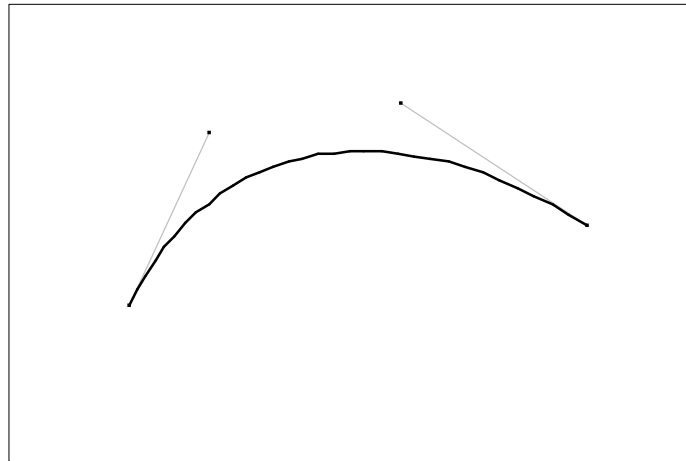


## Bezier Curves

A cubic Bezier curve is a parametric curve formed by using a set of four polynomials to average together a set of four *control points*.

Bezier curves are useful for computer graphics applications because there is a simple and intuitive relationship between the positions of the four control points and the shape of the curve. The first and last control points determine where the curve begins and ends, while the middle two control points determine the overall shape of the curve.



In computer graphics programs users can click and drag the middle pair of control points to reshape the curve, making it relatively easy to produce just the curve shape desired.

Mathematically, points on the curve are computed from the control points  $p_0$ ,  $p_1$ ,  $p_2$ , and  $p_3$  via the formula

$$p(t) = \sum_{k=0}^3 p_k B_k(t)$$

by varying  $t$  from 0 to 1. At  $t = 0$  the curve passes through the first control point  $p_0$ , and at  $t = 1$  the curve passes through the last control point  $p_3$ . The functions  $B_k(t)$  are the four *Bernstein polynomials* of degree three:

$$B_0(t) = (1 - t)^3$$

$$B_1(t) = 3 t (1 - t)^2$$

$$B_2(t) = 3 t^2 (1 - t)$$

$$B_3(t) = t^3$$

The most important characteristic that these polynomials have is that everywhere on the interval  $[0,1]$  the polynomials sum to 1.

$$\begin{aligned}
& B_0(t) + B_1(t) + B_2(t) + B_3(t) \\
&= (1 - t)^3 + 3 t (1 - t)^2 + 3 t^2 (1 - t) + t^3 \\
&= ((1 - t) + t)^3 = 1^3 = 1
\end{aligned}$$

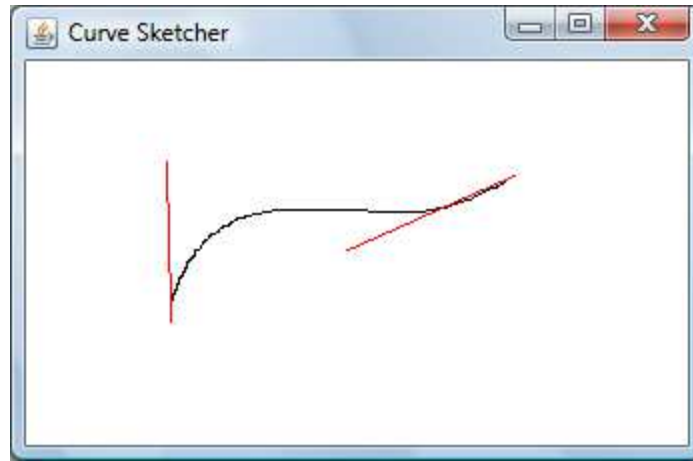
This property allows us to use these functions as weight functions in an averaging process that averages together the four control points as we travel along the curve.

### Converting a Rough Sketch to a Bezier Curve

A nice feature for a computer graphics application is the ability to sketch a freehand curve and have it converted to a smooth Bezier curve. The Curve Sketcher Java application demonstrates part of this process.



Using this application the user can click and drag to draw a freehand curve. As the user drags the mouse the application will lay down a set of points along the curve and draw a rough curve by simply connecting those points in a polygonal line. Once the sample points have been laid down, the application computes a set of control points from the sample points. The control points determine a Bezier curve which is the closest match to the rough sketch drawn by the user. The program then displays a couple of additional lines that connect the two pairs of control points.



Here is the mathematics behind the computation that Curve Sketcher does. We start by labeling the points  $q_i$  along the rough sketch. The  $n$  points on the sketch are numbered from  $i = 0$  to  $i = n-1$ . The first point on the sketch,  $q_0$ , becomes the first control point  $p_0$  and the last point on the sketch,  $q_{n-1}$ , becomes the last control point  $p_3$ . We then want to select locations for the middle control points that produce a curve that comes closest to matching the remaining sample points.

We also set up a crude parameterization of the sketch curve by saying that sketch point  $i$  corresponds to a parameter value of  $t_i = i/(n-1)$ .

We can consider the  $x$  and  $y$  coordinates of the points separately. The  $x$  coordinates of the sketch points give us a set of  $n - 2$  equations:

$$p_x(t_i) = p_{0,x} B_0(t_i) + p_{1,x} B_1(t_i) + p_{2,x} B_2(t_i) + p_{3,x} B_3(t_i) = q_{i,x}$$

Since  $p_0 = q_0$  and  $p_3 = q_{n-1}$  are already determined, we only have to solve these equations for  $p_1$  and  $p_2$ . To emphasize this, we rewrite the equations.

$$q_{i,x} - p_{0,x} B_0(t_i) - p_{3,x} B_3(t_i) = p_{1,x} B_1(t_i) + p_{2,x} B_2(t_i)$$

There are  $n - 2$  such equations, one for each of the interior points on the sketch curve. Since there are many equations and only two unknowns to solve for, the system of equations is an overdetermined system. We will apply a linear algebra technique to this system which is typically used in regression calculations in statistics to draw best-fit curves that depend on relatively few parameters through a large set of data points. We start by converting the  $n - 2$  equations into a single matrix equation.

$$\mathbf{z} = B \mathbf{p}$$

where  $\mathbf{z}$  is a vector with  $n - 2$  components

$$z_i = q_{i,x} - p_{0,x} B_0(t_i) - p_{3,x} B_3(t_i)$$

$\mathbf{p}$  is the vector with components

$$\begin{bmatrix} p_{1,x} \\ p_{2,x} \end{bmatrix}$$

that we seek to solve for and  $B$  is an  $n - 2$  by 2 matrix whose rows are

$$\begin{bmatrix} B_1(t_i) & B_2(t_i) \end{bmatrix}$$

If the  $B$  matrix were square, we would solve this system by computing the inverse of the  $B$  matrix.

$$\mathbf{p} = B^{-1} \mathbf{z}$$

Since  $B$  is not a square matrix, we have to solve this system by using a generalization of the matrix inverse, the *Moore-Penrose pseudoinverse*:

$$\mathbf{p} = (B^T B)^{-1} B^T \mathbf{z}$$

This calculation is guaranteed to produce the closest match curve to the original sketch points.

The Curve Sketcher program isolates this calculation in a class named `BezierMaker`. The `BezierMaker` class contains a single static method

```
static public Point[] makeBezier(ArrayList<Point> points)
```

that carries out the computation discussed above. Given an `ArrayList` containing all of the `Point`s in the user's rough sketch, `makeBezier` does the computation of the best fit control points and then returns those in the form of an array of four `Point` objects.

### Programming Assignment

Modify the Curve Sketcher program to add the following features.

1. Create a `BezierCurve` class that represents individual Bezier curve segments. Give this class a constructor that takes as its parameter the array of four control points returned by the `BezierMaker.makeBezier` method and a `draw(Graphics g)` method that draws the Bezier curve using the `Graphics` object `g`.
2. Add an `ArrayList` of `BezierCurve` objects as a member variable in the `CurvePanel` class. Each time the user draws a new rough sketch curve it will get turned into a `BezierCurve` object and added to the list.
3. The user can modify any of the existing Bezier curves by clicking on or near one of its control points and dragging to reposition it. As the user drags the control point the application should immediately redraw all the curves to give immediate feedback on the user's action.
4. Clicking on a location that is not on or near a control point starts a new rough

sketch curve. As before, the user can drag to draw that rough sketch. When they release the mouse button, the application converts the rough sketch to a Bezier curve, adds the curve to the list of curves, removes the rough sketch, and then redraws the display.

5. Pressing the backspace key removes the most recently drawn Bezier curve.
6. Provide a way to save all the curves to a file and then read them back in from that file later.