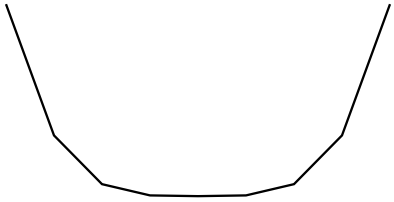
Chapter 6. Curves

So far, the only paths we have learned how to draw in PostScript are sequences of line segments. There is a very simple way to draw arcs of circles as well:

@x y r a b arc@

will build an arc of the circle of radius r, centre (x, y), in the counter-clockwise direction from angle a to b. Angles are measured from the positive x-axis. The similar command @arcn@ constructs paths in the opposite direction. But this is still a very limited repertoire. PostScript allows a third method to build paths which is much more versatile.

The conceptually simplest way to draw even a complicated curve is by drawing a sequence of line segments—that is to say, making a **polygonal approximation** to it—but this usually requires a very large number of segments to be at all acceptable. It also suffers from the handicap that it is not very scalable—that is to say, even if a collection of segments looks smooth at one scale, it may not look good at another. Here, for example, is a portion of the graph of $y = x^4$ drawn with eight linear segments.



The overall shape is not too bad, but the breaks are quite visible. You can certainly improve the quality of the curve by using more segments, but then the number of segments required to satisfy the eye changes with the scale used in representing the curve. The trouble is that the human eye can perceive that the directions (or first derivatives) vary discontinuously in this scheme. This is not at all something to be taken for granted—the more we learn about eyes in nature the more we learn that most features like this are 'hard-wired'. (In contrast, the human eye apparently has trouble perceiving discontinuities in the second derivative.)

In any event, it is better, if possible, to produce smooth curves—at least as smooth as the physical device at hand will allow. PostScript does this by approximating segments of a curve by **Bezier cubic curves**. This allows us to have the tangent direction of the curve vary continuously as well.

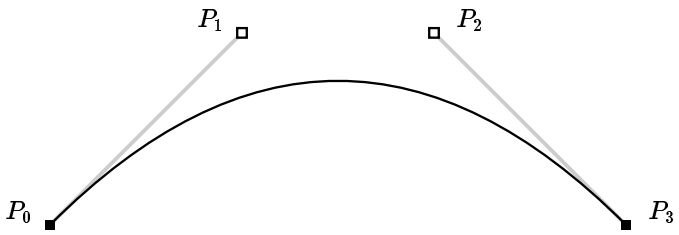
This is the last major component of PostScript you will have to learn. In the rest of this course we shall learn how to manipulate and combine the basic tools we have already been introduced to.

1. Bezier curves

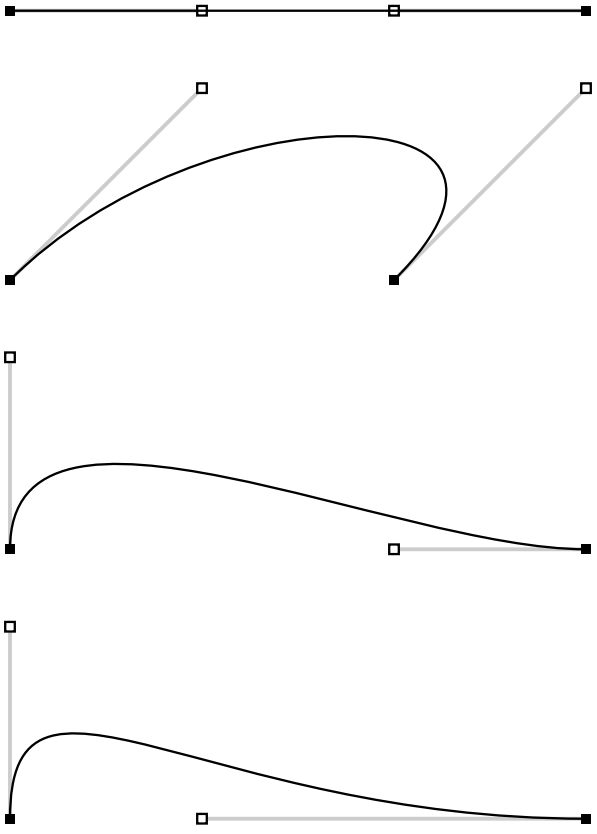
In PostScript, to add a curved path to a path already begun, you put in a command sequence like

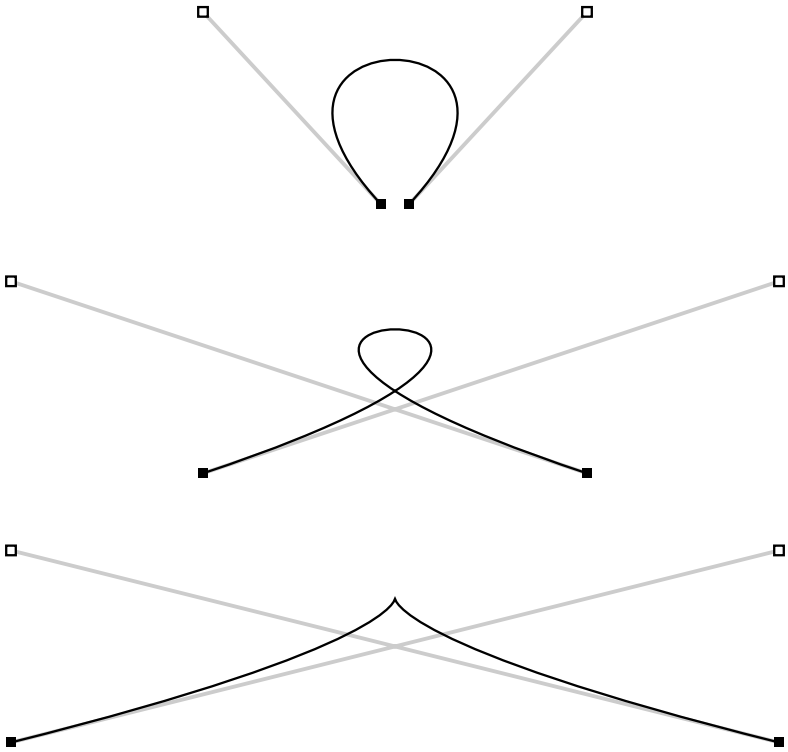
@1 1 2 1 3 0 curveto@

This makes a curve starting at the current point P_0, ending at $P_3 = (3, 0)$, and in between following a path **controlled** by the intermediate points $P_1 = (1, 1)$ and $P_2 = (2, 1)$. If we want to start at $P_0 = (0, 0)$ we would therefore preface this by a command @0 0 moveto@. What we get is this (where the four relevant points are marked):



In these notes I shall usually call P_0 and P_3 the **end points** and P_1 and P_2 the **control points** of the curve. Occasionally I shall just call the lot control points, which is more standard terminology. Even from this single picture you will see that the effect of the control points on the shape of the curve is not so simple. In order to draw curves efficiently and well, we have to understand this matter much better. We shall see later the exact mathematics of what is going on, but right now I shall simply exhibit several examples.





You should be able to see from these examples that the use of control points to specify curves becomes more intuitive with experience. The following facts may give a rough feeling for how things go.

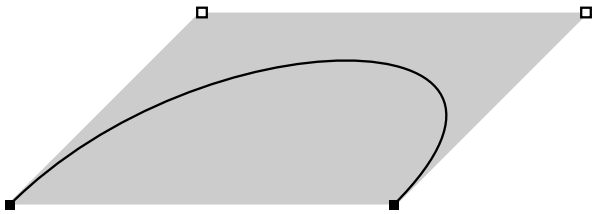
(1) *The path starts at P_0 and ends at P_3.*

(2) *When the curve starts out from P_0 it is heading straight for P_1.*

(3) *Similarly, when it arrives at P_3 it is coming from the direction of P_2.*

(4) *The longer the line from P_0 to P_1, the tighter the curve sticks to that line when it starts out from P_0. Similarly for P_2 and P_3.*

There is another fact that is somewhat less apparent.



(5) *If we wrap up the four points P_i in a quadrilateral box, then the whole curve is contained inside that box.*

The intuitive picture of a Bezier curve conceives of it as a path followed by a particle in motion between certain times. The vector from P_0 to P_1 is proportional to the velocity of the particle as it starts out from P_0, and the vector from P_2 to P_3 is proportional to its velocity when it arrives at P_3. Another way of putting this is that roughly speaking the control points are a convenient way to encode the initial and final velocities in geometric data. This explains properties (2), (3), and (4). Property (5) is implied by the fact that any point on the curve is some kind of weighted average of the four points P_i.

Curves drawn by using control points in this way are called **Bezier curves** after the twentieth century French automobile designer Pierre Bezier who first used them extensively in computer graphics, even though their use in mathematics under the name of **cubic interpolation curves** is much older.

One natural feature of Bezier curves described by control points is that they are stable under arbitrary affine transformations—that is to say that the affine transformation of a Bezier curve is the Bezier curve defined by the affine transformations of its control points. This is often an extremely useful property to keep in mind.

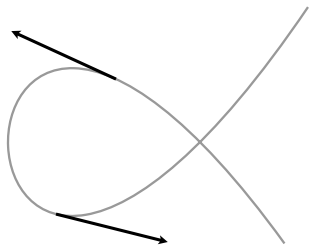
Exercise 1.1. *Write a PostScript procedure @pixelcurve@ with arguments* 4 *arrays* P_0*,* P_1*,* P_2*,* P_3 *of size* 2*, with the effect of drawing the corresponding Bezier curve, including also black pixels of width* $0.05''$ *at each of these points.*

2. How to use Bezier curves

In this section we shall be introduced to a recipe for using Bezier curves to draw very general curves. In the next this recipe will be justified. In order to make the recipe plausible, we shall begin by looking at the problem of how to approximate a given curve by polygons.

The first question we must answer, however, is more fundamental: *How are curves to be described in the first place?* In this course the answer will usually be in terms of a **parametrization**. Recall that a **parametrized curve** is a map from points of the real line to points in the plane—that is to say, to values of t in a selected range we associate points $(x(t), y(t))$ in the plane. It often helps one's intuition to think of the **parameter** t as time, so as time proceeds we move along the curve from one point to another. In this scheme, with a parametrization $P(t)$, the **velocity** at time t is the limit of average velocities over smaller and smaller intervals of time $(t, t+h)$:

$$V(t) = P'(t) = \lim_{h \to 0} \frac{P(t+h) - P(t)}{h} = (x'(t), y'(t)) \ .$$



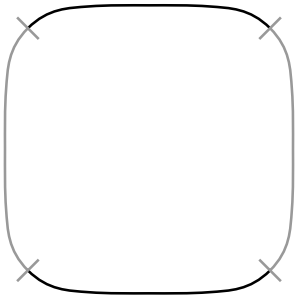
The direction of the velocity vector is tangent to the curve, and its magnitude is determined by the speed of motion along the curve.

Example. The unit circle with centre at the origin has parametrization $t \mapsto (\cos t, \sin t)$.

Example. If $f(x)$ is a function of one variable x, its graph has the parametrization $t \mapsto (t, f(t))$.

Example. There are two common ways to specify a curve in the plane. The first is a parametrization. The second is an equation relating x and y. An example is the oval

$$x^4 + y^4 = 1 \ .$$



This is not the graph of a function, and it has no obvious single parametrization. We can solve the equation $x^4 + y^4 = 1$ to get

$$y = \sqrt[4]{1 - x^4}$$

which gives the top half of our oval, and get the bottom half similarly. Neither half is yet the graph of a good function, however, because both have infinite slope at $x = \pm 1$. We can, however, restrict the range of x away from ± 1, say to $[-\sqrt[4]{1/2}, \sqrt[4]{1/2}]$. We can then turn the curve sideways and now solve for x in terms of y to write the rest as a graph rotated $90°$. To summarize, we can at least express this curve as the union of four separate pieces, each of which we can deal with.

Exercise 2.1. *Find a parametrization of this oval by drawing inside it a circle, and taking as the point corresponding to t the point of intersection of the oval with the ray from the origin at angle t.*

With this understanding of how a curve is given to us, the question we are now confronted with is this:

- *Given a parametrization $t \mapsto P(t)$ of a curve in the plane, how do we draw part of it using Bezier curves?*

If we were to try to draw it using linear segments, the answer would go like this: Suppose we want to draw the part between given values t_0 and t_1 of t. We divide the interval $[t_0, t_1]$ into n smaller intervals $[t_0+ih, t_0+(i+1)h]$, and then draw lines $P(t_0)P(t_0 + h)$, $P(t_0 + h)P(t_0 + 2h)$, $P(t_0 + 2h)P(t_0 + 3h)$, etc. Here $h = (t_1 - t_0)/n$. If we choose n large enough, we expect the series of linear segments to approximate the curve reasonably well.

To use Bezier curves, we will follow the roughly the same plan—chop the curve up into smaller pieces, and on each small piece attempt to approximate the curve by a single Bezier curve. In order to to that, the essential problem we face is this: *Suppose we are given two values of the parameter t, which we may as well assume to be t_0 and t_1, and which we assume not to be too far apart. How do we approximate by a Bezier curve the part of the curve parametrized by the range $[t_0, t_1]$?*

Calculating the end points is no problem. But how to get the two interior control points? Since they have something to do with the directions of the curve at the end points, we expect to use the values of the velocity vector at the endpoints. The exact recipe is this. Start by setting

$$\begin{aligned} P_0 &= (x(t_0), y(t_0)) \\ P_3 &= (x(t_1), y(t_1)) \ . \end{aligned}$$

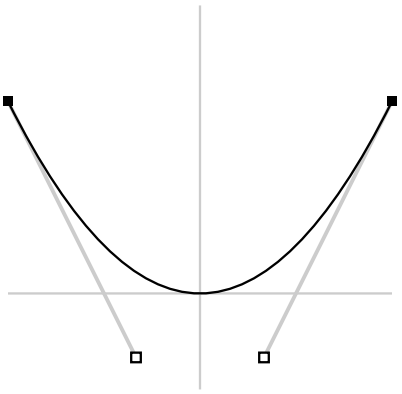
These are the end points of our small Bezier curve. Then set

$$\begin{aligned} \Delta t &= t_1 - t_0 \\ P_1 &= P_0 + (\Delta t/3)P'(t_0) \\ P_2 &= P_3 - (\Delta t/3)P'(t_1) \end{aligned}$$

to get the control points.

Example. Let's draw the graph of the parabola $y = x^2$ for x in $[-1, 1]$. It turns out that a single Bezier curve will make a perfect fit over the whole range. Here the parametrization is $P(t) = (t, t^2)$, $P'(t) = (1, 2t)$.

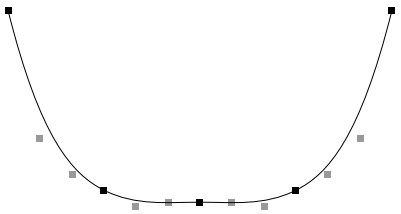
$$\begin{aligned} t_0 &= -1 \\ t_1 &= 1 \\ \Delta t &= 2 \\ P_0 &= (-1, 1) \\ P_1 &= (1, 1) \\ P'(-1) &= (1, -2) \\ P'(1) &= (1, 2) \\ P_1 &= P_0 + (2/3)P'(t_0) \\ &= (-1/3, -1/3) \\ P_2 &= (1/3, -1/3) \end{aligned}$$



Example. Let's draw the graph of $y = x^4$ for $x = -1$ to $x = 1$. Here $P(t) = (t, t^4)$, $P'(t) = (1, 4t^3)$. Divide the graph up into (say) 4 segments $[-1.0, -0.5]$, $[-0.5, 0.0]$, $[0.0, 0.5]$, $[0.5, 1.0]$. We have this table, with the control points interpolated.

x	y	x'	y'
-1.0000	1.0000	1.0	-4.0
-0.8333	0.3333		
-0.6667	0.1458		
-0.5000	0.0625	1.0	-0.5
-0.3333	0.0208		
-0.1667	0.0000		
0.0000	0.0000	1.0	0.0
0.1667	0.0000		
0.3333	0.0208		
0.5000	0.0625	1.0	0.5
0.6667	0.1458		
0.8333	0.3333		
1.0000	1.0000	1.0	4.0

Here is the curve you get, with control points marked. It is almost indistinguishable from the one you get with 16 segments, so the approximation is pretty good. It is perhaps only when you see where the control points lie that you notice the slight rise in the middle.



3. The mathematics of Bezier curves

The mathematical problem we are looking at in drawing good curves in computer graphics is that of approximating an arbitrary parametrized path $t \mapsto (x(t), y(t))$ by a simpler one. If we are approximating a path by line segments, for example, then we are replacing various pieces of the curve between points $P(t_0)$ and $P(t_1)$ by a linearly parametrized path

$$t \mapsto \frac{(t_1 - t)P(t_0) + (t - t_0)P(t_1)}{(t_1 - t_0)}$$

from one point to the other. With Bezier curves, we are asking for a parametrization from one point to the other with the property that its coordinates are cubic polynomials of t (instead of linear). In other words, we are looking for approximations to the coordinates of a parametrization by polynomials of degree three. We expect an approximation of degree three to be much better than a linear one.

The Bezier curve, then is to be a parametrized path $B(t)$ from P_0 to P_3, cubic in the parameter t, and depending in some way on the interior control points P_1 and P_2. Here it is:

$$B(t) = \frac{(t_1 - t)^3 P_0 + 3(t - t_1)^2(t - t_0)P_1 + 3(t_1 - t)(t - t_0)^2 P_2 + (t - t_0)^3 P_3}{(t_1 - t_0)^3} \ .$$

We can rewrite the formula for the linear path as well as that for Bezier path in a somewhat simpler form if we use the **normalized parameter**

$$s = \frac{t - t_0}{t_1 - t_0} \ .$$

This is a linear function of t, and as t goes from t_0 to t_1 it passes from 0 to 1. We also have

$$1 - s = \frac{t_1 - t}{t_1 - t_0} \ .$$

Using s instead of t the formula for the linear path becomes

$$(1 - s)P_0 + sP_1$$

and that for the Bezier path

$$B(s) = (1 - s)^3 P_0 + 3s(1 - s)^2 P_1 + 3s^2(1 - s)P_2 + s^3 P_3 \ .$$

This is somewhat easier to calculate with than the original.

It is simple to verify that

$$B(t_0) = P_0$$
$$B(t_1) = P_3 \ .$$

We can also calculate (term by term)

$$\begin{aligned}(t_1-t_0)^3B'(t) &= -3(t_1-t)^2P_0+3(t_1-t)^2P_1-6(t-t_0)(t_1-t)P_1\\ &\qquad +6(t-t_0)(t_1-t)P_2-3(t-t_0)^2P_2+3(t-t_0)^2P_3\\ B'(t) &= \frac{3(t_1-t)^2(P_1-P_0)+6(T-t_0)(t-t_1)(P_2-P_1)+3(t-t_0)^2(P_3-P_2)}{(t_1-t_0)^3}\\ B'(t_0) &= \frac{3(P_1-P_0)}{t_1-t_0}\\ B'(t_1) &= \frac{3(P_3-P_2)}{t_1-t_0}\end{aligned}$$

These calculations verify our earlier assertions relating the control points to velocity, since we can deduce from them that

$$\begin{aligned}P_1 &= P_0+\frac{t_1-t_0}{3}\,B'(t_0)\\ P_2 &= P_3-\frac{t_1-t_0}{3}\,B'(t_1)\ .\end{aligned}$$

Exercise 3.1. *A* **quadratic path** *defined by three points P_0, P_1, and P_2 is defined by*

$$Q(s)=(1-s)^2P_0+2s(1-s)P_1+s^2P_2\ .$$

What is $Q(0)$? $Q(1)$? $Q'(0)$? $Q'(1)$? Why aren't these used instead of Bezier paths?

4. Mathematical motivation

In using linear or Bezier paths to do computer graphics, we are concerned with the problem of approximating the coordinate functions of an arbitrary path by polynomials of degree one or three. Considering each coordinate separately, we are led to try to approximate an arbitrary function of one variable by a polynomial of degree one or three.

The basic difference between linear approximations and cubic approximations lies in the following facts:

- *If t_0, t_1, y_0, and y_1 are given then there exists a unique linear function $f(t)$ such that*

$$\begin{aligned}f(t_0) &= y_0\\ f(t_1) &= y_1\end{aligned}$$

- *Given t_0, t_1, y_0, y_1, v_0, v_1, there exists a unique cubic polynomial $f(t)$ such that*

$$\begin{aligned}f(t_0) &= y_0\\ f'(t_0) &= v_0\\ f(t_1) &= y_1\\ f'(t_1) &= v_1\ .\end{aligned}$$

Roughly speaking, with linear approximations we can only get the location of end points exactly, but with cubic approximation we can get directions exact as well.

We shall prove here the assertion about cubic functions. If

$$f(t)=a_0+a_1t+a_2t^2+a_3t^3$$

then the conditions on $P(t)$ set up four equations in the four unknowns a_i which turn out to have a unique solution (assuming of course that $t_0 \neq t_1$. Here are the equations:

$$\begin{array}{ccccccccc} a_0 & + & a_1 t_0 & + & a_2 t_0^2 & + & a_3 t_0^3 & = & y_0 \\ & & a_1 & + & 2a_2 t_0 & + & 3a_3 t_0^2 & = & v_0 \\ a_0 & + & a_1 t_1 & + & a_2 t_1^2 & + & a_3 t_1^3 & = & y_1 \\ & & a_1 & + & 2a_2 t_1 & + & 3a_3 t_1^2 & = & v_1 \end{array}$$

The coefficient matrix is

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ & 1 & 2t_0 & 3t_0^2 \\ 1 & t_1 & t_1^2 & t_1^3 \\ & 1 & 2t_1 & 3t_1^2 \end{bmatrix}$$

One thing that simplifies the mathematics quite a bit is to **normalize** the parameter variable t, so that instead of going from t_0 to t_1 it goes from 0 to 1. We can do this by defining a new parameter variable

$$s = \frac{t - t_0}{t_1 - t_0} \, .$$

Note that s takes values 0 and 1 at the ends $t = t_0$ and $t = t_1$. Changing the parameter variable in this way doesn't affect the curve traversed. It simplifies the assertion above.

- *Given y_0, y_1, v_0, v_1, there exists a unique cubic polynomial $f(t)$ such that*

$$\begin{aligned} f(0) &= y_0 \\ f'(0) &= v_0 \\ f(1) &= y_1 \\ f'(1) &= v_1 \, . \end{aligned}$$

The coefficient matrix is now

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{matrix}$$

I leave it to you as an exercise to check now by direct row reduction that the determinant is not zero, which implies that the system of four equations in four unknowns has a unique solution. Of course we know from the formula used in the previous section what the explicit formula is, but the reasoning in this section shows that this formula is the well defined answer to a natural mathematical question.

Exercise 4.1. *What is the determinant of this 4×4 matrix?*

Exercise 4.2. *Find the coefficients a_i explicitly.*

If we put together the results of this section with those of the previous one, we have this useful characterization:

- *Given two parameter values t_0 and t_1, and the four points P_0, P_1, P_2, P_3, the Bezier path $B(t)$ is the unique path $(x(t), y(t))$ with these properties:*
  - *(1) The coordinates are cubic as a function of t;*
  - *(2) $B(t_0) = P_0$, $B(t_1) = P_3$;*
  - *(3) $B'(t_0) = 3(P_1 - P_0)/\Delta t$ and $B'(t_1) = 3(P_3 - P_2)/\Delta t$, where $\Delta t = t_1 - t_0$.*

The new idea here is uniqueness. Roughly, the idea is that four control points require eight numbers, and that the cubic coordinate functions also require eight numbers.

5. Weighted averages

The formula for a linear path from P_0 to P_1 is

$$\begin{aligned} P(t) &= (1-t)P_0 + tP_1 \\ &= P_0 + t(P_1 - P_0) \ . \end{aligned}$$

We have observed before that $P_0 + t(P_1 - P_0)$ may be seen as the point t of the way from P_0 to P_1. When $t = 0$ this gives P_0, and when $t = 1$ it gives P_1. There is also an intuitive way to understand the first formula that we have not considered so far.

Let's begin with some examples. With $t = 1/2$ we get the mid-point of the segment

$$\frac{P_0 + P_1}{2}$$

which is the average of the two. With $t = 1/3$ we get the point one third of the way

$$\frac{2P_0 + P_1}{3}$$

which is to say that it is a **weighted combination** of the endpoints with P_0 given twice as much weight as P_1.

There is a similar way to understand the formula for Bezier curves. It is implicit in what was said in the last section that the control points P_i determine a cubic path from P_0 to P_1

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

and that this path is a parametrization of the Bezier curve with these control points. In other words, $B(t)$ is a weighted combination of the control points. It is actually an average—which is to say, that the sum of the coefficients is 1:

$$(1-t)^3 + 3t(1-t)^2 + 3t^2(1-t) + t^3 = \big((1-t) + t\big)^3 = 1$$

by the binomial theorem for $n = 3$, which asserts that

$$(a+b)^3 = a^3 + 3a^2 b + 3ab^2 + b^3 \ .$$

Since all the coefficients in our expression are non-negative for $0 \leq t \leq 1$, $B(t)$ will lie inside the quadrilateral wrapped by the control points, although we shall not see details here.

Exercise 5.1. *Write the simplest procedure you can with these properties: • it has two arguments x_0 and x_1 and • it draws the graph of $y = x^2$ between x_0 and x_1 with a single Bezier curve.*

Exercise 5.2. *Draw $y = x^5$ between $x = -1$ and $x = 1$ in the same way we drew $y = x^4$ earlier.*

Exercise 5.3. *We can construct polynomial analogues of Bezier cubic functions of any positive degree. In degree one we have the linear function*

$$(1-s)P_0 + sP_1 \ ,$$

in degree two we have the quadratic functions mentioned earlier. In degree n we have the polynomial

$$P_n(s) = c_0(1-s)^n y_0 + c_1 s(1-s)^{n-1} y_1 + \cdots + c_n s^n y_n$$

where the coefficients c_i make up the n-th row of Pascal's triangle

$$\begin{array}{ccccccc} & & & 1 & & & \\ & & 1 & & 1 & & \\ & 1 & & 2 & & 1 & \\ 1 & & 3 & & 3 & & 1 \\ & & & \dots & & & \end{array}$$

Find a formula for the derivative of $P_n(t)$ in terms of some $P_{n-1}(t)$.

These polynomials are called **Bernstein polynomials**.

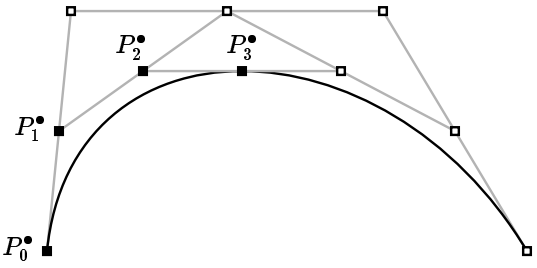
6. How the computer draws Bezier curves

In this section we shall see how the computer goes about drawing a Bezier curve. It turns out to be an extremely efficient process. First of all, a computer 'thinks of' any path as a succession of small points (pixels) on the particular device it is dealing with. This is somewhat easier to see on a computer screen, certainly if you use a magnifying glass, but remains true even of the highest resolution printers. So in order to draw something it just has to decide which pixels to color. It does this by an elegant recursive procedure, something akin to the following way to draw a straight line segment: (1) Color the pixels at each end. (2) Color the pixel at the middle. (3) This divides the segment into two halves. Apply steps (2) and (3) again to each of the halves. And so on, until the segments you are looking at are so small that they cannot be distinguished from pixels.

The analogous construction for Bezier curves goes like this:

Start with a Bezier curve with control points P_0, P_1, P_2, P_3. Perform the following construction. Set

$$\begin{aligned} P_1^\bullet &= \text{ the median between } P_0 \text{ and } P_1 \\ P_2^{\bullet\bullet} &= \text{ the median between } P_2 \text{ and } P_3 \\ Q &= \text{ the median between } P_1 \text{ and } P_2 \\ P_2^\bullet &= \text{ the median between } P_1^\bullet \text{ and } Q \\ P_1^{\bullet\bullet} &= \text{ the median between } Q \text{ and } P_2^{\bullet\bullet} \\ P_3^\bullet &= \text{ the median between } P_2^\bullet \text{ and } P_1^{\bullet\bullet} \\ P_0^{\bullet\bullet} &= P_3^\bullet \end{aligned}$$



The point $P_3^\bullet = P_0^{\bullet\bullet}$ turns out to lie on the Bezier curve determined by the original points P_i, at approximately the halfway point. The Bezier curve can now be split into two halves, each of which is itself a Bezier cubic, and the control points of the two new curves are among those constructed above. The points $P_0^\bullet = P_0$, $P_1^\bullet$, $P_2^\bullet$, and $P_3^\bullet$ (the black squares above) are the control points for the first half (and in particular $P_3^\bullet$ lies on the curve); similarly, $P_0^{\bullet\bullet}$, $P_1^{\bullet\bullet}$, $P_2^{\bullet\bullet}$, $P_3^{\bullet\bullet} = P_3$ are control points for the second half. If we keep subdividing in this way we

get a sequence of midpoints for the smaller segments (actually a kind of branched list), and to draw the curve we just plot these points after the curve has been subdivided far enough. This sort of subdivision can be done very rapidly by a computer, since dividing by two is a one-step operation in base 2 calculations, and in fact drawing the pixels to go on a straight line is not a great deal faster.

Exercise 6.1. *Draw the figure above with PostScript. Label by hand the remaining points $P^\bullet$ and Q in your figure.*

Exercise 6.2. *The point $P_1^\bullet$ is $(1/2)P_0 + (1/2)P_1$. Find similar expressions for all the constructed points in terms of the original four.*

Exercise 6.3. *The purpose of this exercise is to prove that each half of a Bezier curve is also a Bezier curve. Let*

$$P(t) = \frac{(t_1 - t)^3 P_0 + 3(t - t_0)(t_1 - t)^2 P_1 + 3(t - t_0)^2(t_1 - t)P_2 + (t - t_0)^3 P_3}{(t_1 - t_0)^3} \ .$$

The point is to verify that this formula agrees with the geometrical process described above. Let $P_1^\bullet$ etc. be the points defined just above. The first half of the Bezier curve we started with is a cubic curve with initial parameter t_0 and final parameter $t_{1/2}$. Let $\Delta t = t_{1/2} - t_0 = (1/2)(t_1 - t_0)$. Verify that

$$\begin{aligned} P(t_0) &= P_0 \quad \textit{(trivial)} \\ P'(t_0) &= (\Delta t/3)(P_1^\bullet - P_0) \quad \textit{(almost trivial)} \\ P(t_{1/2}) &= P_3^\bullet \\ P'(t_{1/2}) &= (\Delta t/3)(P_3^\bullet - P_2^\bullet) \ . \end{aligned}$$

These equations, by the earlier characterization of control points in terms of derivatives, guarantee that the first half of the original Bezier path is a Bezier path with control points P_0, $P_1^\bullet$, $P_2^\bullet$, $P_3^\bullet$.