```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;
import java.awt.event.*;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.awt.*;
import java.awt.image.DataBufferInt;
import java.awt.image.BufferedImage;
import java.util.Enumeration;
import java.text.NumberFormat;
import java.util.*;


public class bezier extends Applet {
    /*****************************************
     *****************************************
     ***********GLOBAL VARIABLES***************
     *****************************************
     *****************************************/

    /* Applet variables */
    Canvas3D            canvas;
    GraphicsContext3D   gc = null;
    BranchGroup objRoot;

    /* Mouse/keyboard Motion variables */
    WakeupCriterion[]   mouseEvents;
    WakeupOr            mouseCriterion;
    boolean             rendererRunning = true;
    Point3d  screenpt = new Point3d();
    Point3d t3d = null;
    Point3d myOrigin = new Point3d();
    Point3d initialMouseClick = new Point3d();
    Point3d currentMouseClick = new Point3d();
    int MOVE = 0;
    int LAST_MOVE = 0;

    /* These variables are used to construct the bezier */
    double[] a = new double[3];
    double[] b = new double[3];
    double[] c= new double[3];
    double[] d= new double[3];
    Point3d temppoint1 = new Point3d();
    Point3d temppoint2 = new Point3d();
```

```
   int test_pick = -1;
   Color4f color = new Color4f(Color.RED);
   Color4f color_points = new Color4f(Color.WHITE);
   int ret[] = new int[2];
   boolean M_IS_PRESSED = false;
   boolean T_Pressed = false;
   boolean S_Pressed = false;
   boolean R_Pressed = false;
   boolean DRAWING = true;
   int lastx, lasty;
   double[] tmp = new double[3];
   Color4f color_transfer;
   Color4f color_scale;
   Color4f color_rotate;

   /* These variables are for forward ifferencing */
   double[] f = new double[4];
   double[] fd = new double[2];
   double[] fdd  = new double[2];
   double[] fddd  = new double[2];
   double r = 0.05;

/* These variables are for drawing shit */
   Vector points = new Vector();  // This will hold all of our points
   int mk = 0;
   int selectedPoint = -1;
   LineArray           line = null;
   Point3d             coords[] = new Point3d[2];
   Transform3D         ipToVworld = new Transform3D();
   IndexedPointArray ipa = null;
   Point3d c1 = new Point3d(0.01,0.01,0);
   Point3d c2 = new Point3d(-0.01,0.01,0);
   Point3d c3 = new Point3d(-0.01,-0.01,0);
   Point3d c4 = new Point3d(0.01,-0.01,0);


   /*** THESE ARE TEMP VARIBLES ******/
    Point3f[] plaPts = new Point3f[2];
   double[] testDistance = new double[3];
   double[] testClickDistance = new double[3];
   double dxd;
   double dyd;

   Point3d prev = new Point3d(0.0,0.0,0.0);
   Point3d orig = new Point3d(0.0,0.0,0.0);
   Point3d curr = new Point3d();
```

```
/*****************************************
 *****************************************
 ***********MAIN APPLICATION***************
 *****************************************
 *****************************************/

/* Determine if we hit the point or not */
public int hitsPoint(Point3d p1){
 p1.get(testClickDistance);
    for(int i=0;i<points.size();i++) {
     ((Point3d)points.elementAt(i)).get(testDistance);
     dxd = (testClickDistance[0] - testDistance[0] );
        dyd = (testClickDistance[1] - testDistance[1] );
        if( Math.sqrt(dxd*dxd + dyd*dyd) <= 0.05)
            return i;
 }
    return -1;
}

public int checkPoints(int k){
 if (b[0] <-4 )
    return (k += 2);
 if (a[0] <-4 )
    return (k += 1);
 if (c[0] <-4 )
    return (k += 3);
 if (d[0] <-4 )
    return (k += 4);
 return k;
}

public boolean validatePoints(){
    if (a[0] > -4 && b[0] > -4 && c[0] > -4 && d[0] > -4)
     return true;
 else
     return false;
}
public void renderMe(){
 /* Prepare the new frame */
    gc.clear();
 int flags = GeometryArray.COORDINATES | GeometryArray.COLOR_4;
 /* Draw bezier if we have enough points...This is the FASSST method  */

 int k = 0;
 while(k<points.size() && points.size()-k >= 4){
            ((Point3d)points.elementAt(k+0)).get(a);
            ((Point3d)points.elementAt(k+1)).get(b);
            ((Point3d)points.elementAt(k+2)).get(c);
```

```
            ((Point3d)points.elementAt(k+3)).get(d);

            k = checkPoints(k);

     if (validatePoints()){

     /* Pre compute some variables */
          preCompute();

     /* Evaluate bezier WITH forward differencing */
           for (double t = 0 ; t < 0.99 ; t += r){
         temppoint1.set(f[0], f[1], 0);
         calculateNext();
         temppoint2.set(f[0], f[1], 0);

               line = new LineArray(2, flags);
         coords[0] = temppoint1;
         coords[1] = temppoint2;
         line.setCoordinates(0, coords);
         line.setColor(0, new Color4f(Color.WHITE));
         line.setColor(1, new Color4f(Color.WHITE));
         if ( test_pick != -1 && ret[0] <= k && ret[1] >= k) {
            line.setColor(0, color);
                  line.setColor(1, color);
         }
               gc.draw(line);
       }
     k+=3;
}
}
   /* now we draw the points and lines between them */
if (M_IS_PRESSED || DRAWING){
           for(int i=0;i<points.size();i++) {
               ((Point3d)points.elementAt(i)).get(tmp);
         if (  tmp[0] > -4 ){
                  drawSqPoint(c1,c2,c3,c4,i);
             if ( mk > 0 )
              drawLine(i);
          mk=1;
              }
         if (  tmp[0] == -5 )
          mk = 0;
     }
     mk = 0;
}

/* Clean up and swap bufferes */
canvas.stopRenderer();
```

```
        rendererRunning = false;
        canvas.swap();
     gc.flush(false);

    }

    Group createLineTypes() {

     Group lineGroup = new Group();

     Appearance app = new Appearance();
     ColoringAttributes ca = new ColoringAttributes(new Color3f(1.0f, 1.0f,
1.0f),
                                        ColoringAttributes.SHADE_FLAT);
     app.setColoringAttributes(ca);

     // Plain line
     plaPts[0] = new Point3f(-0.9f, -0.7f, 0.0f);
     plaPts[1] = new Point3f(-0.5f, 0.7f, 0.0f);
     LineArray pla = new LineArray(2, LineArray.COORDINATES);
     pla.setCoordinates(0, plaPts);
     Shape3D plShape = new Shape3D(pla, app);
     lineGroup.addChild(plShape);

     return lineGroup;

    }

    /* Here we just set up the applet and stuff */
    public void init() {
        setLayout(new BorderLayout());

        canvas = new Canvas3D(SimpleUniverse.getPreferredConfiguration());

        add("Center", canvas);

        // set up the graphics context
        gc = canvas.getGraphicsContext3D();
        gc.setBufferOverride(true);

        // Create a simple scene and attach it to the virtual universe
        SimpleUniverse u = new SimpleUniverse(canvas);

        // This will move the ViewPlatform back a bit so the
        // objects in the scene can be viewed.
        u.getViewingPlatform().setNominalViewingTransform();

     /* This will transform our stff */
```

```
 TransformGroup objTrans = new TransformGroup();
 objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
 objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

 //objTrans.addChild(createLineTypes());

 Callback myTrans = new Callback();
 myTrans.setTransformGroup(objTrans);
 objTrans.addChild(myTrans);
 BoundingSphere bounds = new BoundingSphere(new Point3d(100,100,100),100.0);
    myTrans.setSchedulingBounds(bounds);
    objRoot = new BranchGroup();



 objRoot.addChild(objTrans);

    u.addBranchGraph(objRoot);
}

public class Callback extends Behavior {
 TransformGroup localTrans = null;
 protected Transform3D transformY;
 protected Transform3D transformX;
 protected Transform3D currXform;
 double  y_angle;
 double  y_factor;
 boolean TESTING = false;

 public void setTransformGroup(TransformGroup t ){
     currXform = new Transform3D();
     transformY = new Transform3D();
     transformX = new Transform3D();
     this.localTrans = t;
 }

 public void initialize(){
        mouseEvents = new WakeupCriterion[4];
        mouseEvents[0] = new WakeupOnAWTEvent(MouseEvent.MOUSE_DRAGGED);
        mouseEvents[1] = new WakeupOnAWTEvent(MouseEvent.MOUSE_PRESSED);
        mouseEvents[2] = new WakeupOnAWTEvent(MouseEvent.MOUSE_RELEASED);
        mouseEvents[3] = new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED);
        mouseCriterion = new WakeupOr(mouseEvents);
        wakeupOn (mouseCriterion);
        setSchedulingBounds(new BoundingSphere(new Point3d(),1000));
     y_angle = 0;
     y_factor = .03;
 }
```

```
public void findSelectedInterval(int t)
{
    ret[0] = t;
    for (int y=t; y < points.size(); y++)
     {
           ((Point3d)points.elementAt(y+0)).get(a);
     if (a[0] < -4 ){
         ret[1] = (y-1);
         break;
     }
}
    for (int y=t; y >= 0; y--)
     {
           ((Point3d)points.elementAt(y+0)).get(a);
     if ( y == 0){
         ret[0] = y;
         break;
     }

     if (a[0] < -4){
               ret[0] = y+1;
         break;
     }
}
}
public int testPickLine(Point3d p1){
int k = 0;
while(k<points.size() && points.size()-k >= 4){
           ((Point3d)points.elementAt(k+0)).get(a);
           ((Point3d)points.elementAt(k+1)).get(b);
           ((Point3d)points.elementAt(k+2)).get(c);
           ((Point3d)points.elementAt(k+3)).get(d);

           k = checkPoints(k);

     if (validatePoints()){

     /* Pre compute some variables */
          preCompute();

     /* Evaluate bezier WITH forward differencing */
           for (double t = 0 ; t < 0.99 ; t += r){
         temppoint1.set(f[0], f[1], 0);
         if ( p1.distance(temppoint1) < 0.05) {
          findSelectedInterval(k);
          test_pick = 1;
          renderMe();
```

```
          return k;
         }
         calculateNext();
         temppoint2.set(f[0], f[1], 0);
       }
     k+=3;
}
}
return -1;
}

public void trCoord(int x, int y){
    canvas.getPixelLocationInImagePlate(x, y, screenpt);
    canvas.getImagePlateToVworld(ipToVworld);
    ipToVworld.transform(screenpt);
}
public boolean isDoubleClick(int x, int y, int lastx, int lasty) {
    if ( Math.abs(x-lastx) < 2 && Math.abs(y - lasty) < 2 )
     return true;
    else
     return false;
}
public void toggleDrawPoints() {
    if (M_IS_PRESSED) {
     M_IS_PRESSED = false;
     selectedPoint = -1;
     DRAWING = false;
           color_points = new Color4f(Color.WHITE);
    }
    else{
     T_Pressed = false;
     S_Pressed = false;
     M_IS_PRESSED = true;
     DRAWING = false;
           color_points = new Color4f(Color.BLUE);
    }
}


public void setUPTransfer(){
    if ( T_Pressed ) {
     T_Pressed = false;
    }
    else {
           color_points = new Color4f(Color.WHITE);
     T_Pressed = true;
     M_IS_PRESSED = false;
     S_Pressed = false;
```

```
     R_Pressed = false;
     DRAWING = false;
     color_transfer = new Color4f(Color.YELLOW);
    }
    renderMe();
}

public void setUPSclae(){
    if ( S_Pressed ) {
     S_Pressed = false;
    }
    else {
           color_points = new Color4f(Color.WHITE);
     T_Pressed = false;
     M_IS_PRESSED = false;
     S_Pressed = true;
     R_Pressed = false;
     DRAWING = false;
     color_scale = new Color4f(Color.ORANGE);
    }
    renderMe();
}

public void setUPRotate(){
    if ( R_Pressed ) {
     R_Pressed = false;
    }
    else {
           color_points = new Color4f(Color.WHITE);
     T_Pressed = false;
     M_IS_PRESSED = false;
     S_Pressed = false;
     R_Pressed = true;
     DRAWING = false;
     color_rotate = new Color4f(Color.PINK);
    }
    renderMe();
}

public void centroid(){
    double t_norm = (double)(1.0/(ret[1]-ret[0]));
    System.out.println("t_norm is" + t_norm);
    double[] t_temp = new double[3];

    double[] t_temp_final = new double[3];
    t_temp_final[2] = 0;

    int c = ret[0];
```

```
     while (c <= ret[1]){
      System.out.println("calcing centroid " + t_temp_final[0]);
      ((Point3d)points.elementAt(c)).get(t_temp);
          t_temp_final[0] += t_temp[0]*t_norm;
          t_temp_final[1] += t_temp[1]*t_norm;
      c++;
    }

     myOrigin.set(t_temp_final);
}

public void updateSelectedCurve(){

    int c = ret[0];


    while (c <= ret[1]){
         myOrigin.negate();
               ((Point3d)points.elementAt(c)).add(myOrigin);
         currXform.transform(((Point3d)points.elementAt(c)));
         myOrigin.negate();
               ((Point3d)points.elementAt(c)).add(myOrigin);
      c++;
    }
    renderMe();

}

   public void processStimulus (Enumeration criteria) {
       WakeupCriterion wakeup;
       AWTEvent[] event;
       int id;
       int dx, dy;

       while (criteria.hasMoreElements()) {
           wakeup = (WakeupCriterion) criteria.nextElement();
           if (wakeup instanceof WakeupOnAWTEvent) {
               event = ((WakeupOnAWTEvent)wakeup).getAWTEvent();
               for (int i=0; i<event.length; i++) {
           if (event[i] instanceof KeyEvent){
              KeyEvent kevent = (KeyEvent)event[i];
              id = event[i].getID();
              switch(id) {
                      case KeyEvent.KEY_PRESSED:
                 if ( kevent.getKeyCode() == KeyEvent.VK_T) {

                     setUPTransfer();
```

```
                    }
                    if ( kevent.getKeyCode() == KeyEvent.VK_R) {

                        setUPRotate();


                    }

                    if ( kevent.getKeyCode() == KeyEvent.VK_S) {
                        setUPSclae();
                    }

                    if ( kevent.getKeyCode() == KeyEvent.VK_M) {
                     toggleDrawPoints();
                     renderMe();
                    }
                            break;
                }
            }


            if (event[i] instanceof MouseEvent){
                 MouseEvent mevent = (MouseEvent) event[i];
                 if (!mevent.isMetaDown() && !mevent.isAltDown()){
                     id = event[i].getID();
                     switch (id) {

               case MouseEvent.MOUSE_PRESSED:
                            initialMouseClick = screenpt;
                        trCoord(mevent.getX(), mevent.getY());
                     System.out.println("I have clicked at" + screenpt);
                            if (!DRAWING){
                     if (M_IS_PRESSED){
                      System.out.println("Trying to see if I hit pt");
                         selectedPoint = hitsPoint(screenpt);
                      System.out.println("Select Point is: " +
selectedPoint);
                      renderMe();
                     }
                     test_pick = testPickLine(screenpt);
                                 if ( test_pick != -1 )
                      centroid();
                     System.out.println("The centroid is at: " + myOrigin);
                      prev = orig;
                   if ( !T_Pressed && !S_Pressed && !R_Pressed &&
                                 !M_IS_PRESSED && selectedPoint < 0 &&
test_pick == -1 ){
```

```
                        DRAWING = true;
                    }
                }
                            if (DRAWING){
                    if ( !isDoubleClick(mevent.getX(), mevent.getY(),lastx,
lasty)) {
                            points.addElement(new Point3d(screenpt));
                            //selectedPoint = points.size()-1;
                                    renderMe();

                    }
                    else {
                                     points.addElement(new Point3d(-5,-5,-
5));
                       DRAWING = false;
                       }

                }
                        lastx = mevent.getX();
                   lasty = mevent.getY();

                break;

               case MouseEvent.MOUSE_DRAGGED:

                 if (M_IS_PRESSED) {
                     if(selectedPoint != -1){
                              t3d = new Point3d();


canvas.getPixelLocationInImagePlate(mevent.getX(), mevent.getY(),
((Point3d)points.elementAt(selectedPoint)));

canvas.getImagePlateToVworld(ipToVworld);
                                    //ipToVworld.transform(t3d);

ipToVworld.transform(((Point3d)points.elementAt(selectedPoint)));

                                    //points.set(selectedPoint, t3d);
                         renderMe();
                 }
                }

                 if (T_Pressed) {
                     if ( test_pick != -1){
                     Point3d tempCurr = new Point3d();
```

```
canvas.getPixelLocationInImagePlate(mevent.getX(), mevent.getY(), tempCurr);
                                    canvas.getImagePlateToVworld(ipToVworld);
                                  ipToVworld.transform(tempCurr);


                          int c = ret[0];

                                    curr = new Point3d(tempCurr);
                          if ( prev.equals(orig))
                            prev = curr;

                          curr.sub(prev);




                                    while (c <= ret[1]){



                          ((Point3d)points.elementAt(c)).add(curr);

                                   c++;
                                    }

                          prev = tempCurr;

                         renderMe();
                          }

                         }


                        if (S_Pressed){

                             if ( test_pick != -1){
                              currXform = new Transform3D();

                            Point3d sTemp = new Point3d();
                            Point3d iTemp = new Point3d(initialMouseClick);

canvas.getPixelLocationInImagePlate(mevent.getX(), mevent.getY(), sTemp);

canvas.getImagePlateToVworld(ipToVworld);
                                     //ipToVworld.transform(t3d);
                                        ipToVworld.transform(sTemp);
```

```
                        currentMouseClick = new Point3d(sTemp);

                    dx = mevent.getX() - lastx;
                    y_angle = dx * y_factor;
                    double scaleSize = 0;
                    double mul = 0;
                    if ( dx > 0){
                     //                          scaleSize =
currentMouseClick.distance(myOrigin)/iTemp.distance(myOrigin);
                       scaleSize = 1.1;
                     System.out.println("Now scale factor is" + scaleSize);
                     mul = 0.9;
                    }
                    else{
                     //                        scaleSize =
(currentMouseClick.distance(myOrigin)/iTemp.distance(myOrigin))/2;
                                                scaleSize = 0.9;
                     mul = -0.9;
                    }

                    currXform.setScale(scaleSize);

                              // Update xform
                                updateSelectedCurve();

                    System.out.println("Now Pla pts are : " + plaPts[0]);
                    transformY.setScale(mul);
                    lastx = mevent.getX();
                     }
                }



                if (R_Pressed){

                     if ( test_pick != -1){
                                    Point3d p1 = new
Point3d(initialMouseClick);
                                    p1.sub(myOrigin);
                            Point3d sTemp = new Point3d();

canvas.getPixelLocationInImagePlate(mevent.getX(), mevent.getY(), sTemp);

canvas.getImagePlateToVworld(ipToVworld);
                                  //ipToVworld.transform(t3d);
                                    ipToVworld.transform(sTemp);
                        sTemp.sub(myOrigin);
```

```
          Vector3d v1 = new Vector3d(p1);
          Vector3d v2 = new Vector3d(sTemp);
                        Vector3d v3 = new Vector3d();
          currXform = new Transform3D();
        currXform.setScale(1);
        dx = mevent.getX() - lastx;
        dy = mevent.getY() - lasty;

                    double[] vtemp = new double[3];
        v3.cross(v1,v2);
        v3.get(vtemp);
        System.out.println("cross is " + v3);

        if (dx > 0){
         y_angle = 0.1;
         System.out.println("Rotating positve " + v2.dot(v1));
         //y_angle = Math.abs(dx)*y_factor;

        }

        else{
         System.out.println("Rotating negative " + v1.dot(v2));
         y_angle = -0.1;
         //y_angle = -1*(Math.abs(dx*y_factor));
        }

        System.out.println("angle is: " + y_angle);


        currXform.rotZ(y_angle);
                  // Update xform
                    updateSelectedCurve();

        System.out.println("Now Pla pts are : " + plaPts[0]);
        lastx = mevent.getX();
        lasty = mevent.getY();
        if (LAST_MOVE != MOVE)
        LAST_MOVE = MOVE;
         }
 }

 break;

case MouseEvent.MOUSE_RELEASED:
 T_Pressed = false;
 S_Pressed = false;
 R_Pressed = false;
```

```
                test_pick = -1;
                renderMe();
                            break;
                        }
                    }
                }

            }
        }
        }

          wakeupOn (mouseCriterion);
    }

}



/**************************************************************************
 **************************************************************************
 ******************HELPER FUNCTIONS****************************************
 **************************************************************************
 **************************************************************************/

public void preCompute(){

 double A[] = new double[2];
 double B[] = new double[2];
 double C[] = new double[2];

 C[0] = 3*(b[0]-a[0]);
 B[0] = 3*(c[0] - b[0])-C[0];
    A[0]= d[0] - a[0] - C[0] - B[0];

 C[1] = 3*(b[1]-a[1]);
 B[1] = 3*(c[1] -b[1])-C[1];
    A[1]= d[1] - a[1] - C[1] - B[1];

 f[0] = a[0];
 fd[0] = A[0]*(r*r*r)+B[0]*(r*r)+C[0]*r;
 fdd[0] = 6*A[0]*(r*r*r)+2*B[0]*(r*r);
 fddd[0] = 6*A[0]*(r*r*r);

 f[1] = a[1];
 fd[1] = A[1]*(r*r*r)+B[1]*(r*r)+C[1]*r;
 fdd[1] = 6*A[1]*(r*r*r)+2*B[1]*(r*r);
 fddd[1] = 6*A[1]*(r*r*r);
```

```
    }

    public void calculateNext() {

     f[0] = f[0] + fd[0];
     f[1] = f[1] + fd[1];

        fd[0] = fd[0] + fdd[0];
     fdd[0] = fdd[0] + fddd[0];
        fd[1] = fd[1] + fdd[1];
     fdd[1] = fdd[1] + fddd[1];
    }

    public void drawSqPoint(Point3d c1, Point3d c2, Point3d c3, Point3d c4, int
i) {
          int flags = GeometryArray.COORDINATES | GeometryArray.COLOR_4;

           c1.set(0.01,0.01,0);
           c2.set(-0.01,0.01,0);
           c3.set(-0.01,-0.01,0);
           c4.set(0.01,-0.01,0);


           IndexedQuadArray geom = new IndexedQuadArray(4, flags, 4);
           c1.add((Point3d)points.elementAt(i));
           c2.add((Point3d)points.elementAt(i));
           c3.add((Point3d)points.elementAt(i));
           c4.add((Point3d)points.elementAt(i));

        Point3d[] coordinates = {c1,c2,c3,c4};

        int[] indices = { 0, 1, 2, 3 };

        geom.setCoordinates(0, coordinates);
        geom.setCoordinateIndices(0, indices);

        Color4f sqCol;
           if (selectedPoint != -1 && selectedPoint == i )
               sqCol = new Color4f(Color.GREEN);
        else
            sqCol = color_points;

        if ( T_Pressed && ret[0] <= i && ret[1] >= i)
            sqCol = color_transfer;

        if ( R_Pressed && ret[0] <= i && ret[1] >= i)
            sqCol = color_rotate;
```

```
        if ( S_Pressed && ret[0] <= i && ret[1] >= i)
            sqCol = color_scale;

        geom.setColor(0, sqCol);
        geom.setColor(1, sqCol);
        geom.setColor(2, sqCol);
        geom.setColor(3, sqCol);

        gc.draw(geom);

    }


  public void drawLine(int j) {
          line = new LineArray(2, LineArray.COORDINATES);

          coords[0] = (Point3d)points.elementAt(j-1);
          coords[1] = (Point3d)points.elementAt(j);

          line.setCoordinates(0, coords);

          gc.draw(line);
}



public static class Cubic
{

    public static double[][] BEZIER = {      //<b> Bezier basis matrix</b>
     {-1  ,  3  , -3  , 1 },
     { 3  , -6  ,  3  , 0 },
     {-3  ,  3  ,  0  , 0 },
     { 1  ,  0  ,  0  , 0 }
    };

    double a, b, c, d;                  //<b> cubic coefficients vector</b>

    Cubic(double[][] M, double[] G) {
     a = b = c = d;
     for (int k = 0 ; k < 4 ; k++) {  //<b> (a,b,c,d) = M G</b>
         a += M[0][k] * G[k];
         b += M[1][k] * G[k];
         c += M[2][k] * G[k];
         d += M[3][k] * G[k];
     }
    }
```

```
    public double eval(double t) {
     return t * (t * (t * a + b) + c) + d;
    }

    double[][] C = new double[4][4];    //<b> bicubic coefficients matrix</b>
    double[][] T = new double[4][4];    //<b> scratch matrix</b>

    Cubic(double[][] M, double[][] G) {
     for (int i = 0 ; i < 4 ; i++)    //<b> T = G M<sup>T</sup></b>
         for (int j = 0 ; j < 4 ; j++)
          for (int k = 0 ; k < 4 ; k++)
              T[i][j] += G[i][k] * M[j][k];

     for (int i = 0 ; i < 4 ; i++)    //<b> C = M T</b>
         for (int j = 0 ; j < 4 ; j++)
          for (int k = 0 ; k < 4 ; k++)
              C[i][j] += M[i][k] * T[k][j];
    }

    double[] C3 = C[0], C2 = C[1], C1 = C[2], C0 = C[3];

    public double eval(double u, double v) {
     return u * (u * (u * (v * (v * (v * C3[0] + C3[1]) + C3[2]) + C3[3])
                         + (v * (v * (v * C2[0] + C2[1]) + C2[2]) + C2[3]))
                 + (v * (v * (v * C1[0] + C1[1]) + C1[2]) + C1[3]))
                   + (v * (v * (v * C0[0] + C0[1]) + C0[2]) + C0[3]);
    }
}
    public static void main(String[] args) {
        new MainFrame(new bezier(), 600, 600);
    }
}
```