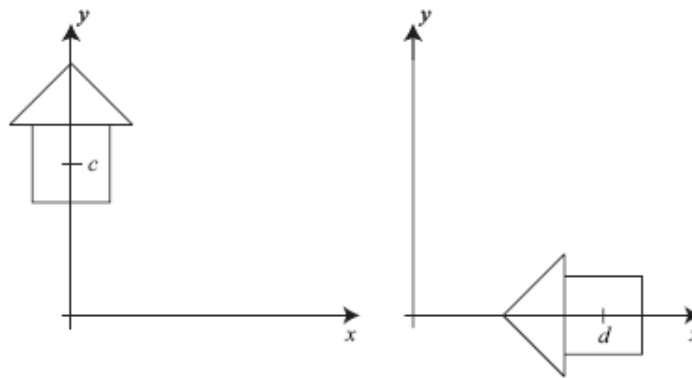
Question 1 (20 Marks)

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

- (a) Z buffer
- (b) two sided lighting
- (c) global illumination
- (d) gamut
- (e) Bresenham's algorithm

Question 2 (25 Marks)

- (a) Find the 3x3 transformation matrix for the transformation shown below.



- (b) Which algorithm is better suited for drawing transparent objects? Z buffer or BSP trees? Explain why.
- (c) A unit sphere (centre $(0, 0, 0)$, radius is 1) is illuminated by a directional light in direction $(1, 1, 1)$ viewed from eye position $(5, 5, 0)$. The intensity of the light is 0.8 and there is no ambient light. k_d is 0.4, k_s is 0.5 and the Phong exponent is 64. Compute the intensity of the illumination at the point $(1, 0, 0)$ on the sphere.

Give as much detail as you can about how to compute the illumination of this point as it appears to the eye. (I haven't given you all the information you need—in your answer you need to tell me what else you need.)

Question 3 (25 Marks)

A company that sells kitchens wants a computer system to display new kitchens to customers. The company has many different kitchen designs that have to be adapted to fit into the customer's kitchen. There are many different styles of cupboards and surface materials available, as well as many different sorts of door handles. The company wants as realistic as possible rendering of the final kitchen to be displayed interactively.

Design a computer system for this company.

Tell me (using diagrams where appropriate)

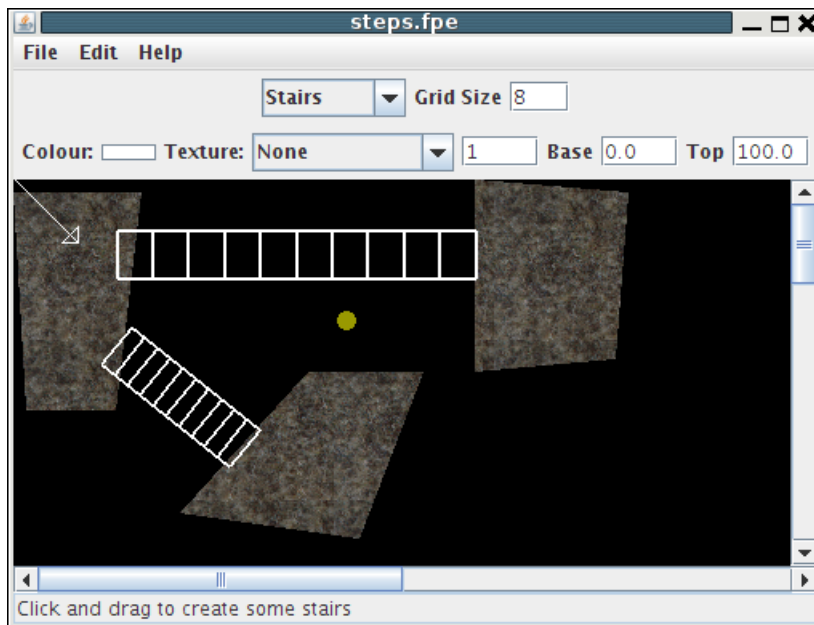
- A: what hardware your system would require,
- B: what software and algorithms your system would use,
- C: how you would store the information needed,
- D: and how a user would interact with your system.

Question 4 (30 Marks)

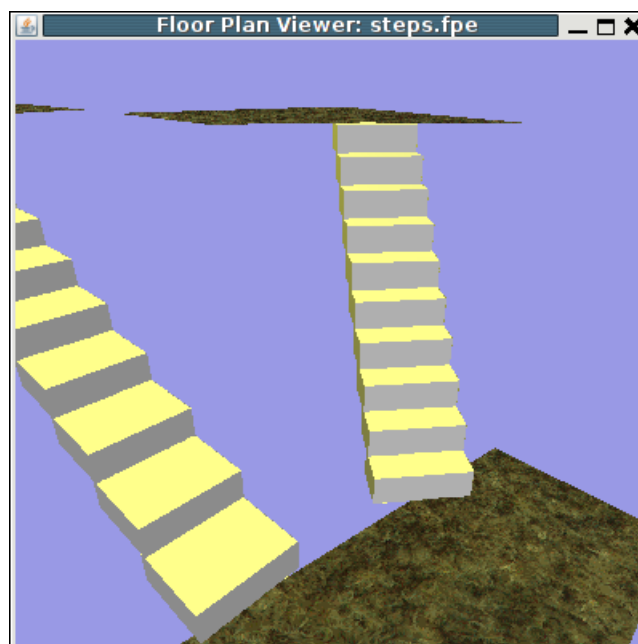
In this question you will modify my solution to the second assignment to add stairs.

A Stairs is defined by two control points. The first control point is the midpoint of the bottom end of the stairs. The second one is the midpoint of the top of the stairs. The first extra parameter is the y coordinate of the bottom of the stairs, the second is the y coordinate of the top. The Stairs themselves are 30 units wide and each step is 10 units high. The number of steps is difference between the extra parameters divided by 10 (the height of one step). To simplify things, assume that the difference is evenly divided by 10 and stairs do not have textures applied to them.

In FPEdit the stairs are drawn as a set of rectangles showing each step



In FPView, each step is a rectangular prism of appropriate size and orientation.



In the image above, each Stairs has a base height of 0 and top height of 100 (so there are 10 steps in each one). The viewpoint is the one shown in the FPEdit view.

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 FPCanvas.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 `StairsTool` that allows interactive creation of a `Stairs`.
- (b) Write a `paint` method for the `Stairs` class that draws the representation of the stairs used in FPEdit.
- (c) Write a `contains` method for the `Stairs` class. It should return true if the user clicks on the `Stairs`.
- (d) Write the `render3D` method for the `Stairs` class.
- (e) Write the `collide` method for the `Stairs` class so that the avatar can walk up or down the stairs. The bottom of the avatar should be on the top of the step it is above.

Hint: Look at `Avatar` to see how to draw rectangular prisms.

Avatar.java

```
1  import java.awt.*;
2  import java.util.*;
3  import javax.media.opengl.*;
4  import com.sun.opengl.util.*;
5
6  public class Avatar {
7      public static final double height = 80;
8      final double moveStep = 1;
9      final double turnStep = 0.03;
10     double turnIncrement;
11     double moveIncrement;
12     int turnframes;
13     int moveframes;
14     Vector3D pos;
15     Vector3D dir;
16
17     float t = 0;
18     public Avatar(Vector3D pos, Vector3D dir){
19         this.pos = pos;
20         this.dir = dir;
21     }
22
23
24     //turn for given number of frames
25     public void turn(int noframes){
26         if (noframes > 0) {
```

```

27         turnIncrement = turnStep;
28     } else {
29         turnIncrement = -turnStep;
30     }
31     turnframes = Math.abs(noframes);
32 }
33
34 //move forward or back for given number of frames
35 public void move(int noframes){
36     if (noframes > 0) {
37         moveIncrement = moveStep;
38     } else {
39         moveIncrement = -moveStep;
40     }
41     moveframes = Math.abs(noframes);
42 }
43
44 //update position, checking for collisions
45 public void advancePos(FPSShapeList shapeList){
46     Vector3D newpos;
47     if (turnframes > 0) {
48         dir = dir.rotatey(turnIncrement);
49         turnframes--;
50     }
51
52     if (moveframes > 0) {
53         newpos = pos.add(dir.scale(moveIncrement));
54         moveframes--;
55     } else {
56         newpos = pos; //not moving
57     }
58     pos = shapeList.collide(pos,newpos);
59 }
60
61 public Vector3D currentPos(){
62     return pos;
63 }
64
65 public Vector3D currentDir(){
66     return dir;
67 }
68
69
70 public void setColor(GL gl, Color c){
71     gl.glColor3ub((byte)c.getRed(),(byte)c.getGreen(),(byte)c.getBlue());
72 }
73
74 public void render3D(GL gl, GLDrawable glc){
75     setColor(gl,Color.white);
76     gl.glDisable( GL.GL_TEXTURE_2D );
77     // System.out.println("cDir = " + cDir.toString());
78     double angle = dir.azimuth();
79     gl.glPushMatrix();
80     gl.glTranslated(pos.x, pos.y-height/2, pos.z);
81     gl.glRotated(-Math.toDegrees(angle), 0, 1, 0);
82     gl.glScaled(5,height,25);

```

```

83         GLUT glut = new GLUT();
84         glut.glutSolidCube(1);
85         gl.glPopMatrix();
86     }
87 }

```

FPolygon.java

```

1  /** This class represents a polygon */
2  import java.awt.Point;
3  import java.awt.Color;
4  import java.awt.Polygon;
5  import java.util.*;
6  import javax.media.opengl.*;
7  import com.sun.opengl.util.texture.*;
8
9
10 public class FPPolygon extends FPShape {
11
12     public String extraName(int i){
13         return i==0 ? "Height" : null;
14     }
15
16     protected ArrayList<Point2D> pts2d;
17     protected int selection = -1;
18
19     public FPPolygon() {
20         pts2d = new ArrayList<Point2D>();
21     }
22
23     public String extraName(){
24         return "Height";
25     }
26
27     /** paint this curve into g.*/
28     public void paint(GL gl, GLDrawable glc){
29
30         if (fill != null || texture != null){
31             if (texture == null) {
32                 setColor(gl,fill);
33                 gl.glDisable( GL.GL_TEXTURE_2D );
34             } else {
35                 Texture gltexture = texture.getTexture(glc);
36                 gltexture.enable();
37                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_REPEAT );
38                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_REPEAT );
39                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER, GL.GL_LINEAR );
40                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER, GL.GL_LINEAR );
41                 gl.glTexEnvf( GL.GL_TEXTURE_ENV, GL.GL_TEXTURE_ENV_MODE,
42                             GL.GL_REPLACE );
43                 gltexture.bind();
44             }
45             gl.glBegin( GL.GL_POLYGON ); //draw the polygon
46             for(Point2D p : pts2d) {
47                 gl.glTexCoord2d(scale*p.x/100, -scale*p.y/100); //dodgy texture scaling

```

```

48         gl.glVertex2d(p.x, p.y);
49     }
50     gl.glEnd();
51
52 }
53
54 }
55
56 public void paintSelect(GL gl){
57     for(Point2D p : pts2d) {
58         drawPoint(gl,p.x,p.y);
59     }
60 }
61
62 public void render3D(GL gl, GLDrawable glc){
63     if (pts2d.size() < 3) return;
64     if (fill != null || texture != null){
65         if (texture == null) {
66             setColor(gl,fill);
67             gl.glDisable( GL.GL_TEXTURE_2D );
68         } else {
69             setColor(gl, Color.white);
70             Texture gltexture = texture.getTexture(glc);
71             gltexture.enable();
72             gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_REPEAT );
73             gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_REPEAT );
74             gl.glTexEnvf( GL.GL_TEXTURE_ENV, GL.GL_TEXTURE_ENV_MODE,
75                 GL.GL_MODULATE);
76             gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER ,
77                 GL.GL_NEAREST);
78             gl.glTexParameterf( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER ,
79                 GL.GL_NEAREST);
80             gltexture.bind();
81         }
82         gl.glPushMatrix();
83         gl.glBegin( GL.GL_POLYGON ); //draw the polygon
84
85         // First we have to work out the normal correctly; make sure it po ints
86         // in the correct direction.
87         //
88         // this calculation uses the cross product of the first two edges. It
89         // It is easy to show that the cross product has a y value of:
90         // (y1-y0).(x2-x1)-(x1-x0).(y2-y1). If this is positive, the norma l is (0,1,0),
91         // if this is negative then it is (0, -1, 0). This works for conca ve polygons.
92         // This should actually be stored in a field somewhere, so it is n ot continually
93         // recalculated but ...
94         Point2D p0 = pts2d.get(0);
95         Point2D p1 = pts2d.get(1);
96         Point2D p2 = pts2d.get(2);
97         double ycross = (p1.y-p0.y)*(p2.x-p1.x)-(p1.x-p0.x)*(p2.y-p1.y);
98         // System.out.println("ycross = " + ycross);
99         if(ycross >= 0){
100             gl.glNormal3d(0,1,0);
101         }
102         else {
103             gl.glNormal3d(0,-1,0);

```

```

104     }
105     for(Point2D p : pts2d) {
106         gl.glTexCoord2d(scale*p.x/100, scale*p.y/100); //dodgy texture scaling
107         gl.glVertex3d(p.x, extra[0], p.y);
108     }
109     gl.glEnd();
110     gl.glPopMatrix();
111 }
112 }
113
114
115
116 static final int EPSILON = 6; /* distance for picking */
117
118 /** return index of control point near to (x,y) or -1 if nothing near */
119 public Point selectPoint(Point p) {
120     double mind = Double.MAX_VALUE;
121     selection = -1;
122     for (int i = 0; i < pts2d.size(); i++){
123         Point2D p2 = pts2d.get(i);
124         double d = sqr(p2.x-p.x) + sqr(p2.y-p.y);
125         if (d < mind && d < EPSILON*EPSILON) {
126             mind = d;
127             selection = i;
128         }
129     }
130
131     if (selection == -1) {
132         return null;
133     } else {
134         return ((pts2d.get(selection)).toPoint());
135     }
136 }
137
138 // square of a double
139 static double sqr(double x) {
140     return x*x;
141 }
142
143 /** add a control point */
144 public void addPoint(Point p) {
145     pts2d.add(new Point2D(p));
146     selection = pts2d.size() - 1;
147 }
148
149 /** find the closest edge of a polygon to p and select end point
150 and return position on that edge as number between 0 and 1*/
151 protected double closestEdge(Polygon pl, Point p) {
152     Point2D closest = null;
153     int closesti;
154     for (int i = 0; i < pl.npoints; i++){
155         int iplus = (i+1) % pl.npoints;
156         Edge2D e = new Edge2D(pl.xpoints[i],pl.ypoints[i],pl.xpoints[iplus],pl.ypoints[iplus]);
157         Point2D pe = e.toLineSpace(p);
158         if (pe.getX() > 0 && pe.getX() < 1 && Math.abs(pe.getY()) < EPSILON) {
159             if (closest == null ||

```

```

160         Math.abs(pe.getY()) < Math.abs(closest.getY()) ) {
161             closest = pe;
162             selection = iplus;
163         }
164     }
165 }
166 }
167 return (closest == null) ? -1.0 : closest.getX();
168 }
169
170 /** set selected control point */
171 public void setPoint(Point p) {
172     if (selection >= 0) {
173         pts2d.set(selection, new Point2D(p));
174     }
175 }
176
177 /** remove specified control point */
178 public void removePoint() {
179     if (selection >= 0) {
180         pts2d.remove(selection);
181     }
182     selection = -1; //otherwise next control point becomes selected
183 }
184 }
185
186 /** Convert a List of Point2D to Polygon */
187 public Polygon toPolygon(List<Point2D> ps){
188     Polygon pts = new Polygon();
189     for (Point2D p2 : ps){
190         Point p = p2.toPoint();
191         pts.addPoint(p.x,p.y);
192     }
193     return pts;
194 }
195
196 public boolean contains(int x,int y){
197     return toPolygon(pts2d).contains(x,y);
198 }
199
200 public Vector3D collide(Vector3D from, Vector3D to){
201     final double increment = 5; //how far to move avatar towards polygon if not on it
202     if (contains((int) Math.round(to.x),(int) Math.round(to.z)) && to.y > extra[0]) {
203         //collision! move avatar towards correct height - Avatar.height above polygon's height
204         if (to.y > extra[0] + Avatar.height + increment) {
205             return new Vector3D(to.x,to.y-increment,to.z);
206         } else if (to.y < extra[0] + Avatar.height - increment) {
207             return new Vector3D(to.x,to.y+increment,to.z);
208         } else {
209             return new Vector3D(to.x,extra[0] + Avatar.height,to.z);
210         }
211     } else {
212         return null;
213     }
214 }
215

```

```

216     public void rotate(Point2D fixed, double angle){
217         apply(Matrix2D.rotateAbout(fixed, angle));
218     }
219
220     public void scale(Point2D fixed, double xscale, double yscale){
221         apply(Matrix2D.scaleAbout(fixed, xscale, yscale));
222     }
223
224     public void translate(int deltax, int deltay){
225         apply(new Matrix2D(1,0,deltax,0,1,deltay));
226     }
227
228     public Point2D centre(){
229         Point2D p0 = (pts2d.get(0));
230         double minx, miny, maxx, maxy;
231         minx = maxx = p0.x;
232         miny = maxy = p0.y;
233         for (Point2D p : pts2d){
234             if (p.x < minx){
235                 minx = p.x;
236             }
237             if (p.x > maxx){
238                 maxx = p.x;
239             }
240             if (p.y < miny){
241                 miny = p.y;
242             }
243             if (p.y > maxy){
244                 maxy = p.y;
245             }
246         }
247         return new Point2D((minx+maxx)/2, (miny+maxy)/2);
248     }
249
250     public void apply(Matrix2D t){
251         for (int i = 0; i < pts2d.size(); i++){
252             Point2D p = t.apply(pts2d.get(i));
253             pts2d.set(i,p);
254         }
255     }
256
257
258     public String toString() {
259         StringBuffer result = new StringBuffer(super.toString());
260         result.append(" " + pts2d.size());
261         for (Point2D p : pts2d) {
262             result.append(" " + p);
263         }
264         return result.toString();
265     }
266
267 // Parse the coordinates
268 public void fromTokens(StringTokenizer st) throws MalformedShapeException {
269     try {
270         selection = -1;
271         int n = Integer.parseInt(st.nextToken());

```

```

272     for (int i = 0; i < n; i++) {
273         pts2d.add(new Point2D(Double.parseDouble(st.nextToken()),
274                               Double.parseDouble(st.nextToken())));
275     }
276 }
277 }
278 catch (NoSuchElementException e) {
279     throw new MalformedShapeException(e.getMessage());
280 } catch (NumberFormatException e) {
281     throw new MalformedShapeException(e.getMessage());
282 }
283 }
284 }
285 }

```

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 }

```

ViewPoint.java

```

1  /** This class represents a viewpoint */
2  import java.awt.*;
3  import java.util.*;
4  import javax.media.opengl.*;
5
6
7  public class ViewPoint extends Wall {
8
9      public ViewPoint(){
10     }
11
12     /* construct a viewpoint given position and direction */
13     public ViewPoint(Vector3D position, Vector3D direction){
14
15         pts2d.add(new Point2D(position.x,position.z));
16         extra[0] = position.y;
17
18         Vector3D look = position.add(direction);
19         pts2d.add(new Point2D(look.x,look.z));
20         extra[1] = look.y;
21     }
22
23
24     public float lineWidth() {
25         return 1.0f;
26     }
27
28     public Vector3D getViewerPos(){
29         Point2D p0 = pts2d.get(0);
30         return new Vector3D(p0.x,extra[0],p0.y);

```

```

31     }
32
33     public Vector3D getViewerDir(){
34         Point2D p1 = pts2d.get(1);
35         Vector3D look = new Vector3D(p1.x,extra[1],p1.y);
36         Vector3D viewdir = look.subtract(getViewerPos());
37         return viewdir.normalize();
38     }
39
40     public void render3D(GL gl, GLDrawable glc){
41     }
42
43     /** paint the viewpoint as a line with an arrow*/
44     public void paint(GL gl, GLDrawable glc){
45         double length = 10; //length of arrow head
46         double angle = Math.PI/4; //angle between arrow head and body
47         if (fill != null){
48             setColor(gl,fill);
49             gl.glDisable( GL.GL_TEXTURE_2D );
50         }
51         gl.glLineWidth(lineWidth());
52         Point2D start = pts2d.get(0);
53         Point2D end = pts2d.get(1);
54         double linelength = end.subtract(start).length();
55         if (linelength == 0.0) {
56             return; //nothing to draw
57         }
58         // length units back along line from end
59         Point2D tip = end.interp(start,length/linelength);
60         Point2D leftTip = Matrix2D.rotateAbout(end,angle).apply(tip);
61         Point2D rightTip = Matrix2D.rotateAbout(end,-angle).apply(tip);
62
63         gl.glBegin( GL.GL_LINE_STRIP); //draw the arrow
64             gl.glVertex2d(start.x, start.y);
65             gl.glVertex2d(end.x, end.y);
66             gl.glVertex2d(leftTip.x, leftTip.y);
67             gl.glVertex2d(rightTip.x, rightTip.y);
68             gl.glVertex2d(end.x, end.y);
69         gl.glEnd();
70     }
71
72
73 }

```

Vector3D.java

```

1     /** This class represents a 3D vector.
2         */
3     import javax.media.opengl.*;
4
5     public class Vector3D {
6         public 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 //rotate about x axis
75 public Vector3D rotatex(double ang){
76     return new Vector3D(x,Math.cos(ang)*y - Math.sin(ang)*z, Math.sin(ang)*y + Math.cos(ang)*z);
77 }
78
79 //rotate about y axis
80 public Vector3D rotatey(double ang){
81     return new Vector3D(Math.cos(ang)*x + Math.sin(ang)*z, y, -Math.sin(ang)*x + Math.cos(ang)*z);
82 }
83
84 //rotate about z axis
85 public Vector3D rotatez(double ang){
86     return new Vector3D(Math.cos(ang)*x - Math.sin(ang)*y, Math.sin(ang)*x + Math.cos(ang)*y, z);
87 }
88
89 //length of vector
90 public double length(){
91     return Math.sqrt(x*x+y*y+z*z);
92 }
93
94 //make it length one
95 public Vector3D normalize(){
96     return scale(1/length());
97 }
98
99 //dot product
100 public double dot(Vector3D q){
101     return x*q.x + y*q.y + z*q.z;
102 }
103
104 //cross product
105 public Vector3D cross(Vector3D q){
106     return new Vector3D(y*q.z - z*q.y,
107                        z*q.x - x*q.z,
108                        x*q.y - y*q.x);
109 }
110
111 //for converting to spherical coordinates
112 //think of earth with the y axis going from South to North pole
113 //(note that Maths texts have this as the z axis instead)
114 //azimuth is the latitude (in radians)
115 public double azimuth() {
116     return Math.atan2(z,x);
117 }
118 //elevation is the longitude (in radians)
119 public double elevation() {
120     return Math.acos(y/length());
121 }

```

```

122
123 //convert to Cartesian coordinates from spherical (r,azimuth,elevation)
124 public static Vector3D fromSpherical(double r, double azimuth, double elevation){
125     return new Vector3D(r*Math.sin(elevation)*Math.cos(azimuth),
126                        r*Math.cos(elevation),
127                        r*Math.sin(elevation)*Math.sin(azimuth));
128 }
129
130
131
132 public String toString(){
133     return x+" "+y+" "+z;
134 }
135
136 //this does exactly the same thing as gluLookat, except that it takes
137 //three vectors
138 public static void lookAt(GL gl, Vector3D eye, Vector3D centre, Vector3D up) {
139     Vector3D n = eye.subtract(centre).normalize();
140     Vector3D u = up.cross(n).normalize();
141     Vector3D v = n.cross(u);
142     /* OpenGL matrices are in column major order */
143     double[] m = {u.x, v.x, n.x, 0,
144                  u.y, v.y, n.y, 0,
145                  u.z, v.z, n.z, 0,
146                  0, 0, 0, 1};
147
148     /* do the rotation*/
149     gl.glMultMatrixd(m,0);
150
151     /* Translate Eye to Origin */
152     gl.glTranslated(-eye.x, -eye.y, -eye.z);
153 }
154
155 //Apply local to world transform given basis and origin
156
157 public static void localToWorld(GL gl, Vector3D u, Vector3D v, Vector3D n, Vector3D O) {
158     /* OpenGL matrices are in column major order */
159     double[] m = {u.x, u.y, u.z, 0,
160                  v.x, v.y, v.z, 0,
161                  n.x, n.y, n.z, 0,
162                  O.x, O.y, O.z, 1};
163
164     /* apply the transform*/
165     gl.glMultMatrixd(m,0);
166 }
167
168 }

```

Wall.java

```

1 /** This class represents a wall */
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 Wall extends FPPolygon {
9
10
11     public String extraName(int i){
12         return i==0 ? "Base" : "Top";
13     }
14
15     public float lineWidth() {
16         return 4.0f;
17     }
18
19     /** paint the wall using gl.*/
20     public void paint(GL gl, GLDrawable glc){
21
22         if (fill != null){
23             setColor(gl,fill);
24             gl.glDisable( GL.GL_TEXTURE_2D );
25         }
26         gl.glLineWidth(lineWidth());
27         gl.glBegin( GL.GL_LINES ); //draw the wall
28         for(int i = 0; i < 2; i++) {
29             Point2D p = (Point2D)(pts2d.get(i));
30             gl.glVertex2d(p.x, p.y);
31         }
32         gl.glEnd();
33     }
34
35
36     public void render3D(GL gl, GLDrawable glc){
37         if (fill != null || texture != null){
38             if (texture == null) {
39                 setColor(gl,fill);
40                 gl.glDisable( GL.GL_TEXTURE_2D );
41             } else {
42                 setColor(gl, Color.white);
43                 Texture gltexture = texture.getTexture(glc);
44                 gltexture.enable();
45                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_S, GL.GL_REPEAT );
46                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_WRAP_T, GL.GL_REPEAT );
47                 gl.glTexEnvf( GL.GL_TEXTURE_ENV, GL.GL_TEXTURE_ENV_MODE,
48                             GL.GL_MODULATE );
49                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MAG_FILTER ,
50                                     GL.GL_NEAREST);
51                 gl.glTexParameteri( GL.GL_TEXTURE_2D, GL.GL_TEXTURE_MIN_FILTER ,
52                                     GL.GL_NEAREST);
53
54                 gltexture.bind();
55             }
56             gl.glPushMatrix();
57             Point2D start = (Point2D) (pts2d.get(0));
58             Point2D end = (Point2D) (pts2d.get(1));
59             double wallLength = Math.sqrt((start.x-end.x)*(start.x-end.x) + (start.y-end.y)*(start
60             double base = extra[0];
61             double top = extra[1];

```

```

62         gl.glBegin(GL.GL_POLYGON);
63         gl.glNormal3d((end.y-start.y)/wallLength, 0, -(end.x-start.x)/wallLength);
64         gl.glTexCoord2d(start.x/100, scale*base/100);
65         gl.glVertex3d(start.x, base, start.y);
66         gl.glTexCoord2d(start.x/100, scale*top/100);
67         gl.glVertex3d(start.x, top, start.y);
68         gl.glTexCoord2d(start.x/100+scale*wallLength/100, scale*top/100);
69         gl.glVertex3d(end.x, top, end.y);
70         gl.glTexCoord2d(start.x/100+scale*wallLength/100, scale*base/100);
71         gl.glVertex3d(end.x, base, end.y);
72         gl.glEnd();
73
74         gl.glPopMatrix();
75     }
76 }
77
78 /** add a control point */
79 public void addPoint(Point p) {
80     if (pts2d.size() < 2) {
81         super.addPoint(p);
82     }
83 }
84
85
86 /** remove specified control point */
87 public void removePoint() {
88     return;
89 }
90
91 public boolean contains(int x,int y){
92     Edge2D e = new Edge2D((Point2D)pts2d.get(0),
93                          (Point2D)pts2d.get(1));
94     Point2D pe = e.toLineSpace(new Point(x,y));
95     return (pe.getX() > 0 && pe.getX() < 1 && Math.abs(pe.getY()) < EPSILON);
96 }
97
98 public Vector3D collide(Vector3D from, Vector3D to){
99     if (contains((int) Math.round(to.x),(int) Math.round(to.z)) && to.y > extra[0] && to.y < extra[1])
100         return from; //collision! put avatar back where it came from
101     } else {
102         return null;
103     }
104 }
105
106 }

```

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(FPEdit editor) {
13         super(editor);
14         extra[0] = 0.0;
15         extra[1] = 2.4; //default height
16     }
17
18     public String getName() {
19         return "Wall";
20     }
21
22     public String getMessage() {
23         return "Click and drag to create a wall";
24     }
25
26     public String extraName(int i){
27         return newWall().extraName(i);
28     }
29
30     protected Wall p; /* Wall being created */
31
32
33     public Wall newWall(){
34         return new Wall();
35     }
36
37     // create the polygon
38     public void mousePressed(MouseEvent e) {
39         p = newWall();
40         editor.getProperties(p);
41
42         canvas.getShapes().add(p);
43         canvas.select(p);
44         p.addPoint(gridPoint(e));
45         p.addPoint(gridPoint(e));
46         canvas.repaint();
47     }
48
49
50     public void mouseDragged(MouseEvent e) {
51         if (p != null){
52             p.setPoint(gridPoint(e));
53             canvas.repaint();
54         }
55     }
56
57     public void mouseReleased(MouseEvent e) {
58         mouseDragged(e);
59         p = null;
60     }
61
62 }

```

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

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

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.