



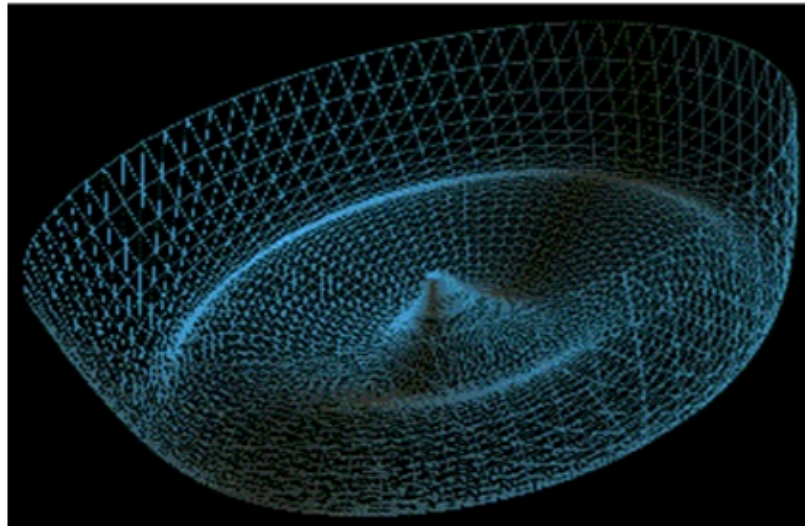
Computer Graphics Using OpenGL Hill

Chapter 11: Curves and Surfaces

■ Curve and Surface Design

■ Goals

- ◆ To develop tools for representing and designing curves.
- ◆ To determine key properties of curves, such as smoothness.
- ◆ To develop the mathematical properties of Bezier and B-spline curves
- ◆ To develop tools to design Bezier, B-Splines, and NURBS surface patches.





Curve and Surface Design

- ♦ Certain shapes are of particular interest to us because they are aesthetically pleasing to behold or they represent a shape found in nature.
- ♦ Other shapes are computer generated with some analysis program as the best possible for the particular job, such as a sweep of an airplane wing to give maximum lift off.
- ♦ Still other shapes, such as a car's fender, are designed by a person, on the basis of a complex combination of engineering utility, ease of manufacturing, and an intuitive sense of what is appealing to the customer



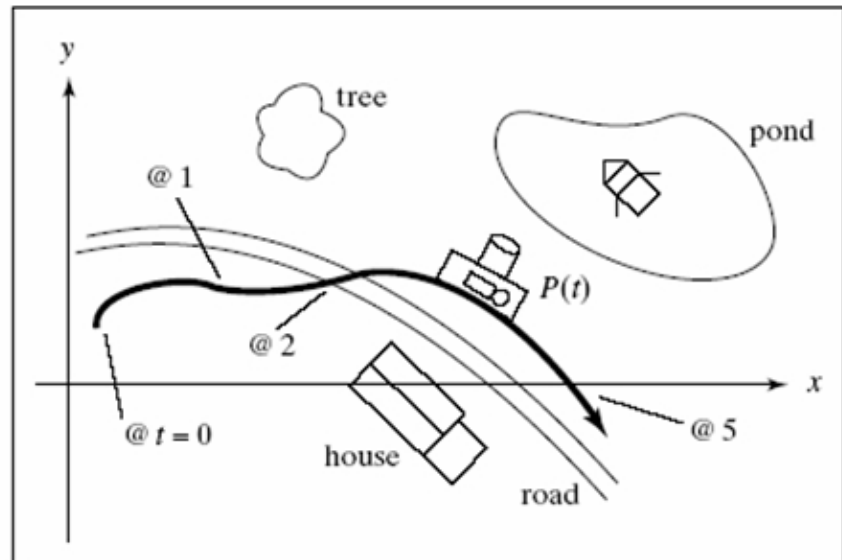
Parametric Curves as Trajectories

- An important application of representing curves parametrically arises when describing a path that an object takes as it moves over time.
- For instance, in designing an animation, the trajectory of the camera through the scene must be specified at each instant.

Parametric Curves as Trajectories

- ♦ The figure below shows a camera at $\mathbf{P}(t)$ at time t , moving through a scene.
- ♦ The designer chooses when to take the snapshots of the scene, in this case at $t = 0.1$ and $t = 0.2$, etc. The view of the camera also needs to be determined as each instant.
- ♦ $\mathbf{P}(t)$ is usually 2D or 3D and contains two parts; one, $F(t)$, that describes the x motion and one, $G(t)$, that describes the y motion. Therefore $\mathbf{P}(t) = (F(t), G(t))$. Often $\mathbf{P}(t)$ is 3 dimensional: $(\mathbf{P}(t) = (F(t), G(t), H(t)))$

FIGURE 11.1 Specifying the path of a camera in an animation.



Smooth Progression of Animation

- ♦ In order to make sure the animation runs smoothly, the designer must impose certain conditions on the velocity $P'(t)$ that we shall consider.
- ♦ In the previous figure, other images may be moving as well, a car may be going down the road, the boat may move in the pond, or someone may come out of the house.
- ♦ The movement of each of these objects must be described by specifying appropriate parametric functions $\mathbf{F}(t), \mathbf{G}(t)$, etc.

3D World Animation

- Animations usually take place in a 3D world, of course, where the camera moves along a 3D line path $P(t) = (x(t), y(t), z(t))$, and each snapshot is formed by projecting the scene onto the film of the camera.

Example: The Ellipse

- ◆ See figure 3.69 (on Hill pg. 121), which shows an ellipse as having parametric form:
- ◆ $F(t) = W \cos(t)$, $G(t) = H \sin(t)$

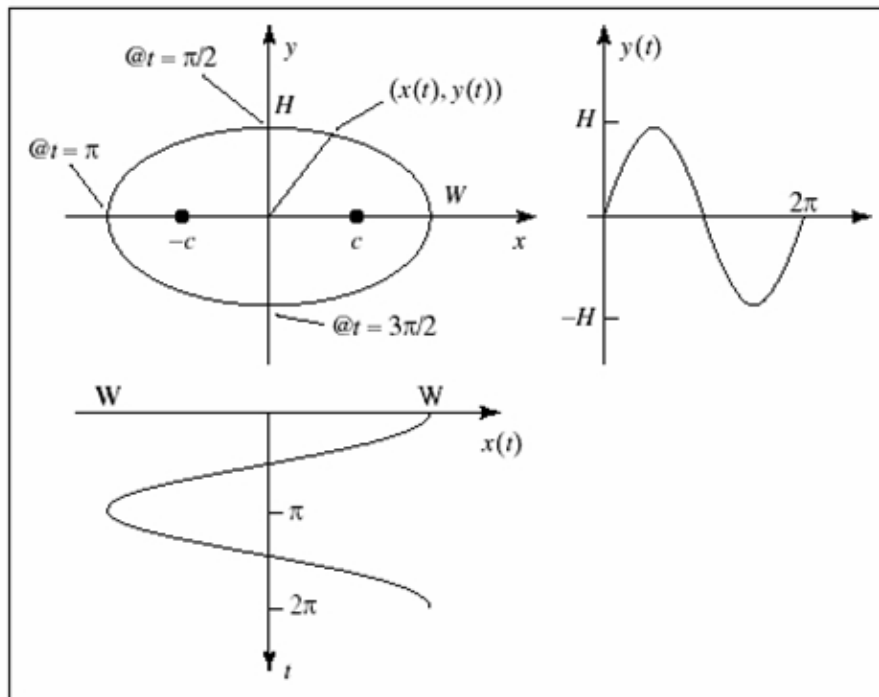


FIGURE 3.69 An ellipse described parametrically.

Smoothness of Motion

- ♦ Suppose the parameter t indicates the passage of time and the point $\mathbf{P}(t)$ moves along the curve as t increases.
- ♦ It is then natural to ask what the velocity of $\mathbf{P}(t)$ is along the curve at each instant.
- ♦ The **velocity** $\mathbf{v}(t)$ is a **vector(vectors again, of course)** that describes the speed and direction of $\mathbf{P}(t)$ as it traverses the curve.
- ♦ From elementary calculus, we have: $\mathbf{P}(t) = (F'(t), G'(t))$

Smoothness of Motion: Speed

- For example, at any time, the ellipse given by $P(t) = (W \cos(t), H \sin(t))$ has the velocity $\mathbf{v}(t) = (-W \sin(t), H \cos(t))$.
- The magnitude of $\mathbf{v}(t)$, denoted $|\mathbf{v}(t)|$ is often called the *speed* at t .

Speed

- The figure below shows the velocity at several points along the ellipse.
- Both the magnitude and direction of v change as t varies.
- Where is the speed greatest?

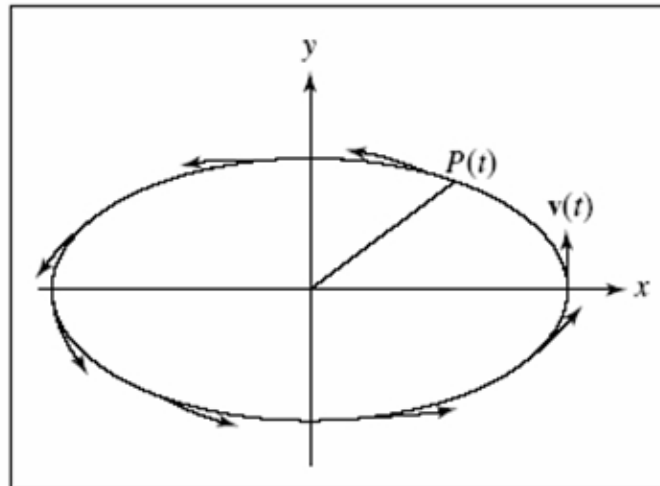


FIGURE 11.2 The velocity of $P(t)$ along a curve.

The Normal Direction

- The **normal direction** to a curve may also be found at each point.
- The normal is defined as the direction perpendicular to the tangent line at the point of interest.
- Thus, if the tangent line has direction $\mathbf{v}(t_0)$ at t_0 , then, as before, the normal direction at t_0 is any multiple of the vector:

$$n(t_0) = \mathbf{v}_\perp(t_0) = \left(\frac{-dG(t)}{dt}, \frac{dF(t)}{dt} \right) \Big|_{t=t_0}$$

Ellipse

- The ellipse above, for example, has normal vector $(-H \cos(t), -W \sin(t))$ or any multiple thereof.
- In the special case of a circle, $\mathbf{n}(t)$ is a multiple of $P(t)$ itself, so the normal points in the same direction as the radius vector from $\mathbf{0}$ to the point $P(t)$.
- (Note that this is *not* the case for an arbitrary ellipse.)

Camera Movement Issues

When a curve $P(t)$ is used to describe how an object such as a camera moves in time, these issue arises:

- ♦ Does the object move “jerkily” or smoothly along its trajectory?
- ♦ The object can move in an infinite variety of ways along a given path, starting, stopping, accelerating, temporarily retracing a portion of the curve, etc.
- ♦ If we are only drawing the curve this motion is irrelevant: all trajectories have the same picture.
- ♦ But it’s a very different story when the parametric function represents camera motion in an animation: the details of movement along the trajectory are recorded in the sequence of snapshots taken.
- ♦ Abrupt accelerations or jumps have a distinctive visual effect which are normally undesirable.

Moving along an Elliptical Trajectory

- As an example, consider a camera, “Bob”, that moves along an elliptical trajectory, but with an altered parametric representation: at $t = a$ the speed of motion suddenly increases by a factor of

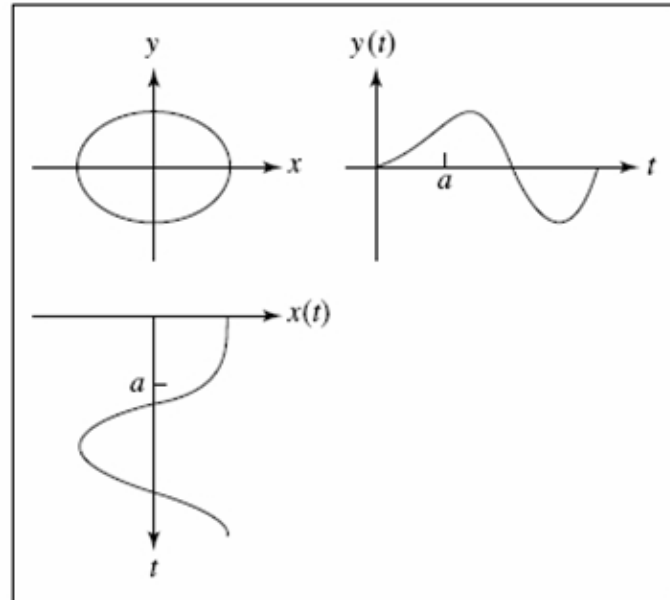
3:

$$P(t) = (x(t), y(t)) = \begin{cases} (W \cos(t), H \sin(t)) & \text{for } 0 < t < a \\ (W \cos(3t - 2a), H \sin(3t - 2a)) & \text{for } a < t < \frac{2(\pi + a)}{3} \end{cases}$$

Trajectory with Sudden Shift Change

- ♦ The situation is shown in Figure 11.3. The trajectory is still elliptical, but at $t = a$ both $x(t)$ and $y(t)$ start to oscillate three times faster, so there is a discontinuity in their derivatives at $t = a$.
- ♦ It's as if the time axis were suddenly compressed by a factor of three starting at this instant.

FIGURE 11.3 A trajectory with a sudden shift in speed of the motion described.



Bob's Velocity Vector

- Bob's velocity vector at $t = a^-$ ("just before" $t = a$) and at $t = a^+$ ("just after" $t = a$) are respectively:

$$P'(a^-) = (-W \sin(a), H \cos(a))$$

and

$$P'(a^+) = (-3W \sin(3a), 3H \cos(3a))$$

so the *direction* of the velocity is the same before and after the shift, but the *magnitude* of the velocity (the speed) suddenly jumps by a factor of 3.

- The parametric curve is unaffected by this jump, but the nature of the motion of the point $P(t)$ along the curve is dramatically different before and after $t = a$.



Smoothness of Curves

- As we study the smoothness of curves and of camera motion, it will be handy to have notation that describes different kinds of continuity of derivatives of a curve. We will be interested in two kinds of smoothness, often called **parametric continuity** and **geometric continuity**.

Parametric Continuity (C_k -continuity or “ k -smoothness”).

- ◆ We say a curve $P(t)$ has k -th order parametric continuity everywhere in the t -interval $[a, b]$ if all derivatives of the curve, up through the k -th, exist and are continuous at all points inside $[a, b]$.
- ◆ To express this briefly we say **$P()$ is k -smooth in $[a, b]$**

The Ellipse (Bob)

- As an example, the ellipse (Bob) in Figure 11.4 (see Hill pg. 602) is 0-smooth everywhere, since the function $P(t)$ itself is continuous everywhere. But it is not 1-smooth everywhere because there is a discontinuity in $P'(t)$ at $t = a$. (It is 1-smooth everywhere except at $t = a$.)



To Avoid Jerky Animations

- To avoid jerky animations we usually insist that camera motion be 1-smooth.
- A curve with a continuous velocity and acceleration is 2-smooth.
- Note that a k -smooth function $[a, b]$ is necessarily $(k-1)$ -smooth, but
- A k -smooth function may or may not also be $(k+1)$ -smooth.

Geometric Continuity (G_k-continuity).

- ♦ This is a more relaxed form of smoothness that describes the visual smoothness of a curve rather than the smoothness of motion along a curve.
- ♦ Basically it requires that the derivative vector have a continuous direction, even though it is allowed to have discontinuities in speed.
- ♦ For example, the ellipse "Bob" in Equation 11.4 is G₁-continuous because its velocity has a continuous direction everywhere. We will call a function that is G_k-continuous in [a,b] "k-continuous in [a,b]".
- ♦ For instance at $t = a$ we know $\mathbf{P}'(a+) = 3 \mathbf{P}'(a-)$.

Geometric Continuity (G_k -continuity).

- The example in the equation below suggests that it's the strange parametrization that causes 1-smoothness to fail and yet G_1 -continuity to succeed, and this is true. We explore the more formal definition of G_k -continuity below. It is based on the existence of a re-parametrization that removes the offending discontinuities.

$$P(t) = (x(t), y(t)) =$$

$$\begin{cases} (W \cos(t), H \sin(t)) & \text{for } 0 < t < a \\ (W \cos(3t - 2a), H \sin(3t - 2a)) & \text{for } a < t < \frac{2(\pi + a)}{3} \end{cases}$$





Curves by means of Polynomials

- ♦ Polynomials are fundamental mathematical objects and are frequently used in computer graphics because they are well behaved and efficient to compute.
- ♦ We will focus on particular forms of polynomials in the rest of the chapter: here we examine the interplay between implicit forms and parametric forms that are simple polynomials.

Definition of Polynomials

- **First a reminder:** An ***L*-th-degree polynomial** in ***t*** is a function given by

$$a_0 + a_1 t + a_2 t^2 + \cdots + a_L t^L$$

where the constants are its **coefficients**, each of which is associated with one of the powers of ***t***. The **degree** of the polynomial is the highest power to which ***t*** is raised. For this to be ***L*-th degree**, we insist that ***a_L*** not be 0. The **order** is the number of coefficients in the polynomial (***L*+1** here). It is always one greater than the degree.



Polynomial Curves of Degree 1.

- **Polynomial Curves of Degree 1.**

- ◆ We have already examined **linear** polynomials and know that a linear parametric form for $x(t)$ and $y(t)$ yields a **straight line**, and the corresponding implicit form is linear in x and y .

Polynomial Curves of Degree 2.

◆ Polynomial Curves of Degree 2.

◆ We naturally ask what curve shapes are attainable using quadratic polynomials for the $x(t)$ and $y(t)$:

$$x(t) = at^2 + 2bt + c$$

$$y(t) = dt^2 + 2et + f$$

where a , b , etc. are constants.

Always a parabola

- The answer is simple (but perhaps disappointing): this curve is always a **parabola**, for any choice of constants $a, b, \dots, f$.
- So there is no way to generate an ellipse or hyperbola using this form.

Implicit forms of degree two:

- Going the other way we look at quadratic implicit forms: that is, polynomials of degree 2 in both x and y .
- Recall from analytic geometry that the general second degree implicit form is given by:

$$F(x, y) = Ax^2 + 2Bxy + Cy^2 + Dx + Ey + F$$

for constants $A, \dots, F$, and that the shape of the curve $F(x, y) = 0$ so described is a conic section (as long as it is not degenerate).

Types of curves

- Which conic is represented depends on the value of the **discriminant**, $AC-B^2$:

if $AC-B^2 > 0$ it is an **ellipse**

if $AC-B^2 = 0$ it is a **parabola**

if $AC-B^2 < 0$ it is a **hyperbola**

Implicit functions for Curves

$$x^2 + xy + y^2 - 1$$

- the implicit function for an ellipse: $(AC-B^2:= 0.5)$

$$x^2 + 2xy + y^2 + 3x - 6y - 7$$

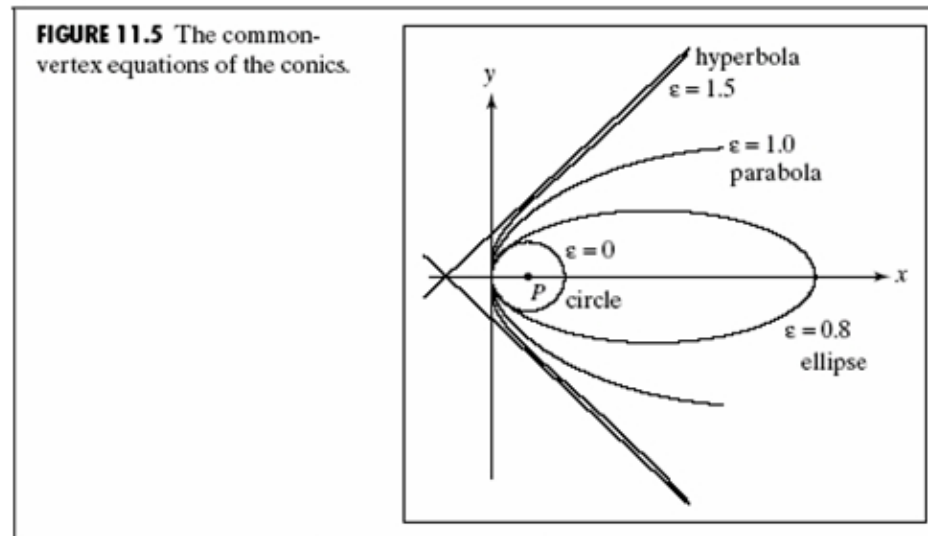
- is the implicit function for a parabola: $(AC-B^2:= 0)$

$$x^2 + 4xy + 2y^2 - 4x + y - 3$$

- is the implicit function for a hyperbola: $(AC-B^2:= -1)$

Eccentricity

- ♦ The curve $y^2 = 2p x - (1-e)^2 x^2$, passes through $(0,0)$ and has a size proportional to constant p . The conic that it describes depends on the value of the **eccentricity** e , as suggested in Figure 11.5.



Polynomial Curves of Degree Three and Higher.

- ♦ We have seen that the situation is well understood if we restrict ourselves to curves parametrized with first and second degree polynomials.
- ♦ Things get more complicated when the polynomials are of higher degree, however. Sederberg et al [sederberg85] show that it is always possible to find an implicit form given polynomial functions for $x(t)$ and $y(t)$, but that a parametric form can in general be found given an implicit form that is of degree one or two.



Polynomial Curves of Degree Three and Higher.

- Our main workhorse below with Bezier and B-spline curves will be the cubic polynomial, and it will be seen to provide a powerful approach to curve design.
- The methods won't start with an implicit form and try to parametrize it. Rather they will start with a collection of “control points” laid down by the designer, allow a specific algorithm to generate points along the curve, and accept the curve regardless of the implicit function it satisfies.
- In many ways this is a more natural approach to curve design than a purely mathematical one.

Polynomial Curves of Degree Three and Higher.

- Before leaving this analytical approach to curve design, however, we preview briefly the class of **rational polynomial** functions, which we examine later. (They are the basis for NURBS as we see later.) This first look shows how the definition of a few *points* can determine a curve shape. It also produces an important result: the conic sections can be represented *exactly* by a ratio of two quadratic polynomials.

Rational Parametric Forms:

- Consider parametrizations where $x(\cdot)$ and $y(\cdot)$ are each defined as a **ratio** of two polynomials. The linear case is explored in the exercises.
- We look here at the quadratic polynomial case, and focus on the particular parametric form:

$$P(t) = \frac{P_0(1-t)^2 + 2wP_1t(1-t) + P_2t^2}{(1-t)^2 + 2wt(1-t) + t^2}$$

Rational Parametric Forms:

$$P(t) = \frac{P_0(1-t)^2 + 2wP_1t(1-t) + P_2t^2}{(1-t)^2 + 2wt(1-t) + t^2}$$

where P_0 , P_1 , and P_2 are any three points in the plane.

- They are called **control points** in this context, as they control the shape of the curve. Equation 11.14 is actually two equations, of course, being shorthand for

$$(x(t), y(t)) = \left(\frac{x_0(1-t)^2 + 2x_1wt(1-t) + x_2t^2}{(1-t)^2 + 2wt(1-t) + t^2}, \frac{y_0(1-t)^2 + 2y_1wt(1-t) + y_2t^2}{(1-t)^2 + 2wt(1-t) + t^2} \right)$$

- where x_0 and y_0 are the components for P_0 , and similarly for the other two points

Analyzing the Equation

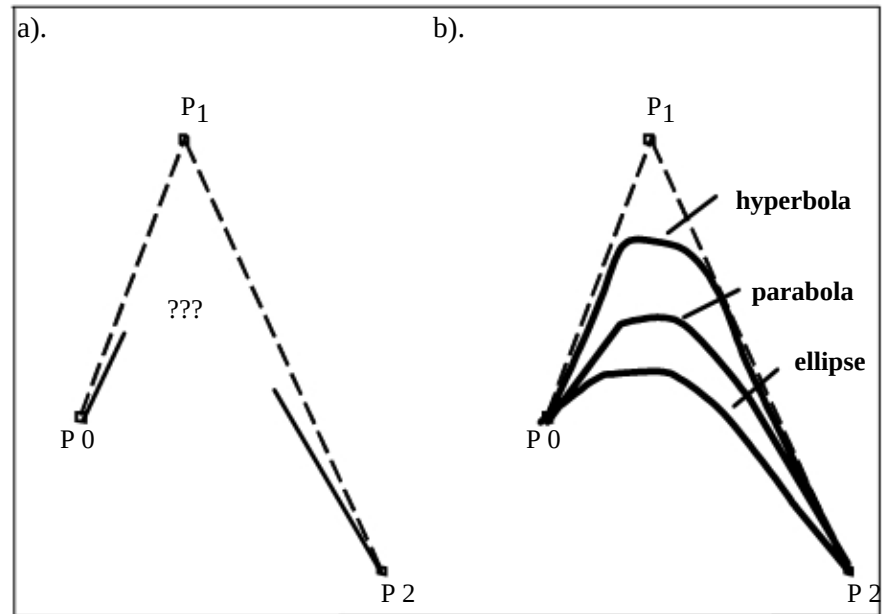
- ♦ The coefficients of the quadratic polynomials in the numerators are the components of the control points.
- ♦ The denominator polynomials are the same for $x(\cdot)$ and $y(\cdot)$ and are also quadratic but do not depend on the points.
- ♦ They do depend on a “weight” parameter w , however.
- ♦ Note that $P(t)$ is a linear combination of control points.
- ♦ In order to make sense as a point it must be an affine combination of these points.

- 
- Note that if we evaluate this form at $t=0$ the right hand side collapses simply to (x_0, y_0) , so this curve passes through, or **interpolates**, the point P_0 .
 - ◆ Similarly at $t=1$ it passes through P_2 .
 - ◆ For t in between $P(t)$ depends on all three points in a complicated way.

Generating conics with rational quadratics.

- Figure 11.6a shows the three control points and shows how the curve emerges from P_0 as t increases from 0 and ends up at P_2 as t approaches 1. It asks what the shape in between is.
- Part B shows the answer: the curve is one of the conic sections, and the type depends on the values of w :

if $w < 1$ it is an **ellipse**
if $w = 1$ it is a **parabola**
if $w > 1$ it is a **hyperbola**





Generating the Conic Sections Parametrically

- So we have a way of generating the conic sections parametrically, without recourse to the trigonometric functions needed above. The exercises show how to generate the “other half” of each of these curves and also how to generate a circle using this parametric form.

Control Points

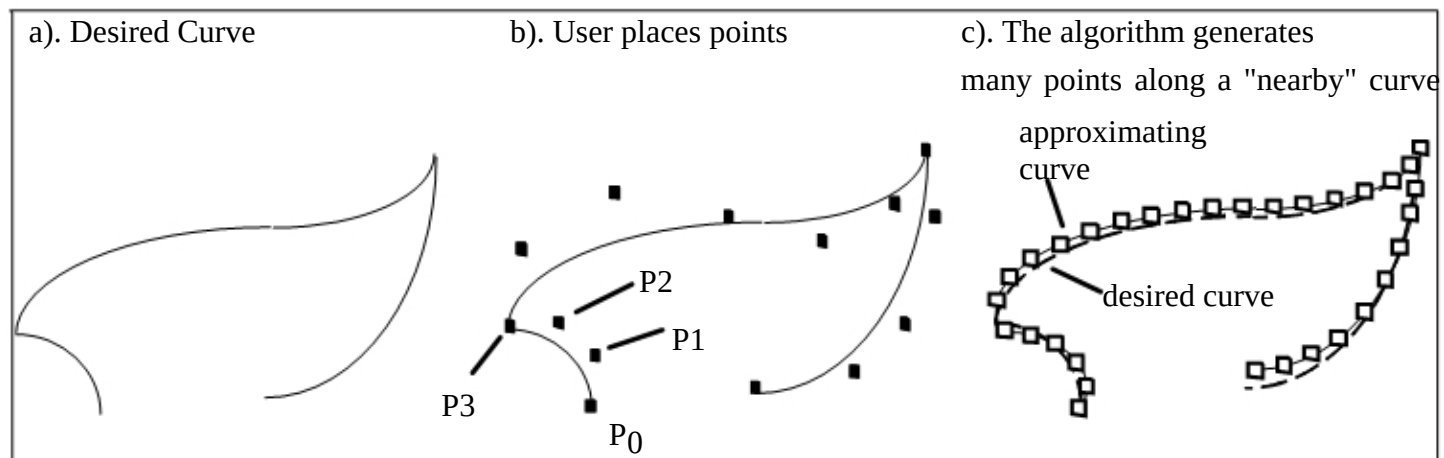
- ♦ The curves considered so far have been based on relatively simple mathematical formulas.
- ♦ Now we want to broaden the task and develop more complex curves that serve a certain purpose.
- ♦ In particular we want to develop tools that allow a designer to fashion a large variety of shapes by simply specifying a *small collection* of “control points”. *Non-data intensive.*

A curve design scenario.

- ♦ Assume, for example, that the designer wishes to create a computer representation of the curve of following image.
- ♦ It might be a part of an emerging design for a car fender, a turbine blade, or the casing of an electric drill.
- ♦ Or it might be the trajectory of a camera as it sweeps through a scene taking snapshots.

A curve design scenario.

- ♦ The goal is to capture the shape of this curve in a form that permits it to be reproduced at will, adjusted in shape and size as desired, sent to a machine for automatic cutting or molding, etc.
- ♦ There is most likely no simple formula that matches it exactly.

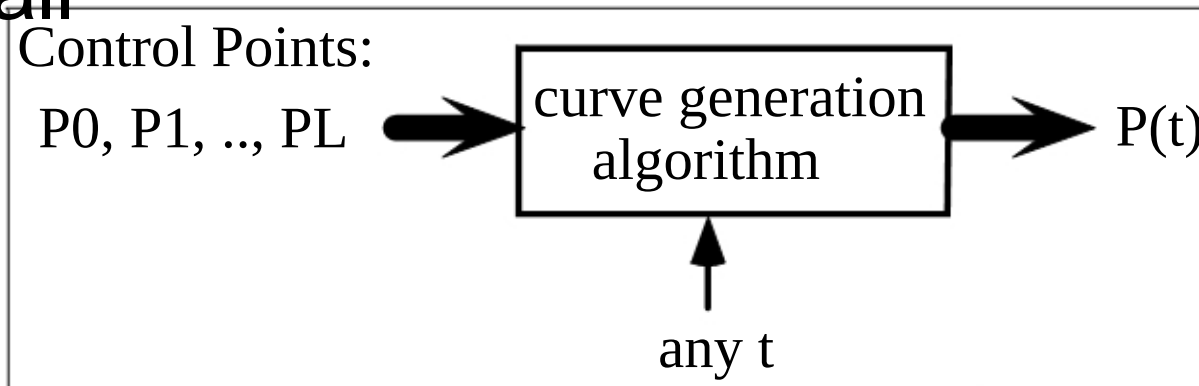


Control polygons

- ♦ To “enter” the curve the designer tapes a sketch of the figure onto a drawing tablet, and then moves the pointer along the curve clicking at a set of *control points* $P_0, P_1, \dots$ close to (not necessarily on) the curve, as shown in Figure 11.9b.
- ♦ The sequence of control points is often called the *control polygon*.
- ♦ The designer enters the control polygon based on a lot of experience, along with a clear understanding of the characteristics of the curve generation algorithm: the algorithm that will later be used to regenerate the curve from the data points.

The Curve-Generation Algorithm.

- ♦ As suggested in Figure below the role of the algorithm is to produce a point $P(t)$ for any value of t given to it.
- ♦ The “data” for the algorithm is the set of control points, which together determine the curve along which the points $P(t)$ will fall



Point Function

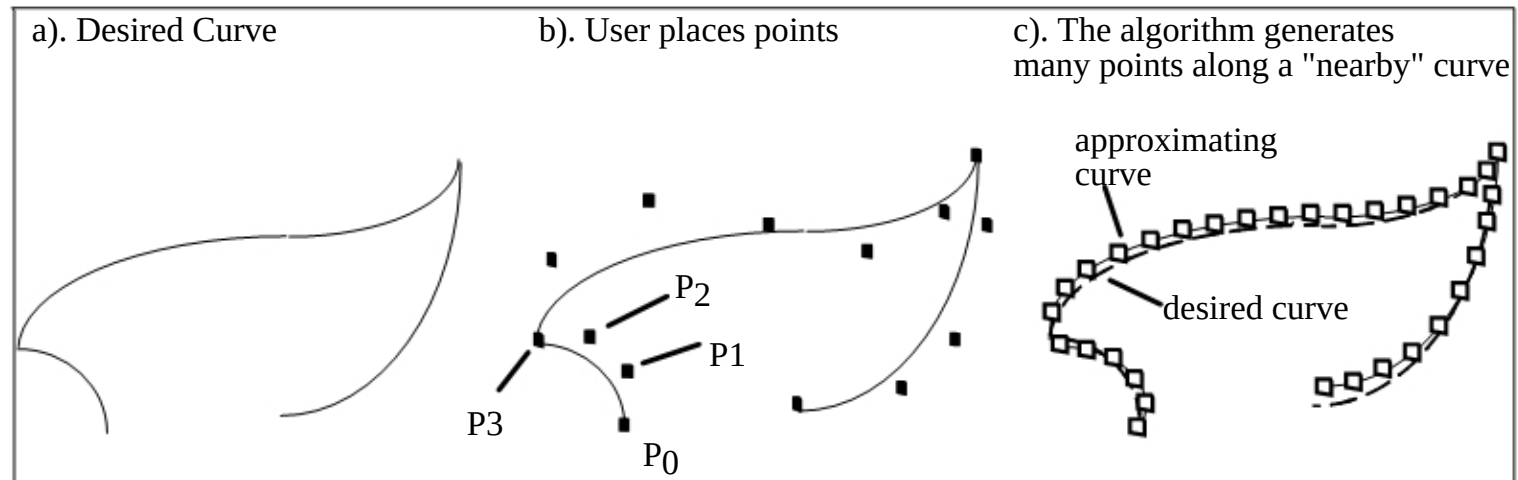
- ♦ The algorithm is usually implemented as a function

Point2 curvePt(double t, RealPointArray pts)

- ♦ This returns a point for any value of t in a certain interval.
- ♦ To draw the curve the user can choose a sequence of t-values, evaluate curvePt() at each of them, and connect the points with line segments to form a polyline.

Desired Curve

- ◆ In the figure below, Part C shows a collection of small squares indicating the points that might be returned by a reasonable algorithm that is given the set of control points in Part B.



Polyline and the Iterative process

- ♦ Hopefully the polyline defined by these points will closely approximate the original curve the designer had “in mind” when entering the control points.
- ♦ If the generated curve does not provide an adequate approximation to the original curve, the designer will “edit” the control points, presumably with the mouse, shifting them this way and that and generating the curve again with many calls to `curvePt()`.
- ♦ This iterative process continues until the designer is satisfied. Interactive design therefore consists of the following steps:



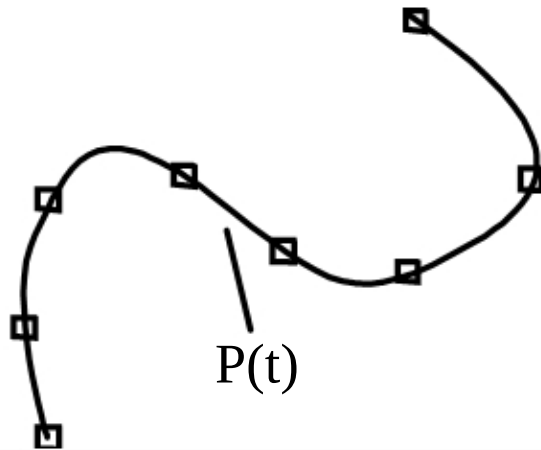
The interactive design process:

1. Lay down the initial control points;
2. Use the algorithm to generate the curve;
3. If the curve is satisfactory stop;
4. Adjust some control points;
5. Go to step 2;

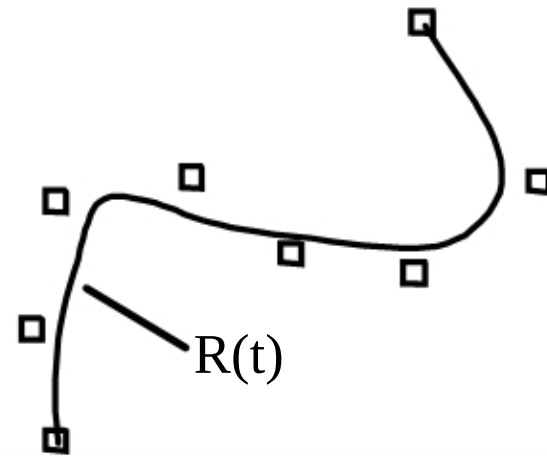
Interpolation versus Approximation.

- The Figure below distinguishes between two main classes of curve generation algorithms.

a). The curve interpolates the control points



b). The curve approximates the control points





A Staple in computer-aided geometric design

- ◆ This scenario of iterative curve design is a staple in the field of computer-aided geometric design (CAGD) and is often used when designing an item to be manufactured.
- ◆ These Bezier and B-spline curves have become very popular in CAGD applications.



Techniques for Curve Approximation

- ◆ We first examine some techniques for curve approximation (whereby the curve $P(t)$ does not necessarily interpolate the control points), focusing on Bezier curves and B-Spline curves.
- ◆ Although it might seem that a designer would always want to interpolate the control points, there are advantages to the curve approximation approach.
- ◆ We then study how to adjust algorithms to insure interpolation of the control points.

Bezier curves for Curve Design.

- ♦ We begin with the simple and elegant **de Casteljau algorithm** that produces Bezier curves, which are fundamental to CAGD.
- ♦ Bezier curves were developed by Paul de Casteljau in 1959 and independently by Pierre Bezier around 1962.
- ♦ They were formulated as ingredients in CAGD systems at two automobile companies, Citroen and Renault, to help design shapes for car bodies.

The de Casteljau Algorithm

- ♦ The de Casteljau algorithm uses a sequence of points, P_0 , P_1 , and P_2 , ..., to construct a well-defined value for the point $P(t)$ at each value of t from 0 to 1.
- ♦ Thus it provides a way to generate a curve from a set of points. Changing the points changes the curve.
- ♦ The construction is based on a sequence of familiar “**tweening**” (hill,p.172) (or lerping) steps that are easy to implement.

Tweening Three Points to Create a Parabola.

- Start with three points, P_0 , P_1 , and P_2 , shown on the following slide. Choose some value of t between 0 and 1, say $t = 0.3$, and locate the point A that is fraction t of the way along the line from P_0 to P_1 .

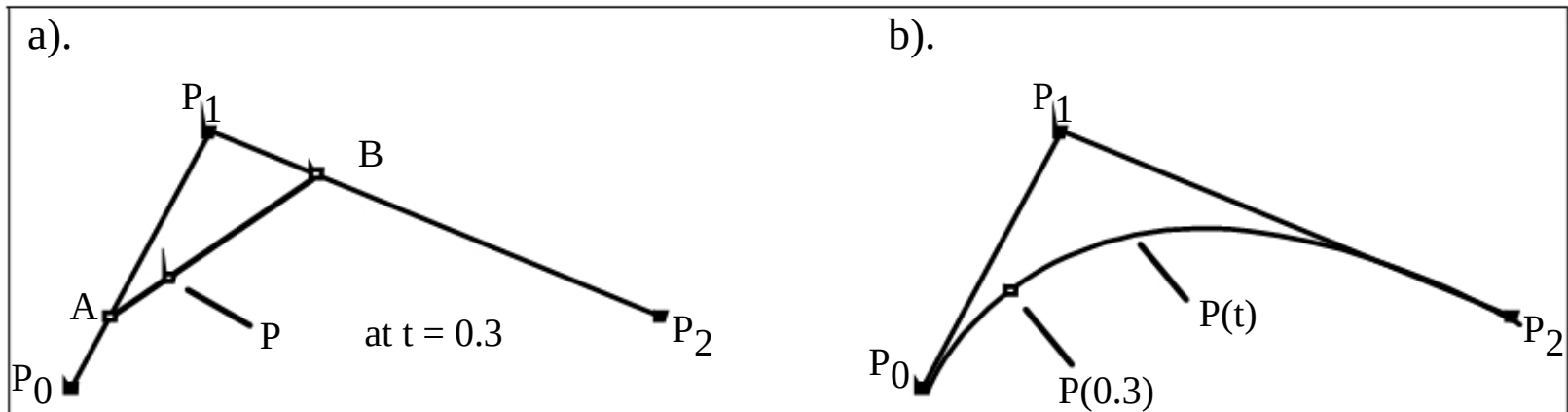


Figure 11-12: Tweening Three Points to Create a Parabola.

Tweening Three Points to Create a Parabola.

- Similarly, locate B at fraction t between the endpoints P_1 and P_2 (using the same t).

$$A(t) = (1 - t)P_0 + tP_1$$

- Using the equation:

$$B(t) = (1 - t)P_1 + tP_2$$

- Now repeat the linear interpolation step on these points (again using the same t):
Find the point, $P(t)$, that lies fraction t of the way between them:

$$P(t) = (1 - t)A + tB$$

Parametric form for the curve

- ◆ As an example, for $t = .5$, $P(0.5)$ is simply the “midpoint between midpoints” for the three given points.
- ◆ If this process is carried out for every t between 0 and 1, the curve $P(t)$ will be generated, as shown in Figure 11.12b.
- ◆ The parametric form for this curve is:

$$P(t) = (1-t)^2 P_0 + 2t(1-t)tP_1 + t^2 P_2$$

Building a Parabola

- ♦ The parametric form for $P(t)$ is quadratic in t , so we know that the curve is a parabola.
- ♦ It will still be a parabola even if t is allowed to vary from $-\infty$ to ∞ .
- ♦ It clearly passes through P_0 at $t = 0$ and through P_2 at $t = 1$.
- ♦ We thus have a well-defined process that can generate a smooth parabolic curve based on three given points.

What if more than three control points are used?

- ◆ The most commonly used family of Bezier curves are those based on four control points.
- ◆ Figure 11.13a shows how the de Casteljau algorithm is applied to the points P_0 , P_1 , P_2 , and P_3 .
- ◆ It is the Bezier curve determined by the four points. Figure 11.13b shows the Bezier curves defined by different configurations of four points.

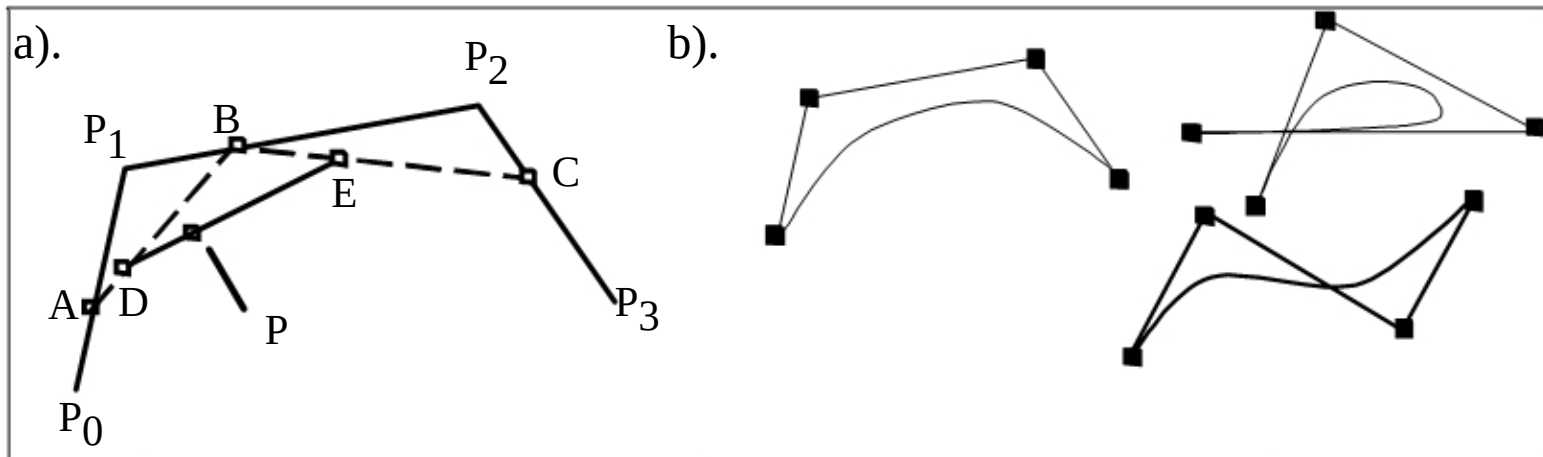


Figure : 11.13

Parametric Form

- ♦ It is easy to show that the Bezier curve based on four points has the parametric form:

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

which is a cubic polynomial in t .

- ♦ Each control point P_i is weighted by a cubic polynomial, and the weighted points are added.
- ♦ See the exercises in Chap. 11)

Bernstein polynomials

- The terms involved here are known as *Bernstein polynomials*. The four cubic **Bernstein polynomials** are:

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

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

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

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

The Cubic Bernstein polynomials

♦ The cubic Bernstein polynomials are easily remembered as the terms one gets by expanding the expression $[(1 - t) + t]^3$ and then collecting terms in the various powers of $(1 - t)$ and t .

Resulting in the equation:

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

- ♦ This immediately yields an important property of these polynomials: they **add to unity** at every t .

$$\sum_{k=0}^3 B_k^3(t) = 1$$

- ♦ For this reason $P(t)$ of Equation:

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

is clearly an affine combination of points, so it is legitimate to add the weighted points together.

The Bernstein polynomials of degree 3.

- Figure 11.14 plots the shapes of the four Bernstein polynomials of degree 3 as t varies between 0 and 1. They are seen to undulate smoothly versus t . Later we see just how “smooth” they are.

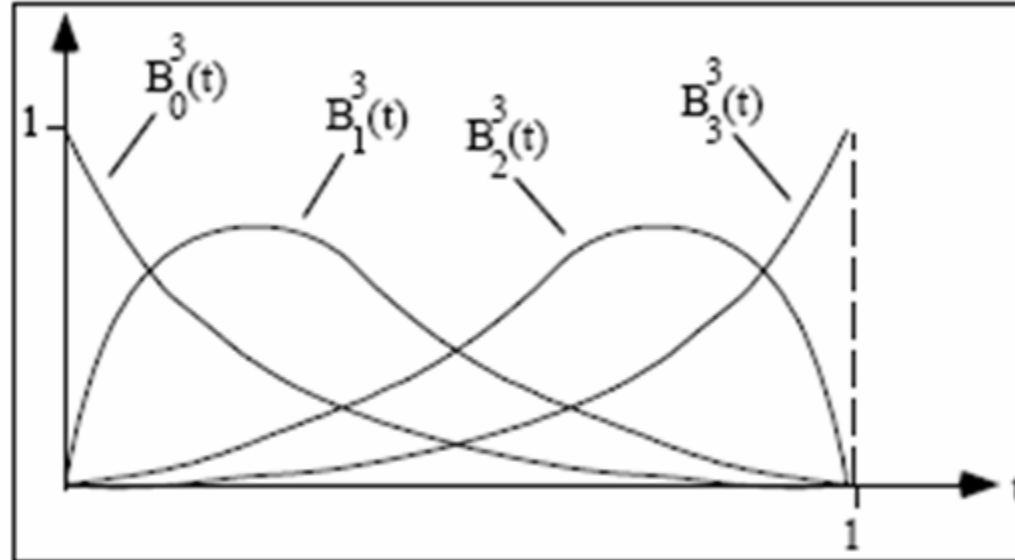


Figure 11.14. The Bernstein polynomials of degree 3.

Weighting and Adding the vectors

- ◆ The Figure below illustrates geometrically how the four points P_0 , ..., P_3 in the equation below are blended together to form $P(t)$.
„ View the points as vectors bound to the origin (so we write P_0 as p_0 , etc.) and let $t = 0.3$.
- ◆ The equation is:
$$p(0.3) = 0.343 p_0 + 0.441 p_1 + 0.189 p_2 + 0.027 p_3$$

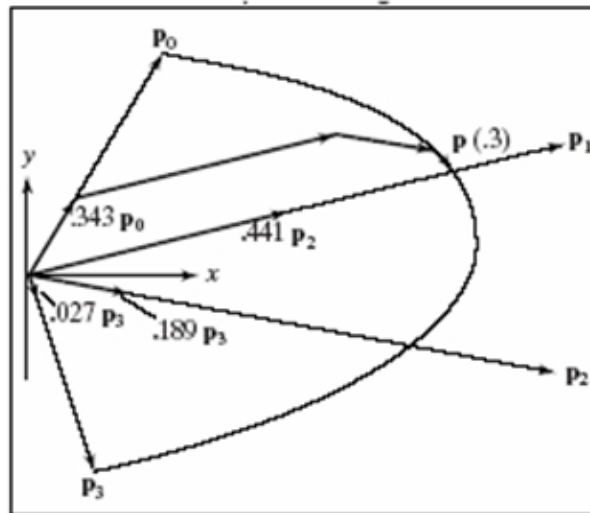
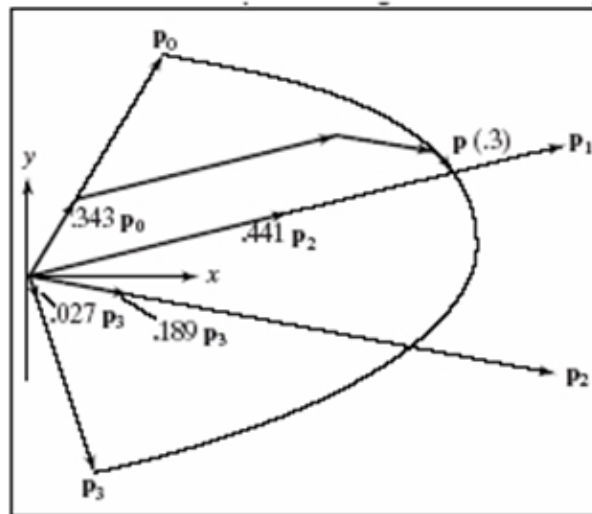


Figure 11.15: Blending four vectors with Bernstein polynomials

Weighting and Adding the vectors

- ◆ In the figure the four vectors are weighted and the results are added using the parallelogram rule to form the vector $p(0.3)$.
- ◆ It is seen that at $t = 0.3$ the main contributors are p_0 and p_1 , although the effect of p_2 is also apparent since p_2 is such a “big” vector.



Weighting and Adding the vectors

- As t varies, the relative weights on the four vectors change, and $p(t)$ shifts to different positions.
- Try to visualize how the vectors would be weighted at $t = 0.5$, and how the four contributions would collaborate to form the vector $p(0.5)$.

Extending the de Casteljau Algorithm to any Number of Points.

- ◆ We have seen that the de Casteljau algorithm uses tweening to produce quadratic parametric representations when three points are used, and cubic representations when four points are used.
- ◆ It generalizes gracefully to the case in which $L+1$ control points $P_0, P_1, \dots, P_L$ are used.

Tweening adjacent points

- For each value of t a succession of “generations” are built up, each one by tweening adjacent points produced in the previous generation:

$$P_i^4(t) = (1 - t)P_i^3(t) + tP_{i+1}^3(t)$$

K

$$P_i^L(t) = (1 - t)P_i^{L-1}(t) + tP_{i+1}^{L-1}(t)$$

for $i = 0, \dots, L$.

- The superscript k in $P_i^k(t)$ denotes the generation.
- The process starts with $P_i^0(t) = P_i$ and ends with the final Bezier curve

$$P(t) = P_i^L(t)$$

- „ The resulting Bezier curve can be written in terms of Bernstein polynomials

$$P(t) = \sum_{k=0}^L P_k B_k^L(t)$$

where the k-th Bernstein polynomial of degree L is defined as

$$B_k^L(t) = \binom{L}{k} (1-t)^{L-k} t^k$$

Binomial Coefficient function

- Here $\binom{L}{k}$ is the binomial coefficient function, given by

$$\boxed{\binom{L}{k} = \frac{L!}{k!(L-k)!}} \text{ for } L \leq k$$

Recall: The Bernstein Polynomials

Sum to 1 for all t

- The value of this term is 0 if $L < k$.
- Each of the Bernstein polynomials is seen to be of degree L .
- The Bernstein polynomials are the terms one gets when expanding $[(1-t) + t]^L$, so

we are assured that

$$\sum_{k=0}^L B_k^L(t) = 1$$

For all t , and that $P(t)$ is a legitimate affine combination of points.