For $\dot{p}_{end}$

$$\begin{bmatrix} 1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 6 \\ 2 \\ 1 \end{bmatrix} = 0 \Longrightarrow \dot{p}_{end} \text{ is in the plane}$$

Therefore the line is in the plane.

Line in plane

Determine if a line lies in a plane.

$$\text{Line} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = (1-t) \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} + t \begin{bmatrix} 3 \\ 6 \\ 2 \end{bmatrix}$$

Plane $x - y + z + 1 = 0$

Discussion. The only difference with the previous problem is that the line is expressed in parametric form and the plane in implicit form. The start and end points can be read directly from the line equation. The plane operator is simply the coefficients of the plane equation, that is, $\begin{bmatrix} 1 & -1 & 1 & 1 \end{bmatrix}$.

Intersecting lines

Determine if the lines $\overline{L}$ and $\overline{K}$ intersect.

$$\text{Line } \overline{K}(s) = \begin{bmatrix} 2 \\ 1 \\ 7 \end{bmatrix} + s \begin{bmatrix} -4 \\ 4 \\ -8 \end{bmatrix}$$

$$\text{Line } \overline{L}(t) = \begin{bmatrix} 3 \\ 5 \\ 2 \end{bmatrix} + s \begin{bmatrix} 6 \\ -3 \\ -3 \end{bmatrix}$$

Discussion. Infinite lines intersect if the lines are not parallel and their four defining points are co-planar.

Instead of the cross product, use the scalar product to test for parallelism. If the vectors of the two lines are parallel so are the lines. The angle between parallel vectors is 0° or 180°, and the cosine of the angle is 1 or -1. Thus for parallel vectors

$$\begin{aligned} \vec{u} \bullet \vec{v} &= |\vec{u}||\vec{v}| \cos\alpha \\ &= \pm |\vec{u}||\vec{v}| \end{aligned}$$

Squaring both sides removes the effect of a 180° rotation.

$$(\vec{u} \bullet \vec{v})^2 = |\vec{u}|^2 |\vec{v}|^2$$

Expanded this is

$$(\vec{u}_x \vec{v}_x + \vec{u}_y \vec{v}_y + \vec{u}_z \vec{v}_z)^2 = (\vec{u}_x^2 + \vec{u}_y^2 + \vec{u}_z^2)(\vec{v}_x^2 + \vec{v}_y^2 + \vec{v}_z^2)$$

In this case

$$\begin{aligned}(\vec{u} \bullet \vec{v})^2 &= ((-4)(6) + (4)(-3) + (-8)(-3))^2 \\ &= ((-24) + (-12) + (24))^2 \\ &= (-12)^2 \\ &= 144\end{aligned}$$

and

$$\begin{aligned}|\vec{u}|^2 |\vec{v}|^2 &= ((-4)^2 + 4^2 + (-8)^2)(6^2 + (-3)^2 + (-3)^2) \\ &= (16 + 16 + 64)(36 + 9 + 9) \\ &= (96)(54) \\ &= 5184\end{aligned}$$

The example lines are not parallel.

With non-parallelism established, use both points of one line and the start point of the other to determine a plane. Use Equation 6.6 on page 109 to determine if the three points are collinear – all three components of the cross product must be equal to zero. If so, the lines intersect.

Collinearity of $\overline{K}_{start}$, $\overline{K}_{end}$ and $\overline{L}_{start}$

$$\begin{aligned}\overline{K}_{\vec{u}} = \vec{u} &= \begin{bmatrix} -4 & 4 & -8 \end{bmatrix}^T \\ \overrightarrow{K_{start} L_{start}} = \vec{w} &= \begin{bmatrix} -5 & 4 & -5 \end{bmatrix}^T\end{aligned}$$

$$\begin{aligned}\vec{n} &= \vec{u} \otimes \vec{w} \\ &= \begin{bmatrix} 4(-5) - (-8)4 & (-8)(-5) - (-4)(-5) & (-4)4 - 4(-5) \end{bmatrix}^T \\ &= \begin{bmatrix} 12 & 20 & 4 \end{bmatrix}^T\end{aligned}$$

The cross product vector is not zero, so the points are not collinear. The cross product $\vec{n}$ is normal to the plane formed by the three points.

With non-collinearity now known, use Equation 6.8 on page 110, the normal vector $\vec{n}$ and any of the three points, say, $\overline{K}_{start}$, to determine a plane operator. Test the end point of the second line with the plane operator.

The plane operator $\lambda = \left[\begin{array}{c|c} \overrightarrow{n} & d \end{array} \right] = \left[\begin{array}{cccc} 12 & 20 & 4 & -72 \end{array} \right]$

$$\begin{aligned} d &= -(122 + 201 + 47) \\ &= -(24 + 20 + 28) \\ &= -72 \end{aligned}$$

Test $\overline{L}_{end}$ against λ.

$$\left[\begin{array}{cccc} 12 & 20 & 4 & -72 \end{array} \right] \left[\begin{array}{c} 3 \\ 2 \\ -1 \\ 1 \end{array} \right] = 36 + 40 - 4 - 72 = 0$$

Since all four points are co-planar and the two lines are not parallel, the lines intersect.

Since the lines are in parametric form, they define both infinite lines and line segments. These segments intersect if the infinite lines intersect and the point of intersection has parameter values s and t on each line between zero and one, $0 \leq s \leq 1$ and $0 \leq t \leq 1$. This latter test requires that the actual parameters and therefore point of intersection be computed. See Section 6.8.3 below.

6.8.2 Generate an object lying on another object

Plane through point

Find the equation of a plane λ through $\dot{m}(2,-1,5)$ and parallel to the plane γ through the points $\dot{a}(3,-7,1)$, $\dot{b}(2,0,-1)$, $\dot{c}(1,3,0)$.

Discussion. The plane γ is expressed as three points. Use Equation 6.7 on page 110 to determine a vector normal and a d value for the plane.

Find the plane normal using the cross product on vectors between pairs of points.

$$\begin{aligned} \overrightarrow{\dot{a}\dot{b}} &= \left[\begin{array}{ccc} 2 & 0 & -1 \end{array} \right] - \left[\begin{array}{ccc} 3 & -7 & 1 \end{array} \right] = \left[\begin{array}{ccc} -1 & -7 & -2 \end{array} \right] \\ \overrightarrow{\dot{a}\dot{c}} &= \left[\begin{array}{ccc} 1 & 3 & 0 \end{array} \right] - \left[\begin{array}{ccc} 3 & -7 & 1 \end{array} \right] = \left[\begin{array}{ccc} -2 & 10 & -1 \end{array} \right] \end{aligned}$$

$$\begin{aligned} \overrightarrow{n} &= \overrightarrow{\dot{a}\dot{b}} \otimes \overrightarrow{\dot{a}\dot{c}} \\ &= \left[\begin{array}{ccc} 10(-2) - (-1)7 & (-1)(-1) - (-2)(-2) & (-2)7 - 10(-1) \end{array} \right]^T \\ &= \left[\begin{array}{ccc} 13 & 5 & -4 \end{array} \right]^T \end{aligned}$$

Use this normal and the given point $\dot{m}$ in the plane operator of Equation 6.5 on page 104 yielding an equation in the single unknown d. Solve for d.

$$\begin{aligned} d &= -(132+5(-1)+(-4)5) \\ &= -(26-5-20) \\ &= -1 \end{aligned}$$

The plane λ is then $\left[\begin{array}{c|c} \vec{n} & d \end{array}\right] = \left[\begin{array}{cccc} 13 & 5 & -4 & -1 \end{array}\right]$.

Plane through point

Find the equation of the plane λ through $\dot{q}(6,1,0)$ and perpendicular to the line

$$\overline{L}: \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix} + s \begin{bmatrix} 8 \\ -2 \\ 0 \end{bmatrix}$$

Discussion. The line's direction vector is $\vec{n} = \left[\begin{array}{ccc} 8 & -2 & 0 \end{array}\right]^T$.

The plane λ is then

$$\lambda : \vec{n} \bullet (\dot{p} - \dot{q}) = 0$$

For the given points the equation is

$$\begin{bmatrix} 8 \\ -2 \\ 0 \end{bmatrix} \bullet \left(\dot{p} - \begin{bmatrix} 6 \\ 1 \\ 0 \end{bmatrix} \right) = 0$$

6.8.3 Intersect two objects

Line and plane

Consider the line $\overline{L}$ given by point $\dot{p}$ and vector $\vec{d}$.

$$\dot{p}(t) = \dot{p} + t\,\vec{d}.$$

Also, consider a plane λ determined by a point $\dot{q}$ and a normal vector $\vec{n}$.
Determine the intersection of the line and the plane.

Discussion. Convert the plane description to the *plane operator* form, that is,

$$\lambda = \begin{bmatrix} \vec{n}_x & \vec{n}_y & \vec{n}_z & d \end{bmatrix}$$

where

$$d = -\vec{n} \bullet \dot{q}$$

Test each of the line's endpoints against the plane operator.

$$\begin{aligned} \text{Let } \dot{p}_{start} &= \dot{p} \\ \dot{p}_{end} &= \dot{p} + \vec{d} \\ \lambda_{start} &= \lambda \bullet \dot{p}_{start} \\ \lambda_{end} &= \lambda \bullet \dot{p}_{start} \end{aligned}$$

If $\lambda_{start} = 0$ and $\lambda_{end} = 0$, the line is on the plane, else

If $\lambda_{start} = \lambda_{end}$, the line is parallel to the plane, else

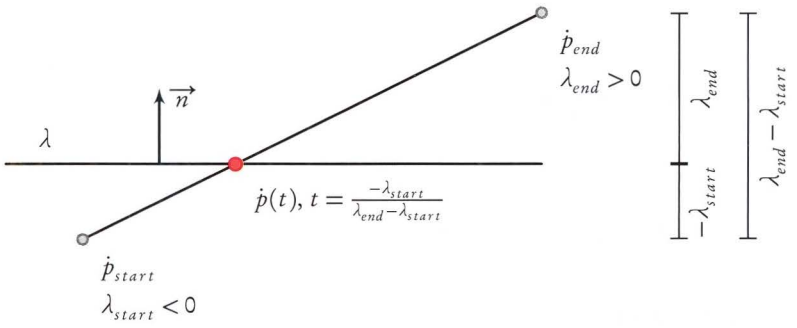
If $\text{sign}(\lambda_{start}) = \text{sign}(\lambda_{end})$, the segment is on one side of the plane, else

If $\text{sign}(\lambda_{start}) \neq \text{sign}(\lambda_{end})$, the segment intersects the plane.

The actual intersection occurs at the point with parameter

$$\frac{-\lambda_{start}}{\lambda_{end} - \lambda_{start}}$$

Figure 6.31 shows why this is so.



6.31: Computing the parameter value for the intersection point of a plane and a line. The plane is viewed "on edge", that is, the plane's normal vector is perpendicular to the view vector. The values λ_{start} and λ_{end} are proportional measures of the signed distance of their respective points to the plane (they are actually scaled by the length of the normal vector). The ratio of λ_{start} to $(\lambda_{end} - \lambda_{start})$ is the parameter value sought. This occurs because the domain λ_{start} to λ_{end} maps linearly onto the range 0 to 1.

Plane and plane

Consider two planes, λ and γ defined in normal-point form.

$$\lambda : \overrightarrow{n}_{\lambda} \bullet (\dot{p} - \dot{q}_{\lambda}) = 0$$
$$\gamma : \overrightarrow{n}_{\gamma} \bullet (\dot{p} - \dot{q}_{\gamma}) = 0$$

The locus of points $\dot{p}$ defines either a plane, a line or is null. Determine the intersection of the two planes.

Discussion. If $\overrightarrow{n_{\lambda}} \otimes \overrightarrow{n_{\gamma}} = \overrightarrow{0}$ the planes are parallel and the intersection is either null or the planes are the same.

If the planes are parallel and the point of one plane is on the other, the planes are coincident and the intersection can be specified simply as either plane.

$$\overrightarrow{n_{\lambda}} \bullet (\dot{q}_{\gamma} - \dot{q}_{\lambda}) = 0$$

Otherwise the planes intersect on the line $\overline{L}$. A normalized direction vector of the line is the normalized cross product of the two plane vectors.

$$\overrightarrow{dir}_{\overline{L}} = \frac{\overrightarrow{n}_{\lambda} \otimes \overrightarrow{n}_{\gamma}}{|\overrightarrow{n}_{\lambda} \otimes \overrightarrow{n}_{\gamma}|}$$

A point on the two planes completes a point-vector equation for the line. Clearly, there is an infinity of such points. A suitable one is the point on the line that is closest to the origin. This point is found as the solution of the following three linear equations ($\dot{o}$ is the origin).

$$\lambda : \overrightarrow{n_{\lambda}} \bullet (\dot{p} - \dot{q}_{\lambda}) = 0 \qquad \dot{p} \text{ is on } \lambda$$
$$\gamma : \overrightarrow{n_{\gamma}} \bullet (\dot{p} - \dot{q}_{\gamma}) = 0 \qquad \dot{p} \text{ is on } \gamma$$
$$\overrightarrow{dir}_{\overline{L}} \bullet \overrightarrow{\dot{o}\dot{p}} = 0 \qquad \overrightarrow{\dot{o}\dot{p}} \text{ is perpendicular to } \overline{L}$$

Plane and plane

Determine the intersection of the following two planes.

$$\lambda \quad : 2x - 3y + 5z = 2$$
$$\gamma \quad : 3x - y + z = 4$$

Discussion. This problem is the same as the prior problem, except that the first two equations are in implicit form – the respective plane vectors are simply the coefficients of the x, y and z terms in the equations.

Like the previous example, finding the point needed to define the result line requires three equations. The first two are the implicit plane equations. The third results from taking the cross product of the normal vectors to the planes.

$$\begin{bmatrix} 2 \\ -3 \\ 5 \end{bmatrix} \otimes \begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 13 \\ 7 \end{bmatrix}$$

The three needed equations are thus as follows:

$$\begin{aligned} 2x - 3y + 5z &= 2 && \dot{p} \text{ is on } \lambda \\ 3x - y + z &= 4 && \dot{p} \text{ is on } \gamma \\ \overrightarrow{OP} \bullet \begin{bmatrix} 2 \\ 13 \\ 7 \end{bmatrix} &= 0 && \overrightarrow{OP} \text{ is perpendicular to } \overline{L} \end{aligned}$$

6.8.4 Closest fitting object

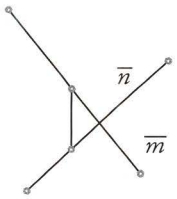
Line between two lines

Given two lines in space, determine the shortest line segment joining the two lines. If the lines are parallel there is an infinity of shortest lines, all themselves mutually parallel. If the lines intersect, the line has zero length but still exists. If the lines do not intersect and are not parallel, they are said to be *skew*. Many CAD systems provide this function, either separately or as a special condition of intersection. It can also be computed efficiently. The point of including it here is to demonstrate how simple geometric constructions can produce needed answers. Sometimes direct geometric reasoning is faster; certainly it is more designerly than equation solving.

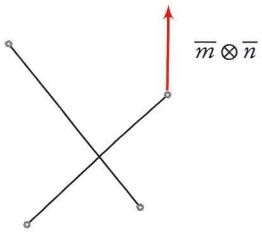
Discussion. The solution combines a cross product, two vector projections, a converse vector projection and a few miscellaneous operations such as point–vector sums. Each can be visualized as a step in a geometric construction.

Geometrically, when the two lines are parallel, the solution is indeterminate as there is an infinity of lines perpendicular to both and of identical length. In this case the cross product of the two lines is the zero vector.

When the cross product is not the zero vector, the two lines either intersect directly or are skew. In this case, the cross product produces a vector partially defining the line. What remains is to find a point on the line and to scale the vector so that its length is equal to the shortest distance between the two lines.

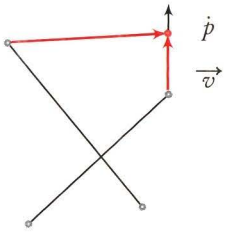


(a)

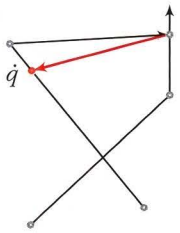


(b)

6.32: (a) Two skew lines $\overline{m}$ and $\overline{n}$ and the shortest line between them.
(b) Compute the cross product of the vectors of $\overline{m}$ and $\overline{n}$ and place at the end of line $\overline{n}$. This vector defines the direction of the shortest line between the two lines $\overline{m}$ and $\overline{n}$.

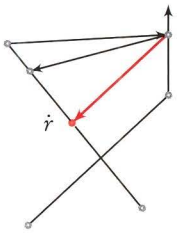


(c)

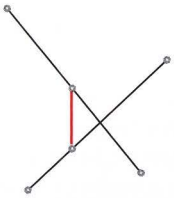


(d)

6.33: (c) Project a point from line $\overline{m}$ onto the cross product vector. Since the cross product vector is located, this produces a point $\dot{p}$. The length of the vector $\overrightarrow{v}$ between the end point of the line $\overline{m}$ and this point is the length of the shortest line.
(d) Project point $\dot{p}$ back onto line $\overline{m}$. This gives a point $\dot{q}$ and a vector $\overrightarrow{\dot{p}\dot{q}}$ perpendicular to $\overline{m}$.



(e)



(f)

6.34: (e) Conversely project $\overrightarrow{\dot{p}\dot{q}}$ onto the vector of $\overline{n}$. Add the resulting vector to point $\dot{p}$ on the cross product. This produces the point $\dot{r}$ as one end of the shortest line.
(f) Project point $\dot{r}$ onto the line $\overline{n}$. Alternatively, subtract the vector $\overrightarrow{v}$ from the point $\dot{r}$ just found.

6.9 Curves

Lines and planes are convenient. They are simple to represent and objects based on them are simple to construct. Look around you though. Unless you are on a wilderness survival trip, you are most likely surrounded by artifacts – things people have created, manufactured or built. When practical, for example, in paper sizes and window glass, straight lines and flat surfaces dominate. Most artifacts though have curved outlines and surfaces. Look carefully at a few of the artifacts around you and ask yourself the question "What determined the curve used in this design?" In a physically constrained environment, function typically dominates. For example, a sailing dinghy operates in the boundary between water and wind. Its form is deeply constrained by the complex forces acting upon it. Hull, centreboard, rudder and sails are all designed to convert force efficiently into forward motion and take whatever form is needed to meet that end. In an environment constrained by fabrication, the tools used impose geometry on the design. For example, in building construction, the relative lower cost of straight elements puts a premium on straight lines, flat surfaces and curves that can directly develop from them. In less constrained situations, representational tools impose geometry. As I write, I am sitting at a desk with a computer, a digital camera, a printer, an MP3 player, a telephone, a mobile telephone, a calculator and a set of speakers in my immediate view. Each has circular curves in its design that seem neither functional, nor constructional in origin. I would guess that they arise simply because it is easy for a designer to make a circle in a drawing or model. Function, fabrication and representational convenience all seem to influence the forms we make. CAD systems introduce computation as a fourth determinant. Especially with curves and surfaces, the tools CAD systems provide are formed more by computational tractability than functionality, constructional affordances or representational appropriateness. Indeed, the history of curves and surfaces in CAD systems can be well-read as the progressive development of representations guaranteeing increasing levels of computational capability. There is amazingly little in this literature about intended function or constructional constraint!

This section introduces curves, especially the so-called *free-form curves*, paying particular attention to the computational properties that they provide. This book is about parametric modeling, not mathematics, so why go into depth here? One answer is that curves are exemplary parametric objects. They can be defined clearly and elegantly using simple parametric structures. Understanding how this is done may well help in making your own structures. Another answer is that the architectural literature contains much nonsense about how curves and surfaces relate to architecture (I could cite some of the guilty parties here, but this would not be fair – there are too many to name them all). By showing the mathematics of curves in a largely qualitative (and hopefully readable) form, perhaps I can remove some of the mystery around these very common design objects and help writers avoid future embarassment.

Designers describe curves by specifying a small set of objects (often points) that form an abstract representation of the curve. An algorithm then computes the curve from these objects. For example, a circle can be described as a centre, a radius and a plane to which the circle is parallel.

6.9.1 Conic sections

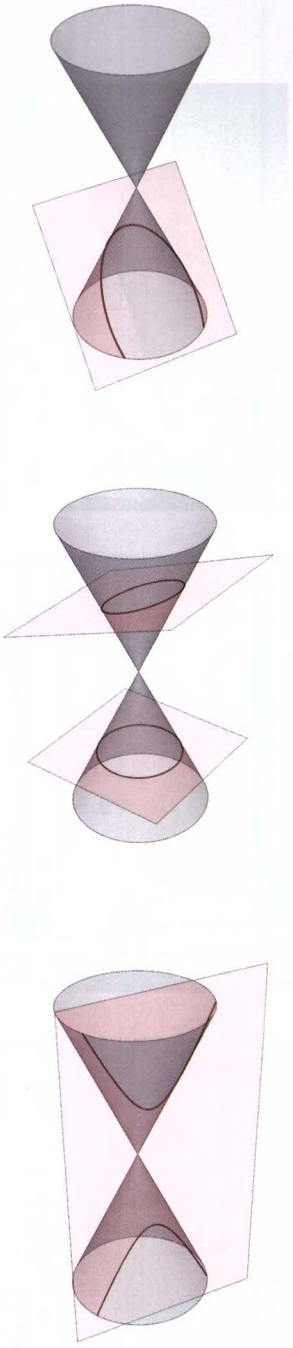
The *conic sections* curves are the circle, parabola, hyperbola and ellipse. Each is described by an equation with a maximum exponent of two. Each is relatively simple to draw and physically construct. Designers, being sensible and frugal, use these curves frequently. The key issue is connecting them smoothly. For joining circle segments, the French curves of manual drawing are a mature and stable technology. Repeating use of conic sections through a design can aid visual composition. Conic sections do present problems. While they can be joined without obvious kinks (this is called first-order continuity), they cannot achieve any higher smoothness.

In contemporary CAD, both conventional and parametric, designers use these curves less than they might. The so-called *free-form curves* are easy to use and give the immediate appearance of fluid control.

6.9.2 When conic sections are not enough

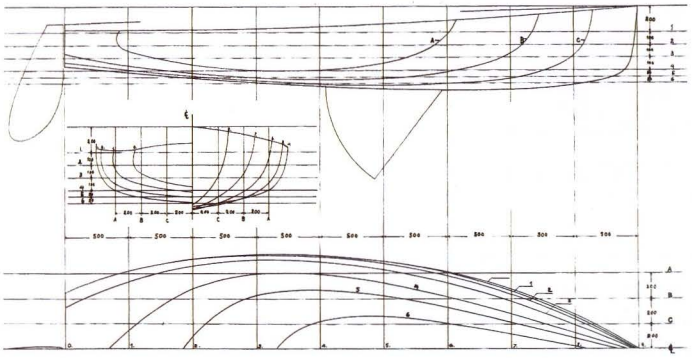
Sometimes conic sections are truly not sufficient for design. Figure 6.37 shows a boat hull (the International Finn Dinghy). Its design is influenced, nay driven, by narrow considerations of stability, speed and volume and by the designer's eye for a fair and sleek form. In its constrained world, conics hinder rather than help. Other domains feel similar forces, for instance, airplanes, automobiles and hand-held tools. In the 1990's and 2000's there was certainly great interest in non-conic sections in architectural design. Whatever the specific motivation, it was seldom comparable to the necessity experienced in other domains.

In CAD, *free-form* curves have come to dominate the toolbox, most likely due to the wide range of forms they encompass and their relative ease of editing. Some parametric modelers do not even support the full range of conic sections! Mathematically, free-form curves are hardly free. Rather, they are a constrained and specific means of expressing parametric polynomial curves. A polynomial is a sum of non-negative integer powers of one or more variables. Each variable may be multiplied by a real coefficient. A polynomial of one variable (called a *univariate* polynomial) has the general form $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$. The equation $3.1x^2 - 2x$ is a polynomial, while $4x^{2.2} + 7x - 4$ is not, as it has a real-valued power. Free-form curves were initially motivated by the process of laying out complex forms using physical splines on a lofting floor and by the reality of World War II. Mathematics could be copied and thus was much less



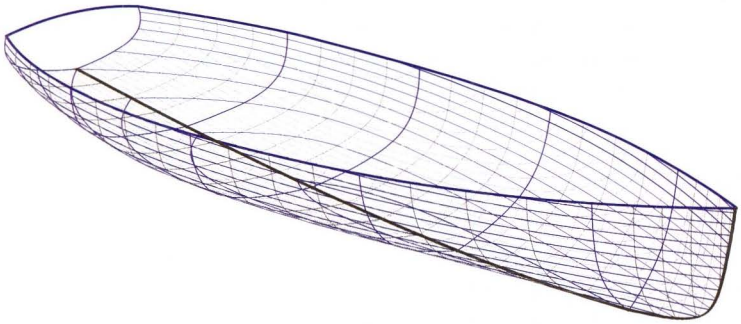
6.35: The four conic section curves and how they are derived from a cone. The circle's plane is perpendicular to the cone axis. The parabola's is parallel to a cone side. The hyperbola's is parallel to the cone axis. All other planes produce an ellipse.

6.36: Finn dimensions persist, but technology advances. The wooden boats of the 1950s had wooden masts and cotton sails. Current fibreglass boats have carbon fibre masts and Mylar sails.
Source: International Finn Association



Source: International Finn Association

(a)



Source: Gilbert Lamboley

(b)

6.37: The lines for the Internatial Finn Class sailboat (designed by Rickard Sarby), which has been in the Olympics since 1952. (a) A drawing of unknown provenance, but believed to be that sent to competing countries as they prepared for the 1952 Helsinki Olympic Games. (b) A digital model of the Finn hull. The history of measurement records for the Finn Class demonstrates the practical need for accurate mathematical representation. Until 1964, the International Finn Association had only tables of offsets sourced from the Scandinavian Yacht Racing Union. The drawings from which those offsets were issued are believed to have disappeared in a fire. Charles Currey (at Fairey Marine) carved a physical template of full size Finn lines (together with transverse template lines) onto sheets of aluminium alloy in 1964. These sheets were treated to neutralize residual stress and so be dimensionally stable. The template was made according to the earlier offsets, obvious mistakes being ignored. Mylar copies taken from the aluminium templates were later found to be dimensionally unstable. Aluminum transverse section templates fabricated for field use themselves deformed over time by being dropped or otherwise impacted. In turn, the orignal template sheets were lost in the late 1990s. In 2003, working from the spotty historical record, Gilbert Lamboley reconstructed the tables of offsets and prepared the first digital Finn models.

likely to be bombed out of existence than was a factory floor. The boat hull in Figure 6.37 provides an actual case of such need as the original drawings were lost in a fire! Prior to having these mathematical and ultimately computer-based representations, the principal design media were drawings and half-models of the hull lines in four orthogonal projections (see Figure 6.37 (a)). Building a boat started by consturcting stations along a strongback. The lines were then faired by physical splines through the stations. In abstracting to mathematics, the constraints of physical splines largely disappeared, leaving only metaphors such as *poles* and the word "spline" itself in the new toolbox. Free-form curves have their own logic, divorced from their physical origins and largely aimed at achieving mathematically and computationally well-behaved curves that can be used in design. In turn, CAD developers and designers have adopted and adapted these curves into their modeling toolboxes. In this process of cultural co-evolution, mathematics enables design, but is constrained by the possible, and design poses new questions to mathematicians based on the realities of the design profession and its marketplace.

The following sections provide an introduction to curves, applicable in both two and three dimensions. They form the mathematically most involved part of this book. Why spend so many pages on such detail? The answer is simple. Curves are exemplary parametric objects. Understanding how they work gives insight into both the form-making possibilities of curves and into parametric modeling in general. With few exceptions, everything learned about curves translates to surfaces, so the chapter on surfaces is brief, introducing only key new concepts needed for effective modeling and design.

6.9.3 Interpolation versus approximation

Figure 6.38 shows that some curve algorithms *interpolate:* they compute curves that go through the input points. Others *approximate:* they place curves that are, in some sense, "near" the input points. We call the input points the *control points* and the (possibly open) polygon they define the curve's *control polygon*.

(a) (b)

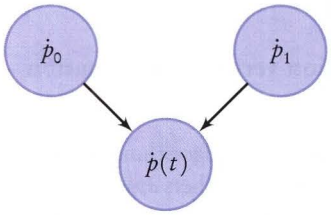
6.38: (a) An interpolating curve and (b) an approximating curve.

In most design systems, the dominant mode is approximation. This may seem surprising: having a curve go through known locations seems a useful idea. The reason is that approximating curves tend to be geometrically more predictable and "well-behaved", and are mathematically more simple.

6.9.4 Linear interpolation ≡ tweening

The fundamental constructor for many kinds of curves employs the concept of *linear interpolation* or *tweening*. Informally, interpolation moves a value "within" a set of other values. Linear interpolation moves it smoothly and in constant proportion. You have seen this concept before in the mathematics of the parametric line and plane equations. Parametric curves result from linearly interpolating a parameter in an equation to generate the points on the curve. In a parametric line, the point and the parameter have a direct relationship: equal increments between parameter values produce corresponding equal increments between points on the line. In curves, this relation becomes indirect. Identical parameter changes can yield unequal spacing between points – the implications deeply affect the form-making process.

Figure 6.39 shows a useful diagram, called a *systolic array*, for representing a parametric line equation, that is, the relationship between $\dot{p}(t)$, $\dot{p}_0$, $\dot{p}_1$ and t. The coordinate values from $\dot{p}_0$ and $\dot{p}_1$ flow into $\dot{p}(t)$ where they combine in the equation $\dot{p}(t) = \dot{p}_0 + t(\dot{p}_1 - \dot{p}_0)$.



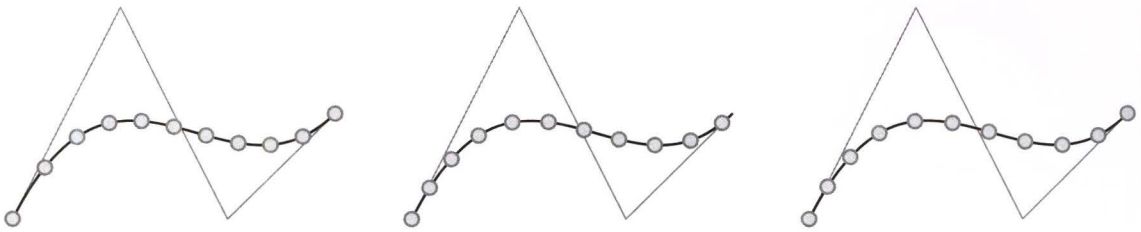
6.39: This *systolic array* comprises three points where the lower point is determined by the upper points and the *systolic array parameter* t. It is the basic (primitive) structure from which systolic arrays representing curves can be constructed.

6.9.5 Parametric curve representations

Like parametric lines, parametric curves are defined by a point that moves with a parameter t.

Unlike lines, the movement is not linear; the distance along a curve between $\dot{p}(t)$ and $\dot{p}(t + \delta t)$ is not necessarily the same as that between $\dot{p}(t + \delta t)$ and $\dot{p}(t + 2 * \delta t)$, where δt is a number expressing a very slight change relative to t.

Points can be placed at uniform increments by distance along the curve – except that the last point may not be at the given distance from the end of the curve. Points can also be placed at uniform spacing given a specified number of points.



(a) (b) (c)

6.40: Points can be distributed along a parametric curve in several ways: (a) shows points at equal parameter intervals of 0.1, that is $t = \{0, 0.1, 0.2, \ldots, 1\}$. Note that the distance along the curve between points varies. (b) shows equal spacing at a given distance. Note that the rightmost point is not at the end of the curve, leaving a gap less than the chosen distance between it and the curve end. (c) shows equal spacing given a specified number of points on the line.

The very big lesson here is that parametric and geometric space are different. It is easy to work in parametric space, but designs are built in geometric space. The difference between the two bedevils much work.

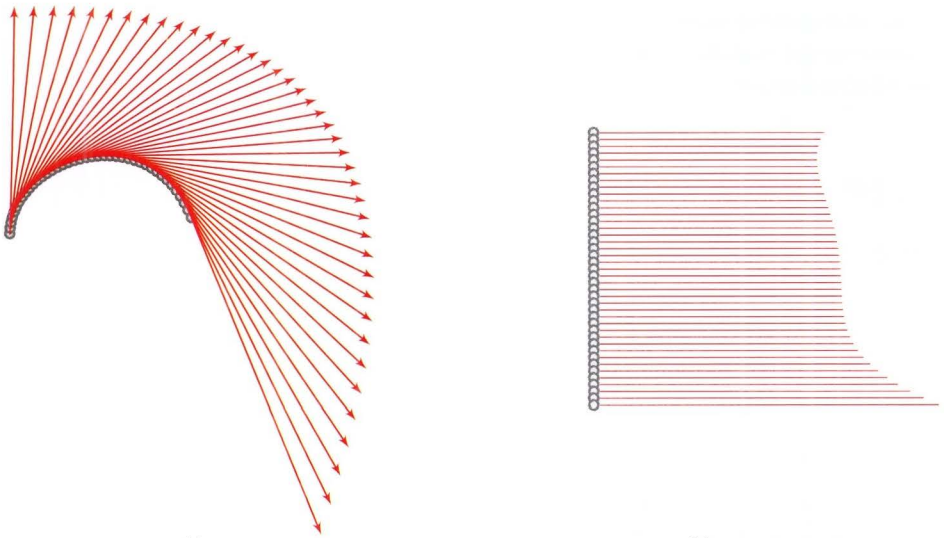
6.9.6 Relating objects to curves

In design, curves relate to other objects and complex relationships are built from simple ones. Two basic relationships involve vectors: tangent and normal.

Tangent vector

Every (well, almost every) point $\dot{p}(t)$ on a curve has a family of vectors tangent to it. The sole exceptions occur when the *first derivative* (from calculus) is not defined or is the zero vector.

Of the infinitely many vectors tangent at a parametric point $\dot{p}(t)$, only one is the *tangent vector*. The reason is that the length of the tangent vector captures the rate at which $\dot{p}(t)$ moves along the curve as t changes. In calculus terms, the tangent vector is the first derivative of the parametric curve at point $\dot{p}(t)$. The tangent vectors vary in length along the curve. When all are bound to one point, it is easy to see that their lengths differ. Figure 6.41 shows points along the curve at equal parameter spacing. The relative geometric distance between points approximates the relative tangent vector lengths. When successive points are close together, the tangent vectors are commensurately short. Normalizing the tangent vector gives the *unit tangent vector*, which is useful, for example, when constructing coordinate systems on curves. See Figure 6.42.



(a) (b)

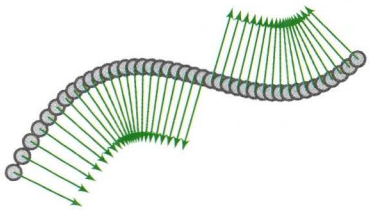
6.41: (a) The tangent vectors at example points along a curve. (b) A set of vectors each of the length of its corresponding tangent vector but sharing the same direction. Notice that the variation in length. This depends on the specific equation representing the curve. In particular the tangent vector is the first derivative of the curve.

(a) (b)

6.42: 6.4(a) Unit tangent vectors arrayed along a curve.
(b) Corresponding vectors, of identical direction and each of length of of the respective unit tangent vector shows that all such vectors share the same length.

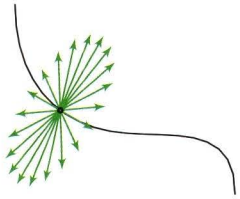
Normal vector

In the case of a two-dimensional curve there is (in almost every case) a unique direction vector normal to the curve. Of the two such vectors, one pointing to each side of the curve, by convention, we choose the one that points "into the curve". Of course, it lies in the plane of the curve.



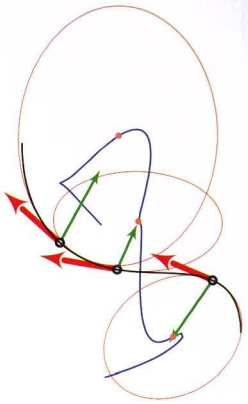
6.43: Almost every point on a two-dimensional curve has a unique unit normal vector.

For three-dimensional curves, things are more complex. A point on the curve and its tangent vector define a plane normal to the curve at that point. Every vector in this plane is normal to the curve.



6.44: A sample from the infinite family of co-planar unit vectors normal to the tangent vector at three points on a curve.

However, there is a distinguished vector in that plane. It is called the *normal vector* of the curve at $\dot{p}(t)$. The normal vector is of unit length and lies in what is called the *osculating plane* of the curve at $\dot{p}(t)$. Its direction is approximated by the second derivative. This is the plane that most closely approximates the curve at $\dot{p}(t)$. Lying in this plane is the *osculating circle*, which is the circle that is both tangent to and has the same curvature as the curve at $\dot{p}(t)$. The centre points of the osculating circles at each point along the curve define another curve called the *evolute*.

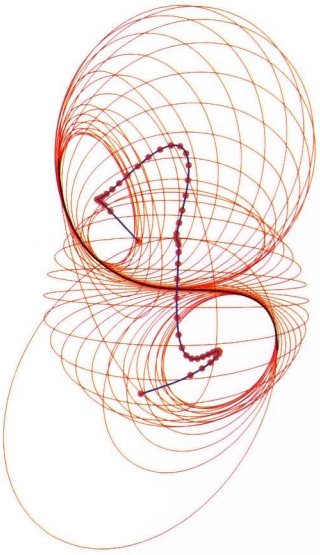


6.45: The unit tangent vector; the normal vector; the osculating circle for three points on a 3D curve; and the curve evolute, the collection of all osculating circle centres.

Binormal vector

The binormal vector is the cross product of the unit tangent vector and the normal vector.

The unit tangent, normal and binormal vectors can be combined into a structure that, with a few exceptions, provides a sensible coordinate system at every point on the curve. This is the *Frenet frame*.



6.46: A collection of 50 osculating circles distributed uniformly along a curve. The circle centres trace the curve evolute.

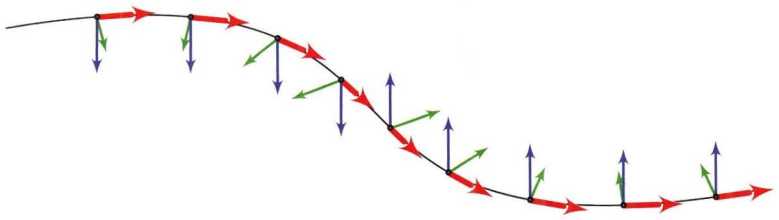
Frenet frames

The Frenet frame is an orthonormal frame defined at (almost) every point on a 3D curve. It comprises the unit tangent, normal and binormal vectors as the x-, y- and z-axes of the frame.

Frenet frames have some problems as design tools. At singular and inflection points they are not defined. When a point crosses an inflection point, the Frenet frame seems to invert or "flip", that is, it instantaneously rotates 180° around the tangent vector. This is not a good thing if, for instance, you are using a Frenet frame to orient windows on a curved façade and the frame inverts twice at each inward curve of the façade. Geometrically the osculating circle has an infinite radius at an inflection point.

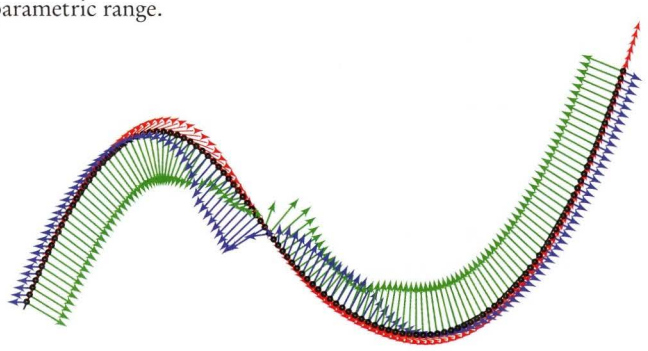
When a curve is confined to a plane, such inflection points are frequent, indeed they are to be expected. Figure 6.47 shows one such curve.

Frenet frames are not defined on straight lines. A straight line is essentially an infinite inflection point.



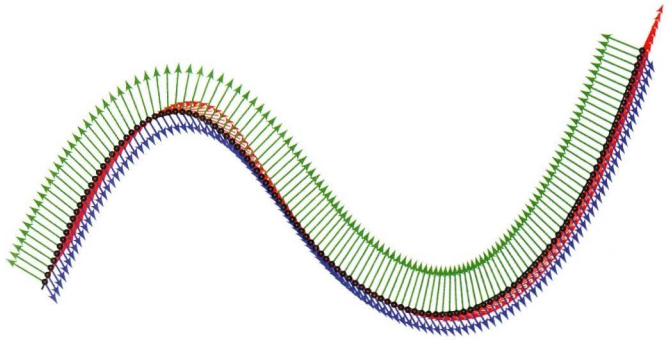
6.47: Frenet frames on a curve that has an inflection point. Note that the Frenet frame inverts on each side of the inflection point. At the inflection point itself, the frame is not defined.

Inflection points seldom occur in 3D curves. In their place comes something worse: high torsion. Even though a curve may appear to contain an inflection point, it usually avoids inflection narrowly by, in effect, twisting around the point. This results in the Frenet frame rotating nearly or exactly 180° within a short parametric range.



6.48: A curve that comes close to an inflection condition results in its Frenet frame rapidly rotating around the curve. This makes it difficult to orient objects along the curve.

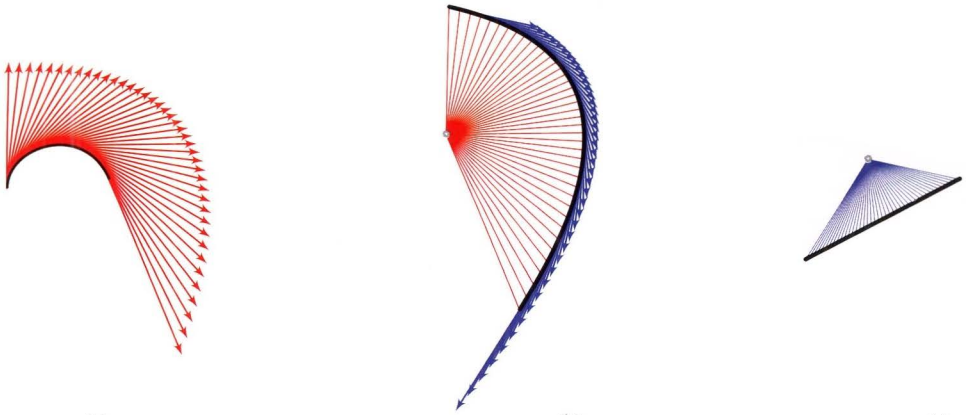
A common remedy for these situations is to adopt a reference direction that does not depend on the local context of the point $\dot{p}(t)$ on the curve. Then place a frame on the curve with its x vector set to be the x vector of the Frenet frame and its z vector at right angles to the x vector and co-planar with the x vector and the reference vector. The frame will be on the curve and its x vector will have the same direction as the curve tangent vector. There are many ways to make the choice, for example, the z vector of the global coordinate system. But such a frame is not a Frenet frame; it no longer holds information on curvature or torsion. Its y vector will not always point towards the centre of curvature. The choice of a reference direction external to the curve will sometimes result in a strange orientation for the new frame. A better choice is to compute a local reference based on three non-collinear points in the curve control polygon or, better, on the average plane of the curve control polygon. This approach will fail less frequently, for example, when the curve is a straight line.



6.49: This is the same curve as in Figure 6.48. The coordinate system on the curve has its x vector tangent to the curve and its z vector approximating a reference vector.

Almost any point...

Several times above, I have stated that a property exists at "almost any point" on a parametric curve. The four exceptions occur where first or second derivatives are either undefined or zero. In practice, curves almost always have defined first derivatives, so this case is rare. Unfortunately, the other three cases are quite common, or at least common enough to cause trouble. Actually, I lied; they are really common. For instance, a line is a curve, but has a constant first derivative and a zero second derivative everywhere, so the Frenet frame is not defined at any point on a line.



(a) (b) (c)

6.50: The (red) tangent vectors at example points along a curve, (a) placed on the curve and (b) collected at a single point into the hodograph and thus displaying the first derivative of the curve. (c) The hodograph of the hodograph collects the (blue) tangent vectors of the first derivative and locates them at the origin. This is the second derivative of the original function. In this and the following two figures, the second derivative vectors are scaled to 20% of their actual length, otherwise the figures become too large. In all three cases the second derivative is a straight line. Foreshadowing Section 6.9.9, this occurs because the example curves are of order 4 and degree 3.

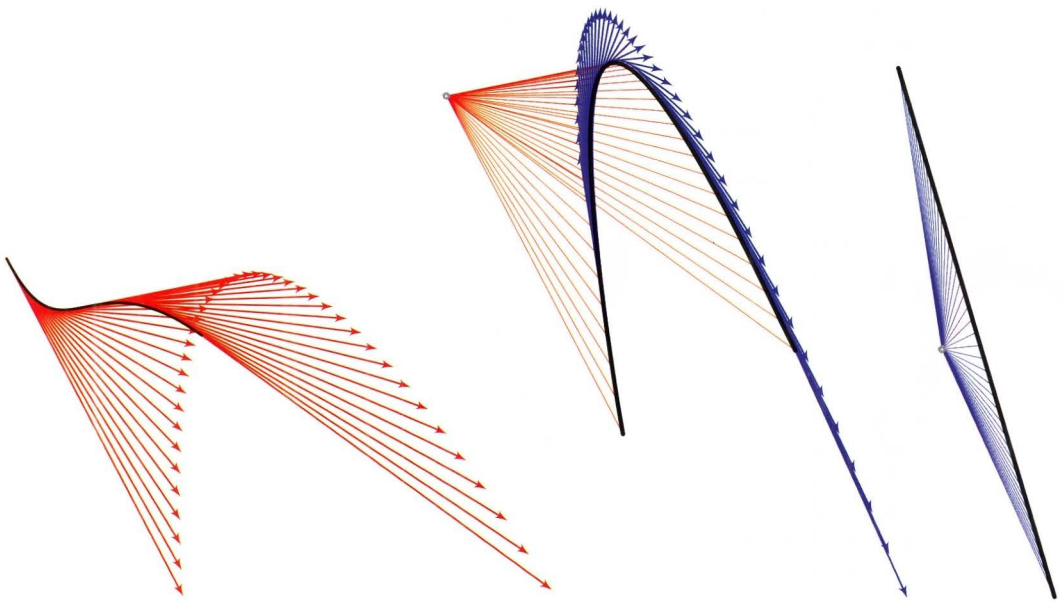
Locating the tangent vectors of a curve at the origin produces the *hodograph*, a curve defined by the ends of the vectors and a good device for illustrating when exceptional points arise. The hodograph is the first derivative of the curve. The hodograph of the hodograph is the second derivative. Using the curve from Figure 6.41 above, Figure 6.50 shows the first and second hodographs.

An exceptional point on the curve occurs when either hodograph goes through zero or the two become locally collinear. The first hodograph goes through zero at a *cusp* and the two hodographs align at inflection points. For example, see the curves in Figures 6.51 and 6.52 below.

6.9.7 Continuity: when curves join

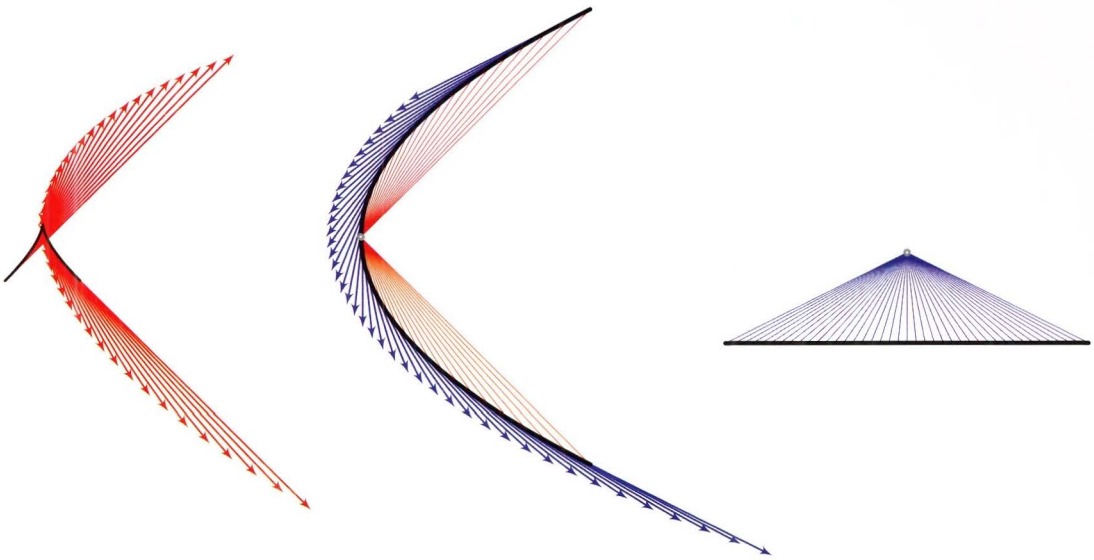
With parametric functions, continuity is trick as it comes in two flavours, one with respect to parametric space and one with respect to geometric space. These are called C and G continuity respectively.

If a curve is connected, it has C_0 continuity. If its first derivative is continuous, the original curve has C_1 continuity. If the n^{th} derivative is continuous, the original curve has C_n continuity. The curves in Figures 6.50, 6.51 and 6.52 are all C_2 continuous.



(a) (b) (c)

6.51: The two hodographs of an inflected curve are collinear at the inflection point, resulting in an undefined Frenet frame.



(a) (b) (c)

6.52: The hodograph (b) of a cusped curve (a) goes through zero, thus the Frenet frame is undefined at the cusp.

However, C continuity does not mean that a curve is geometrically smooth. For example, the cusped curve in Figure 6.52 above has a geometric kink, but is parametrically smooth. This is because C continuity is measured in parameter space not geometric space.

The notion of G continuity captures geometric smoothness.

If a curve is C_0 continuous, it is G_0 continuous. It is connected and this means the same thing in both parameter and geometric space.

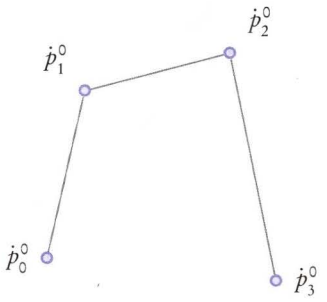
If a curve is G_0 continuous and its tangent direction varies continuously, the curve is G_1 continuous. An example of a curve with C_1 continuity but not G_1 continuity is any curve whose hodograph goes through the origin, as shown in Figure 6.52 above. Coming into the origin, the tangent has one direction – leaving the origin, the tangent jumps to a different direction. Mathematically, G_1 continuity exists if the normalized tangent vector of a curve is continuous.

Most parametric modelers implement C continuity and leave control of G continuity to the user.

The next sections move from generic properties that apply to all curves to representation of specific curve types.

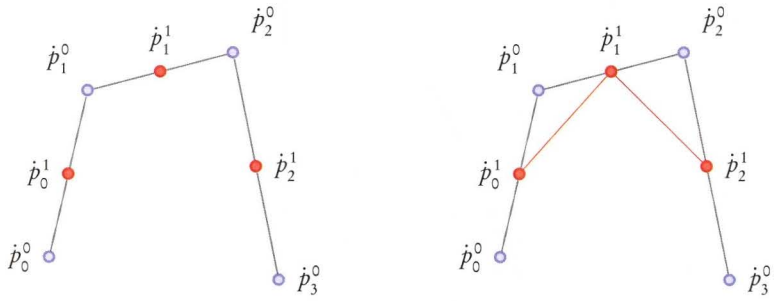
6.9.8 Bézier curves – the most simple kind of free-form curve

The most simple free-form curve is the Bézier curve, which is named after its inventor, Pierre Bézier. The cubic form (more on the *cubic* label later) of the Bézier curve is recursively defined on a control polygon of four points.



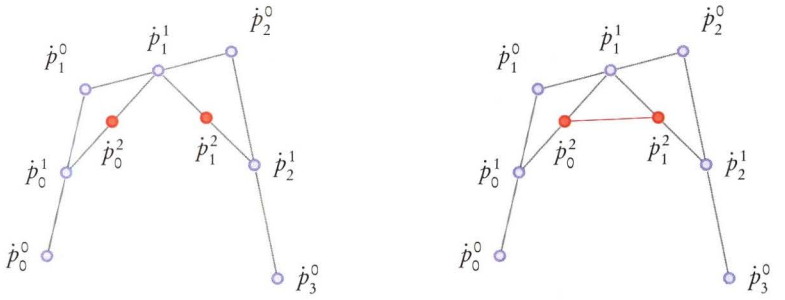
6.53: Bézier curve control points. In the notation $\dot{p}_j^0$, j stands for the j^{th} control point and 0 stands for the i^{th} *level* of the control polygon. This is the outer or 0^{th} level.

Then a parametric point is placed on each line. Each of the lines in the control polygon holds a parametric point with a given parametric value, say, $t = 0.5$. These are the control points of the level 1 control polygon, which joins the level 1 control points in order.



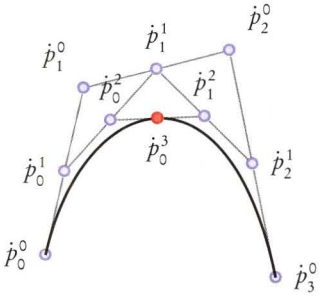
6.54: The level 1 control points $\dot{p}_0^1, \dot{p}_1^1, \dot{p}_2^1$. Each has the same parameter t, in this case, $t = 0.5$. The level 1 control polygon joins these points.

Parametric points with the same $t = 0.5$ value form the level 2 control points, which define the level 2 control polygon comprising a single line.



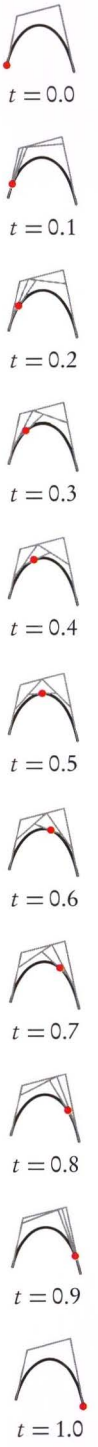
6.55: The level 2 control points $\dot{p}_0^2, \dot{p}_1^2$. Again, each has the same $t = 0.5$ parameter. The level 2 control polygon is a single line.

The level 3 control point $\dot{p}(t) = \dot{p}_0^3$ is on the Bézier curve at parameter $t = 0.5$. Figure 6.57 show that, as t varies from 0 to 1, $\dot{p}(t)$ traces out the Bézier curve.

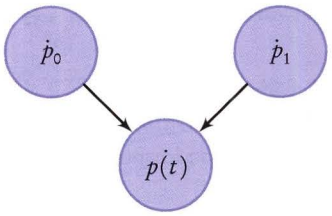


6.56: The level 3 control point $\dot{p}_0^3$. This is the defining point of the Bézier curve.

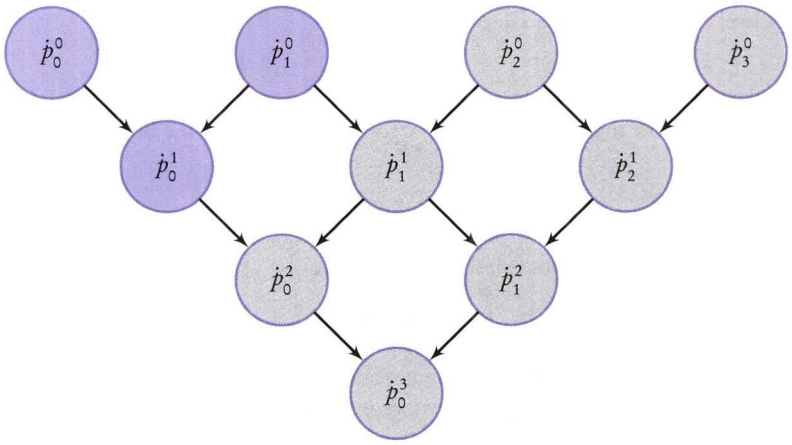
A Bézier curve can be represented symbolically by combining *primitive systolic arrays* (Figure 6.58) into a *composite systolic array* (or just a *systolic array*), shown in (Figure 6.59) that defines the entire Bézier curve. The systolic array suggests a clear convention for labeling the intermediate points of the array. In the point $\dot{p}_j^i$, i refers to the level in the systolic array (starting with the zeroth level) and j refers to the index of the point (the point's place in a sequence) at its particular level. In the systolic array data flows downwards along arcs from higher nodes to lower nodes. The *top* nodes in the systolic array receive no data; they are the inputs to the system. The *internal* nodes of the array receive inputs from those above and connected to them. The arcs denote data flow from an upstream to a downstream node. The nodes in this particular systolic array combine the inputs by a parametric line equation. (An expression could be added to each node to determine how inputs are handled, but that is not necessary here as all nodes use the same simple operation: a sum of a point and a vector scaled by t.)



6.57: As t varies between 0 and 1, $\dot{p}(t)$ travels along, indeed it defines, the curve.



6.58: A *primitive systolic array* records the parametric line equation with the line endpoints as the upstream nodes and the parametric point as the sole downstream node.



6.59: This *systolic array* combines six primitive systolic arrays (the first is shown in blue), each representing a line, to define the parametric point $\dot{p}(t) = \dot{p}_0^3$.

The definition of each point (other than the control points) in the systolic array is simply a parametric line equation using the two points above the point being computed.

The systolic array is used to define what is called the deCasteljau algorithm for computing a point $\dot{p}(t)$ on a Bézier curve for a given value of t. In essence, the algorithm can be stated as:

```
Start with a set of control points p_i^0 and a parameter t.

Create the systolic array

Compute the values of any nodes for which you have the values of
  the upstream nodes.

Stop when you cannot compute anything more.
```

Geometrically, the algorithm steps down through levels in the geometric structure, finding points with the same parameter values at every level.

6.9.9 Order and degree

Bézier curves belong to a very useful class of curves made from polynomials. A polynomial is a sum of monomials. A monomial is a product of constants and variables raised to exponents that are positive integers. For instance, $3x^7$ is a monomial and $4x^2 - 2x + 1$ is a polynomial.

Monomials and polynomials have two descriptors: order n and degree d, with $n = d + 1$. The term *degree* refers to the maximum exponent in the polynomial. *Order* equals $d + 1$. So $4x^2 + 2x + 1$ has degree $d = 2$ and order $n = 3$.

Bézier curves take their order (and thus degree) from the number of vertices in the control polygon. An order 4 Bézier curve has a four-point control polygon, an order 3 curve a three-point control polygon and an order 2 curve a two-point control polygon. Figure 6.60 shows that the two-point case defines a straight line segment. The simple parametric line equation is, in fact, a trivial Bézier curve.

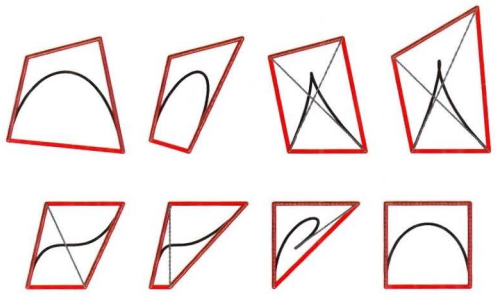
6.60: Order 4, 3 and 2 Bézier curves (in black) and their control polygons (in grey).

6.9.10 Bézier curve properties

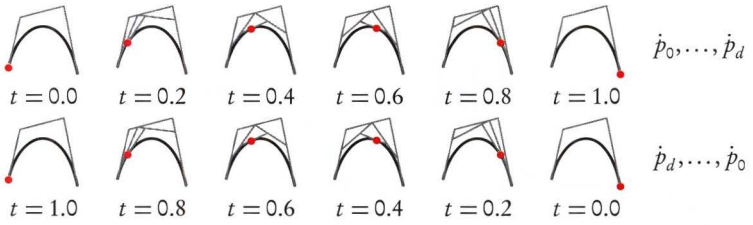
Bézier curves have several useful properties that help us to understand the curves and to write algorithms that use them.

Convex hull. Intuitively, the convex hull of a set of points can be described by thinking of a rubber band stretched around the points. Some of the points will form vertices of a convex polygon; others will be on the interior of the polygon. Such a polygon is called the *convex hull.* It a useful approximation of the region occupied by the points. Algorithms over convex hulls can use the property of convexity. For example, testing if a point lies in a convex hull is a simple matter of checking that the point is on the same side of each hull line (or plane if 3D).

A Bézier curve is entirely contained in the convex hull of its control polygon.

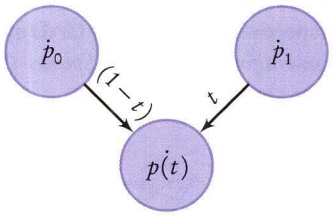


Symmetry. For a Bézier curve it does not matter whether we label the control points $\dot{p}_0,\ldots,\dot{p}_d$ or $\dot{p}_d,\ldots,\dot{p}_0$. The curves corresponding to the two orderings look the same, they only differ by their direction of parametric traversal.



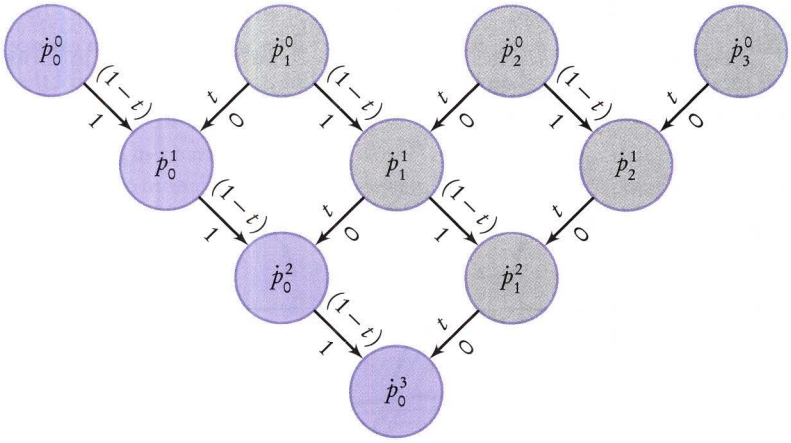
Endpoint interpolation. A Bézier curve of degree d passes through $\dot{p}_0$ and $\dot{p}_d$. In a design situation, having control over the starting and ending points of a curve is very important.

This can be seen directly from the systolic array by labeling each of the arcs with the factor that each source point contributes the result point. Remember that a parametric line equation is written as $\dot{p}(t) = (1-t)\dot{p}_0 + t\,\dot{p}_1$. Encoded into a primitive systolic array (as in Figure 6.61), the left arc carries the factor $1-t$ and the right arc t. The equation in each internal node of the systolic array takes the sum of the upstream points, weighted by the arc factors.



6.61: A primitive systolic array with arcs labeled with the scale factors defined in the parametric line equation.

Labeling the entire systolic array (Figure 6.62) shows that, when $t = 0$, all right branches of the systolic array contribute nothing, meaning that $\dot{p}_0^0$ is the sole contributor to the final point $\dot{p}_0^3$. Thus $\dot{p}(0) = \dot{p}_0^0$. When $t = 1$, $\dot{p}_0^0$ is the sole contributor to $\dot{p}_0^3$. Thus $\dot{p}(1) = \dot{p}_3^0$.

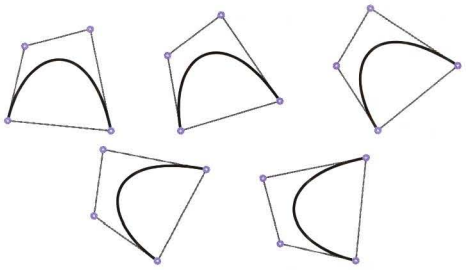


6.62: An entire systolic array labeled with scale factors for the parametric line equation.

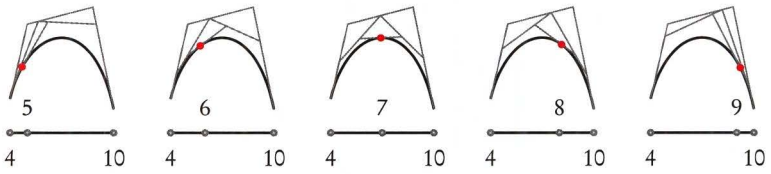
Affine invariance. Bézier curves are invariant under affine maps. This means that the following two processes yield the same result: (1) first calculate $\dot{p}(t)$ and then apply an affine map to it; (2) first apply an affine map to the control polygon and then evaluate the image at t.

Affine invariance comes in handy when, say, we want to plot a rotated cubic curve $\dot{p}(t)$ by evaluating it at 100 points. Instead of rotating each of the 100 computed points and then plotting them, we can rotate the four control points, then evaluate 100 times and plot. Instead of 100 matrix multiplications we can do only four.

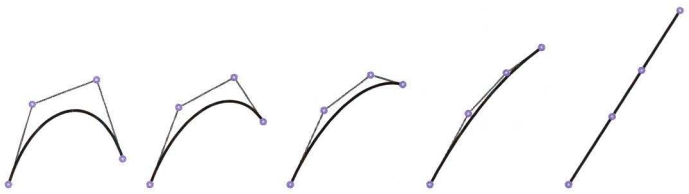
A big disadvantage of Bézier curves is that they are not projectively invariant, that is, they change in perspective. Of course, all CAD systems use perspective. Section 6.9.13 outlines *Non-Uniform Rational B-Splines* (NURBs), the main purpose of which is to ensure projective invariance.



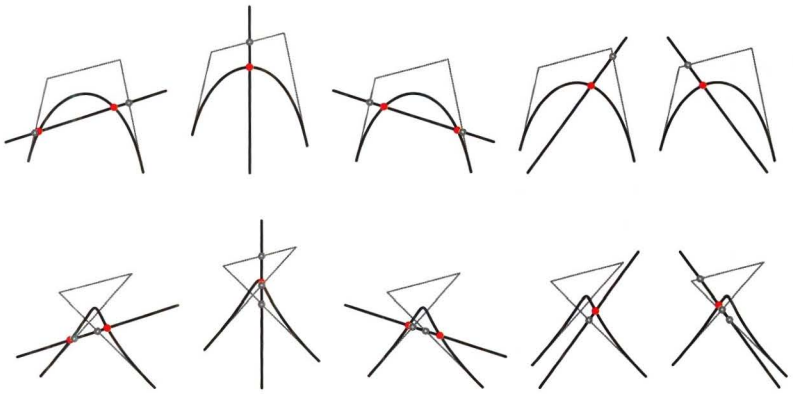
Invariance under affine parameter transformations. We usually consider Bézier curves defined on the interval $[0,1]$. However, we can also think of a Bézier curve as being defined on any interval $[p,q]$ with parameter s since by taking $t = \frac{s-p}{q-p}$, $p \leq s \leq q$, we convert the interval $[p,q]$ into $[0,1]$. In essence this means that any interval on the real number line can be used to control the parameter t. It just takes a little work.



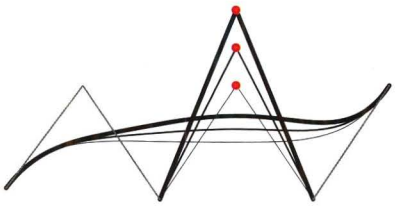
Linear precision. When the Bézier control points are collinear, the Bézier curve is a straight line.



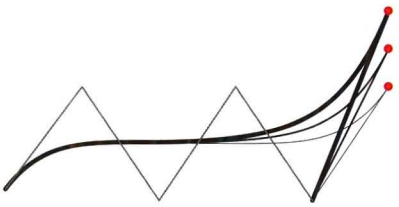
Variation diminishing. Any line intersects the control polygon of a Bézier curve at least as many times as it intersects the Bézier curve.



Pseudo-local control. The principle of local control means that moving any control point should move only the part of the curve "near" the control point. Local control is good – as counterpoint, think about editing a single point at the corner of a stadium roof and having the entire roof change as a consequence. Bézier curves fail to meet this principle, as all points (except for the endpoints) are affected by the movement of any given control point. They do implement local control in a partial sense: the "closer" to a control point that part of the curve lies, the more it is affected by movement of the control point. The quotes around the words "near" and "closer" signal their mathematical informality.



6.63: Moving an internal point on the control polygon moves all points except for the endpoints. Points parametrically closer to the control point move more than points further away, demonstrating the informal notion of pseudo-local control.



6.64: Moving an endpoint of the control polygon similarly moves all points on the curve, except the other endpoint.

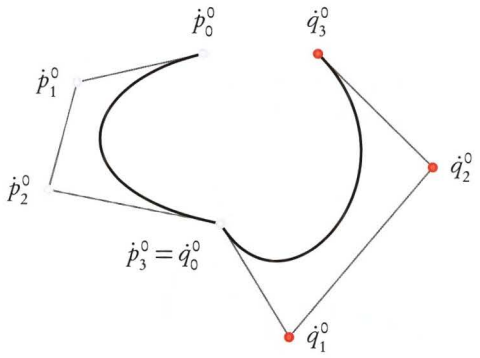
6.9.11 Joining Bézier curves

A *spline* is a composite curve in which the parts connect. Bézier curves can be joined together to make splines. Doing so reveals why Bézier curves are not normally used in applications where splines are required.

We are interested in joining together curves with various levels of *continuity* or *smoothness,* which two terms we treat qualitatively here. Recall that Section 6.9.7 introduces some of the basic ideas of continuity and smoothness.

Joining together two Bézier curves can be done in sequence of methods, with each member of the sequence increasing the smoothness of the result.

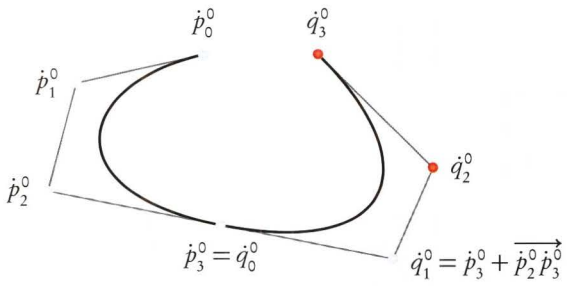
If two curves share the last and first control points respectively, they will join, since Bézier curves interpolate their endpoints, two connected control polygons will produce curves sharing endpoints, but there may be a "kink" where they join. Such splines have both C_0 and G_0 continuity.



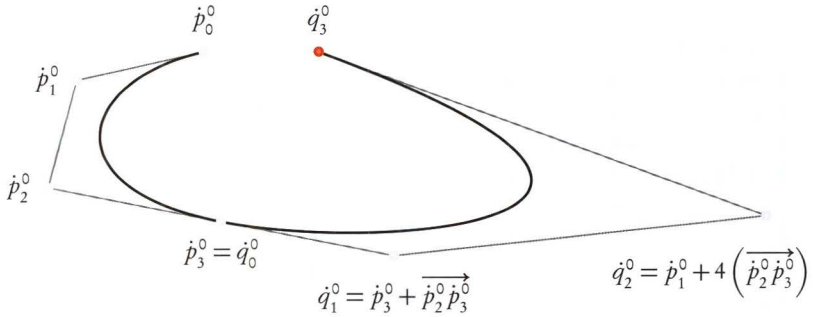
6.65: In C_0 continuity the control polygons of two Bézier curves share a single control point. Points $\dot{q}_1^0$, $\dot{q}_2^0$ and $\dot{q}_3^0$ (shown in eopdRed) can be freely moved.

If curves share the last and first control points and if the next control point on each control polygon and the joined point are collinear and equidistant from the joint, the Bézier spline will be C_1 continuous. It will be smooth across the join, but the distance between equal parametric points may suddenly change. A consequence is that, if you are joining one curve to another, two points are pre-determined.

In addition to the constraints of C_1 continuity, C_2 continuity requires that the second derivatives of the two Bézier curves be the same at the joining point. For the Bézier curves we have used so far, this condition has a surprising geometric result, as shown in Figure 6.67.



6.66: C_1 continuity constrains two points on each control polygon. Two points $\dot{q}_2^0$ and $\dot{q}_3^0$ (in red) remain free.



6.67: To join two curves $\dot{p}(t)$ and $\dot{q}(t)$ with C_2 continuity, the second derivative of their endpoints and startpoints must be equal. That is, $\dot{p}''(1.0) = \dot{q}''(0.0)$. This determines $\dot{q}_1^0$ and $\dot{q}_2^0$. Three of the four points on a degree 3 Bézier curve are determined when splining to C_2 continuity, leaving only $\dot{q}_3^0$ free (in red).

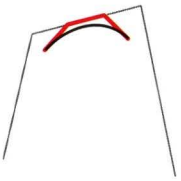
6.9.12 B-Spline curves

Bézier curves are geometrically and mathematically simple, but they have deep (and related) problems: control is only pseudo-local, and the order of a Bézier curve is the number of points in the control polygon. Pseudo-local control means that all points in a curve change when any control point is changed: it would be good to adjust the bow of a boat hull without affecting the stern. The link between order and control points means that a complex design must be done with high-order curves and this creates problems in interactive editing: it is easy to introduce local bumps and hard to make a curve visually fair. The curve also may lie far from the control polygon, making it hard to predict how an editing action might affect the curve. All of these problems can be remedied by connecting a series of curves into a composite curve, that is, a *spline curve*. Unfortunately, Bézier curves do not spline gracefully.

B-Spline curves address all of these problems. Like Bézier curves, B-Splines are defined on a control polygon. Unlike Bézier curves, they can be easily splined. In fact, splining is so natural that often no distinction is made between a curve (one spline segment) and a spline (multiple segements). The only constraint on the order of the curve is that it must be less than or equal to the number of points in the control polygon.

There are beautiful mathematical and computational ways to describe B-Splines (Rockwood and Chambers, 1996; Piegl and Tiller, 1997; Rogers, 2000; Farin, 2002) that provide great insight on how and why the curves work. But using B-Splines is the important thing here, and B-Splines provide two new modeling controls to the control points of Bézier curves: choice of *knots* and independent specification of order. The following explanation expands on the treatment of Bézier curves above to demonstrate B-Splines and their controls.

In essence, B-Spline curves are a framework for constructing Bézier curves – the B-Spline control polygon is just a new way to specify a Bézier control polygon that, in turn, defines the intended curve. This has profound implications for design – B-Splines and Béziers can model exactly the same possibilities; they just do it differently; see Figure 6.68.

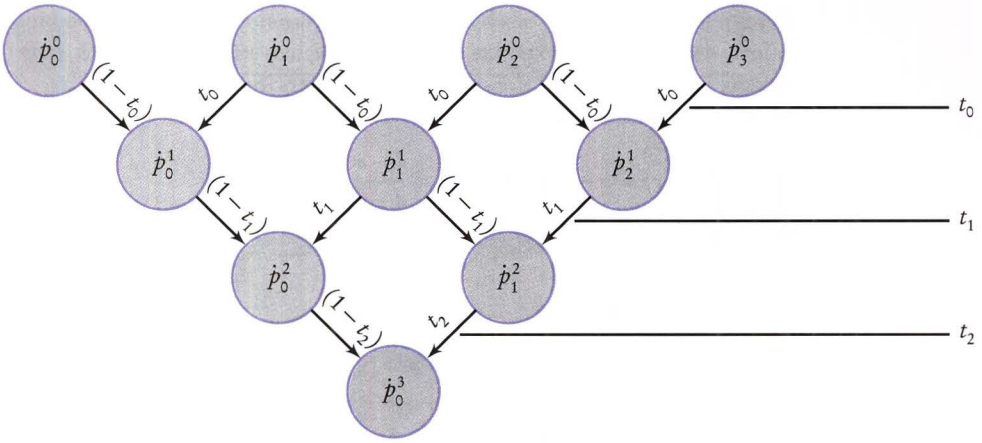


6.68: A B-Spline control polygon (in grey), the derived Bézier control polgyon (in red) and the resulting B-Spline (and Bézier) curve.

The data needed for an order n B-Spline curve are the same as those for a Bézier curve, with the addition of a *knot vector* comprising a non-decreasing sequence of real values. The knot vector not only determines the parameter values over which the curve is defined, but also affects the shape of the curve. Depending on the specific mathematical explanation, a knot vector of length k for an order n curve with p control points has either $k = p + n$ or $k = p + n - 2$ elements. The technique shown here uses the shorter knot vector, that is, having length $k = p + n - 2$; see Rogers (2000) for the longer form.

We demonstrate the B-Spline construction method (the deBoor algorithm) for an order 4 curve with four control points. This yields a single B-Spline curve segment.

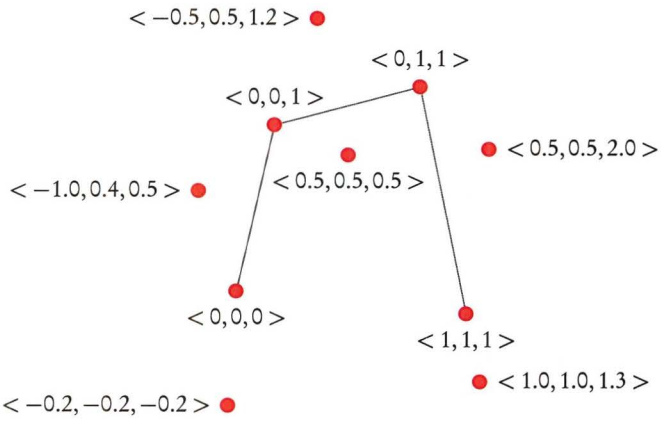
In essence, B-Splines generalize the deCasteljau algorithm to produce both the new Bézier control points and the curve from these new points. The first of two key ideas, shown in Figure 6.69, is that the parameter used at each level in the algorithm can be different. Instead of a single t, use a collection t_0, t_1, t_2. Thus, a point produced by the algorithm has not one parameter, but three: $\dot{p}(t_0, t_1, t_2)$.



6.69: The first step in generalizing the deCasteljau algorithm defines a different parameter t_i at each level of the systolic array.

Clearly, the point produced by the algorithm is freer than before.

Taken together, the t-values are called the *blossom values* of the point $\dot{p}(t_0, t_1, t_2)$ and are written as $\langle t_0, t_1, t_2 \rangle$. Thus the input to the algorithm is a blossom value $\langle t_0, t_1, t_2 \rangle$ and the output is a *blossom point* with the input blossom value $\langle t_0, t_1, t_2 \rangle$. Strangely, the order of the blossom values does not matter: $\langle 0, 1, 2 \rangle$ produces the same output as $\langle 2, 0, 1 \rangle$, or any other permutation of the values! To make it easy to distinguish blossom values producing different results, we make a canonical notation by sorting blossom values in non-decreasing order (numeric order for numbers and alphabetic order for variables).

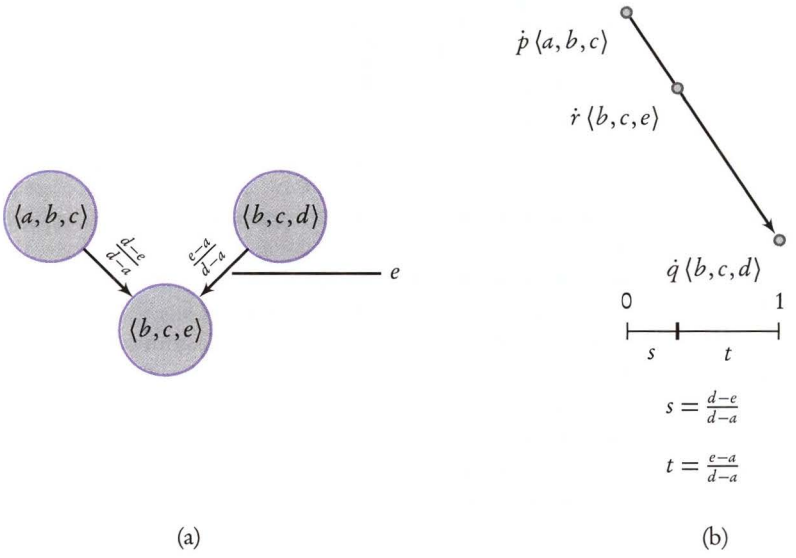


6.70: The location of blossom points $\langle a, b, c \rangle$ depends on the values given to a, b and c. Blossoms $\langle 0, 0, 0 \rangle$, $\langle 0, 0, 1 \rangle$, $\langle 0, 1, 1 \rangle$ and $\langle 1, 1, 1 \rangle$ correspond to the control points.

If two points share all but one blossom value in common, they can be combined to form a new point. Two points with blossom values $\dot{p}\langle a,b,c\rangle$ and $\dot{q}\langle b,c,d\rangle$ produce a third point $\dot{r}\langle b,c,e\rangle$ through a parametric line equation with an affine parameter transformation, as shown below.

$$\dot{r}\langle b,c,e\rangle = \dot{p} + \frac{e-a}{d-a}(\dot{p}-\dot{q}) = \frac{d-e}{d-a}\dot{p} + \frac{e-a}{d-a}\dot{q}$$

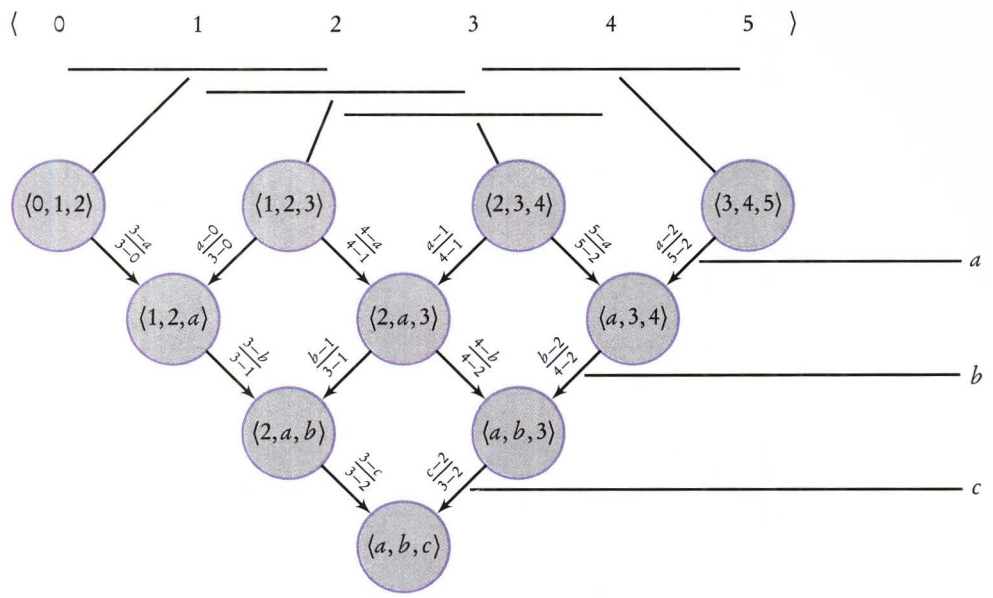
The point $\dot{r}$ lies on the line between $\dot{p}$ and $\dot{w}$.



6.71: Computing a point from blossoms that share two common values. (a) The labels on the nodes are blossom values. The labels on the arcs give the coefficients for each input point in computing the resulting point. (b) The resulting point lies on the line between $\dot{p}$ and $\dot{q}$.

The second key idea generalizes the algorithm one step further by assigning each of the points in the systolic array its own blossom value, and using those blossom values to determine how linear interpolation works between points. Here is where the main new control of the B-Spline comes into the picture. The *knot vector* is a non-decreasing sequence of real values, the most simple being $\langle 0,1,2,3,4,5\rangle$. Such a *uniform* knot vector has identical increments between each successive knot.

To use a knot vector, distribute three of its successive elements over each control point. To control point $\dot{p}_0^0$ assign knot vector elements $\langle 0,1,2\rangle$; to control point $\dot{p}_1^0$ assign elements $\langle 1,2,3\rangle$, and so on. In the general case, using knot vector k, assign $\langle k_i, k_{i+1}, k_{i+2}\rangle$ to control point $\dot{p}_i^0$. These are the *blossom values* of the control points. Note well that each pair of adjacent control points share two blossom values – *they can be combined using the above logic, and their result will share two blossom values with the original points as well!*



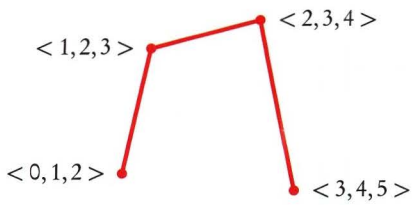
6.72: The blossom values $\langle a, b, c\rangle$ enter the algorithm, one at each level as the parameter in an affine parameter transformation using the unsharede blossom values as the bounds of the transformation. The blossoms, in turn, are defined by the knot vector.

At each layer in the graph, the algorithm uses the corresponding element from the input blossom value to compute the points and their blossom values at the next step. The algorithm is completed by building this equation into every primitive element of the systolic array. The output of the algorithm is a point with blossom value $\langle t_0, t_1, t_2\rangle$ given parameters t_0, t_1 and t_2.

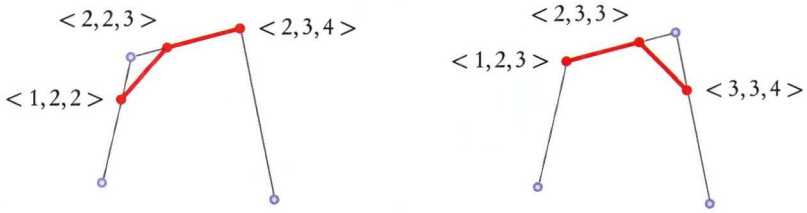
The middle two elements $\langle 2,3\rangle$ of the knot vector $\langle 0,1,\mathbf{2},\mathbf{3},4,5\rangle$ determine the parametric interval over which the implied Bézier curve will be defined. The Bézier control points are the blossom points with values $\langle 2,2,2\rangle$, $\langle 2,2,3\rangle$, $\langle 2,3,3\rangle$ and $\langle 3,3,3\rangle$.

Using the control points and their respective blossoms, the deBoor algorithm computes points on the curve by equating its three input arguments. That is, $\dot{p}(t) = \text{deBoor}(t,t,t), 2 <= t <= 3$. Of course, using a knot vector other than $\langle 0,1,2,3,4,5\rangle$ will change these bounds. For a single B-Spline segment the order is given by the number of control points, so the length of the knot vector is twice the curve degree $k = 2d$ or twice the curve order minus two, that is, $k = 2n - 2$. The n^{th} and $(n+1)^{th}$ elements of the knot vector determine the lower and upper curve parameters respectively.

Any point $\dot{p}(t)$ on the derived Bézier curve can be computed with either the deCasteljau algorithm over the derived control polygon or by using the deBoor algorithm with uniform knots: the result is the same. In turn, the deCasteljau algorithm over the derived control polygon is a special and simple case of the deBoor algorithm over these same points with blossom values $\langle 2,2,2\rangle$, $\langle 2,2,3\rangle$, $\langle 2,3,3\rangle$ and $\langle 3,3,3\rangle$ and bounds $\langle 2,3\rangle$.

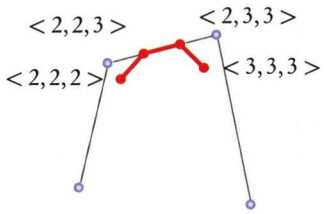


6.73: A B-Spline control polygon labeled with blossom values. This is the 0^{th} level of the deBoor algorithm.



6.74: The level 1 control points for $\langle 2,2,2\rangle$ (on the left) and $\langle 3,3,3\rangle$ (on the right). Each has its respective blossom values as computed by the first level of the deBoor algorithm.

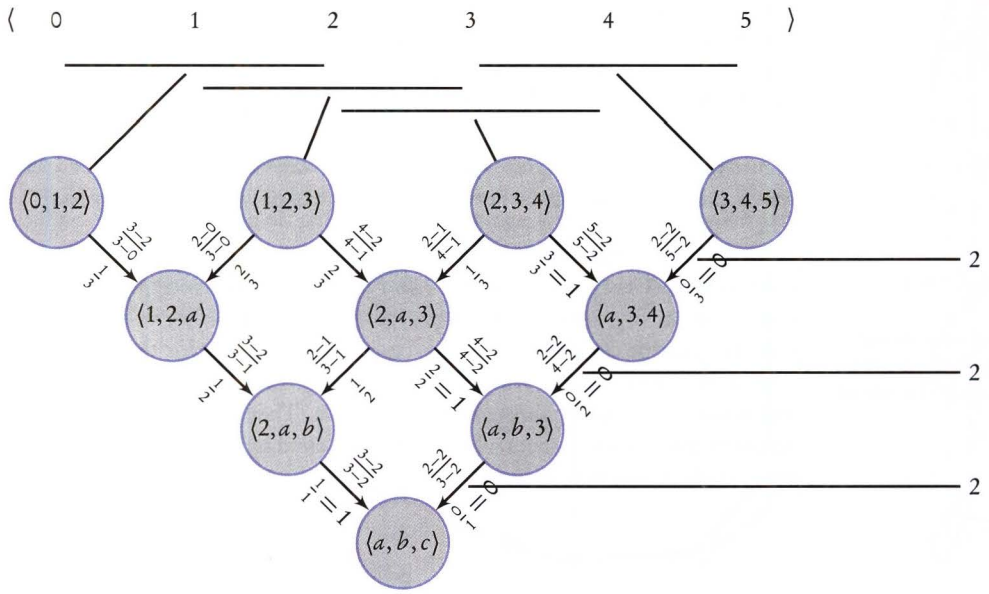
6.75: The level 2 control points for $\langle 2,2,2\rangle$ (on the left) and $\langle 3,3,3\rangle$ (on the right). Each has its respective blossom values as computed by the second level of the deBoor algorithm. The level 3 points are already computed at this stage as one of the level 2 control points.



6.76: A B-Spline control polygon, curve and the derived Bézier control polgyon.

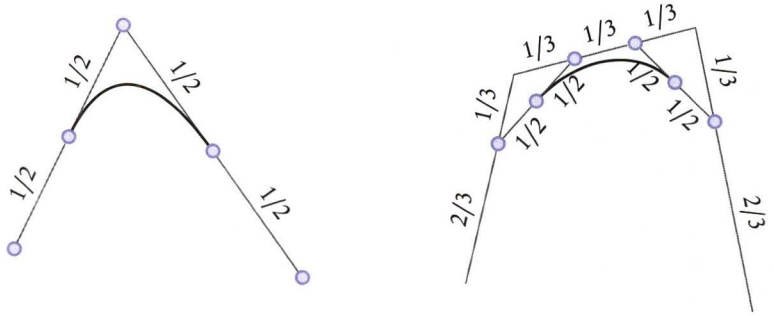
A look at the above figures and the deBoor algorithm shows more work being done than is strictly necessary to compute the internal Bézier control points. For instance, with inputs $\langle 2,2,2\rangle$, $\dot{p}\,\langle 2,2,3\rangle$ is computed at the first level of the algorithm, and $\dot{p}\,\langle 2,2,2\rangle$ at the second level. The algorithm though is general: it works to compute all control points, any blossom value and any point on the B-Spline curve.

An elegant shorthand for determining the Bézier control points and thus the B-Spline curve segment for the standard knot vector uses coefficients of the affine parameter transformation at each level of the deBoor algorithm. (The standard knot vector is $\langle 0,1,2,3\rangle$ for order 3, and $\langle 0,1,2,3,4,5\rangle$ for order 4 curve segments.) For example, when computing the point $\dot{p}\,\langle 2,2,2\rangle$ for an order 4 curve, the deBoor algorithm uses fractions $\frac{1}{3}$ and $\frac{2}{3}$ at the first level, and $\frac{1}{2}$ at the second level, as shown in Figure 6.77 below.

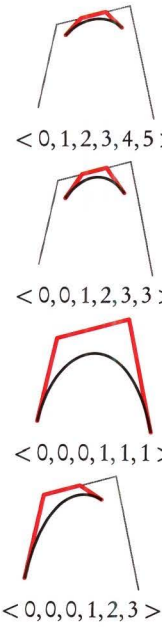


6.77: Computing the Bézier control points $\dot{p}\,\langle 2,2,2\rangle$, $\dot{p}\,\langle 2,2,3\rangle$, $\dot{p}\,\langle 2,3,3\rangle$ and $\dot{p}\,\langle 3,3,3\rangle$ with knot vector $\langle 0,1,2,3,4,5\rangle$ produces simple fractions at each level of the deBoor algorithm. Shown here is computation for $\dot{p}\,\langle 2,2,2\rangle$.

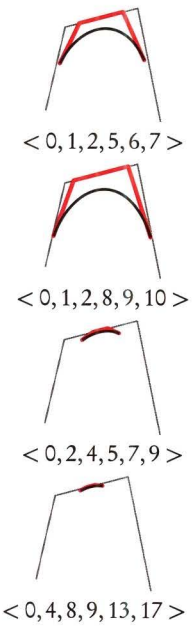
The Bézier control points can be directly drawn using these fractions as follows.



6.78: The Bézier control points for order 3 and order 4 B-Spline curve segments, drawn using fractional proportions from the deBoor algorithm.



6.79: Repeating knots at the end of a knot vector pulls the curve towards the endpoints of the control polygon.



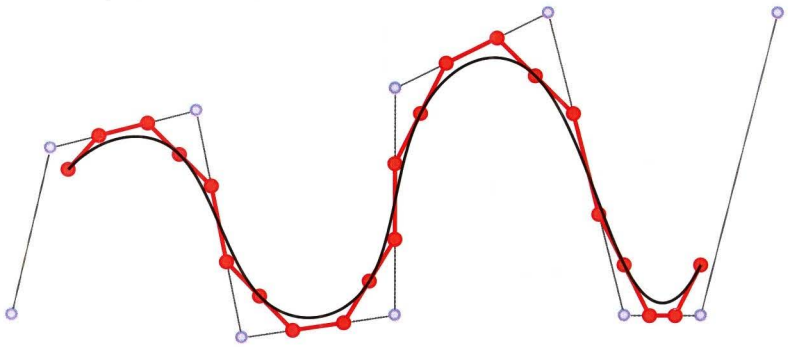
6.80: Increasing the relative spread between central knot values pulls the curve towards the control points.

The knot vector itself provides a control for B-Splines. Using it, B-Splines can be forced to interpolate their control polygon endpoints, moved and changed within the control polygon and joined with different degrees of continuity.

When values are repeated at the ends of a knot vector, as in Figure 6.79, the curve is "pulled towards" the ends of the control polygon. For a single curve segment (the number of control polygon vertices k and the curve order n are the same), when the knot vector repeats $n-1$ values at both beginning and end, a B-Spline becomes a Bézier curve. In the contemporary curve literature such curves are described as being *clamped*. A very confusing historical fact is that they were called *open* curves by Rogers (2000) but now *open* generally means the opposite of clamped! Repeating values at one end and not the other leaves the other endpoint of the curve unchanged.

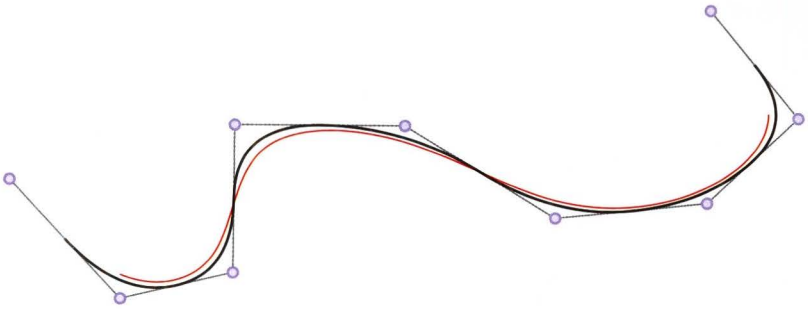
Figure 6.80 shows that "spreading" knot vector values in the centre of the vector moves the curve towards the bottom of the control polygon (and vice versa).

At first glance, B-Splines seem a rather awkward way to compute the Bézier curves of which they are composed. After all, Bézier curves interpolate their endpoints, the first and last control polygon segments directly give the endpoint tangents, and the curve is closer to the control polygon than for the B-Spline. The benefit becomes clear when curves spline together. B-Splines connect easily and maintain continuity through the connection. They join curve pieces into an entire spline with the control points being shared by adjacent curves. This is more easily drawn than written.



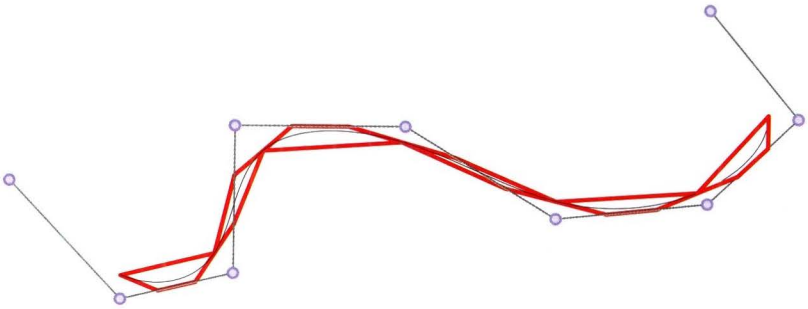
6.81: A B-Spline control polygon; its Bézier structures; and the B-Spline curve.

The order (and thus degree) of the curve simply determines how many points of the control polygon to use for each segment. For example, an order 4 B-Spline uses four control points per piecewise curve. Each of the points contributes to the location of every point on each curve segment in which it participates, but has no effect on segments in which it does not participate. Figure 6.83 shows successive B-Spline segments for a multi-segment curve. Of course, the curve is affected by the choice of order. With decreasing order the curve moves closer to the control polygon. When order equals 2, the curve is the control polygon: each piece of the curve is given by a simple parametric line equation.



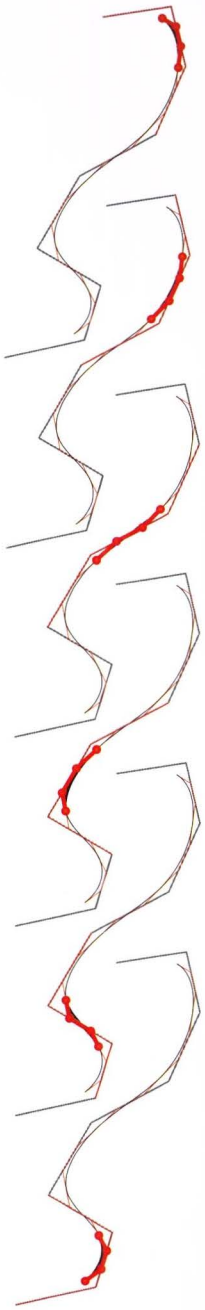
6.82: Order 2, 3 and 4 curves from the same control polygon. An order 2 curve is the control polygon. As the order increases from order 3 (black) to order 4 (red), the curve moves away from the control polygon and generally varies less.

B-Splines inherit all of the properties of Bézier curves and strengthen two. First, Figure 6.84 shows that the convex hull condition is much stronger. Whereas a Bézier curve lies within the convex hull of its control polygon, a B-Spline lies piecewise within the convex hull of its implied Bézier control polygons.

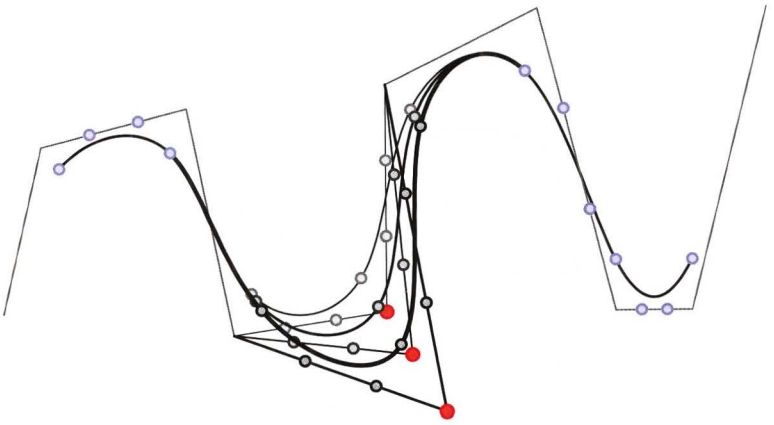


6.84: B-Spline curve segments lie within the convex hull of their implied Bézier control polygons. This allows for rapid approximate tests for likely intersections between the curve and other objects.

Second, B-Splines demonstrate true local control. Figure 6.85 demonstrates that moving a vertex of the control polygon for an order n curve affects at most the n curve segments whose control polygons use that vertex. When the vertex is close to the end of the control polygon even fewer segments are affected.

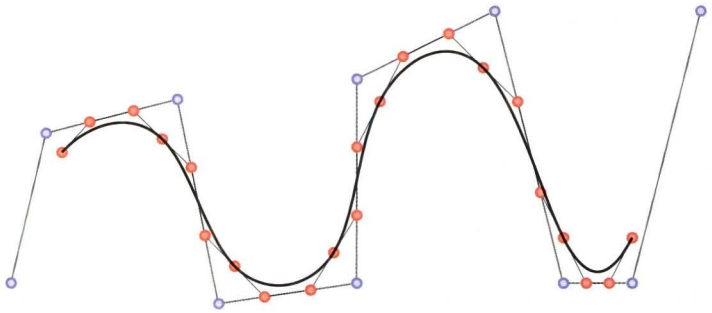


6.83: A B-Spline control polygon, implied Bézier control polygon and curve segment for each of the piecewise components of a B-Spline curve.

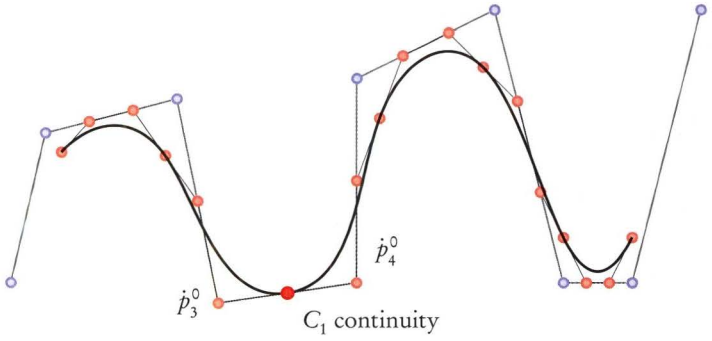


6.85: When a single control point (in eopdRed) of an order 4 B-Spline curve moves, only the four parts of the curve using that control point are affected. B-Splines implement true local control.

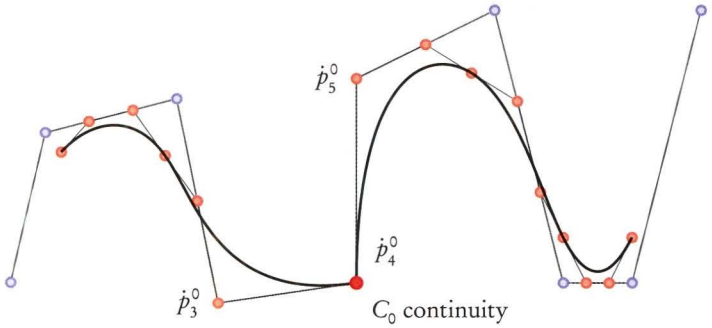
As described above, B-Splines introduce knots as a new control. Figure 6.86 shows that repeating knots internal to a B-Spline reduces the continuity at the affected vertex.



(a) Uniform knot vector $\langle 0,1,2,3,4,5,6,7,8,9,10,11\rangle$

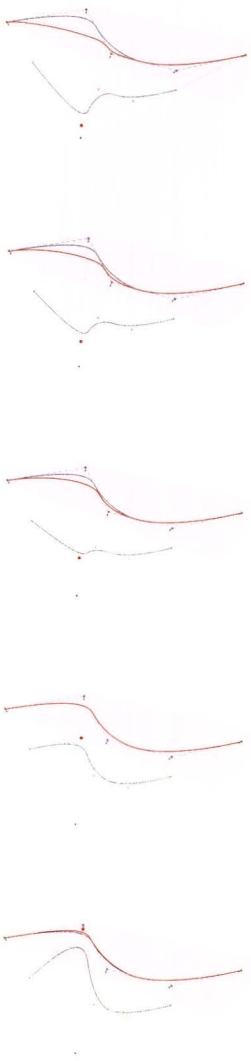


(b) Non-uniform knot vector, with two duplicate internal knots
$\langle 0,1,2,3,4,4,5,6,7,8,9,10\rangle$



(c) Non-uniform knot vector with three duplicate internal knots
$\langle 0,1,2,3,4,4,4,5,6,7,8,9\rangle$

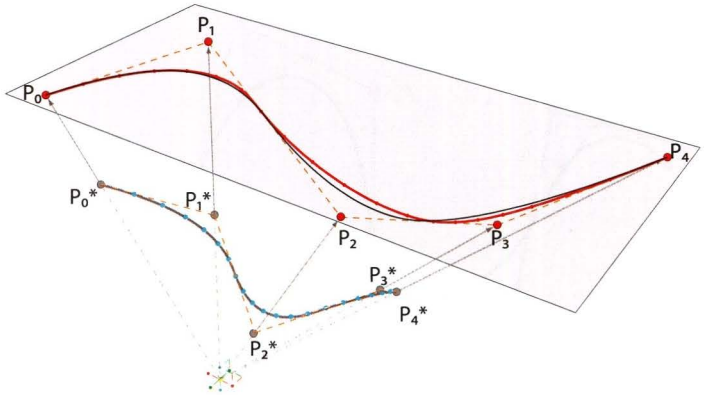
6.86: (a) In this order 4 B-Spline, non-duplicate knots guarantee C_2 continuity. (b) This non-uniform knot vector duplicates the fifth and sixth knots. This changes the blossom values at control polygon vertices $\dot{p}_3^0$ (blossom $\langle 3,4,4\rangle$) and $\dot{p}_4^0$ (blossom $\langle 4,4,5\rangle$) and reduces continuity between the Bézier curve segments 2 and 4. Bézier curve segment 3 has four identical control points, so essentially disappears. (c) A third duplicate knot spreads the change in blossom value to a third vertex: now $\dot{p}_3^0$, $\dot{p}_4^0$ and $\dot{p}_5^0$ have blossoms $\langle 3,4,4\rangle$, $\langle 4,4,4\rangle$ and $\langle 4,4,5\rangle$ respectively. This reduces continuity to C_0 between Bézier curve segments 2 and 5; and makes curve segments 3 and 4 have all identical vertices.



6.88: Increasing the z-coordinate of a B-Spline control point in three-dimensional space moves the two-dimensional NURB curve towards the two-dimensional projection of the control point. The z-coordinate of a control point of three-dimensional B-Spline is the weight of its corresponding point in the two-dimensional NURB. When weights are equal as, in the second from bottom curve, the two-dimensional projection of the B-Spline and the two-dimensional NURB are the same.

6.9.13 Non-uniform rational B-Spline curves

Curves have one more control: *weights*, which are introduced in the step from B-Splines to *Non-Uniform Rational B-Splines* (NURBs). CAD system interfaces and marketing literature feature the word "NURB" as if it were some kind of magic. Some design literature goes even further, attributing high meaning to the term "non-rational". Reality is both more plebian and essentially below design. NURBs exist so that curves control polygons can be taken through a perspective projection and the curve computed afterwards. To do this, NURBs define *weights*. Mathematically NURBs are specified in a space one dimension higher than the geometric space in which they are embedded. The weights are the highest dimension coordinates of the control points in that space. In design terms, weights manifest as controls that draw a curve closer to a control point as the weight on that point is increased. Many CAD systems do not even provide access to either weights or knots. Such systems may claim NURB capability and be based on NURBs underneath the interface, but they essentially provide only B-Splines. NURBs do have one geometrically important feature. With the correct choice of weights, they can represent conic sections, a task B-Splines cannot do. For CAD systems this means that only the NURBs representation is needed. From a design perspective this matters much less.



6.87: NURBs in two-dimensional space are B-Splines in three-dimensional space. The two-dimensional NURB (rendered in red), with control points P_0, P_1, P_2, P_3 and P_4, corresponds to a three-dimensional B-Spline (rendered in grey) with control points P_0^*, P_1^*, P_2^*, P_3^* and P_4^* when the z-coordinates of the B-Spline control points are equal. Otherwise, as is the case shown here, the NURB varies from the B-Spline.

Weights complete the lexicon of curve properties and controls. Béziers, B-Splines and NURBs form a sequence, each building on its predecessor. Here is how they compare.

Property	Bézier	B-Spline	NURB
convex hull	yes	yes	yes
symmetry	yes	yes	yes
endpoint interpolation	yes	optional	optional
affine invariance	yes	yes	yes
affine parameter invariance	yes	yes	yes
variation diminishing	yes	yes	yes
local control	pseudo	yes	yes
splining with continuity	hard to do	yes	yes
order control	no	yes	yes
knots	no	yes	yes
projective invariance	no	no	yes
conic sections	no	no	yes
weights	no	no	yes

From a design perspective, what is most striking with all of these properties is their relative irrelevance. Yes, we rely on each of these properties sometimes. For instance, affine invariance is important. As we move control points around as a group the generated curve does not change with respect to the control points. But the generic curve concepts are what count. That parametric and geometric distances differ, that the Frenet frame is (almost) always defined and that we want to control continuity are more important to design. Béziers, B-Splines and NURBs are the (not so simple) mathematical devices we need to get there.

6.9.14 The rule of four and five

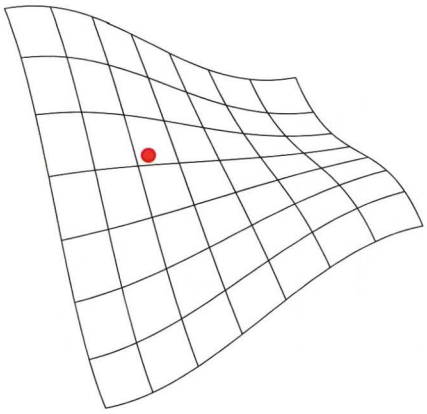
How many control points are actually needed? What is a good choice for order? These are separate questions, but with linked answers. There are good reasons to keep each number small, and it turns out that just five control points and order 4 is sufficient for many, many modeling tasks. Five control points means that a curve can have a "dip" in it. Order 4 means that curves can join smoothly, without obvious joints, even under light reflection. Having a small number of control points makes it easier to predict how a model will change. The lower the order, the closer the curve is to the control polygon, and this also helps in understanding a model's behaviour.

6.89: An order 4 curve with five points on its control polygon allows a single "dip". This simple and computationally light description is sufficient for many design situations.

6.10 Parametric surfaces

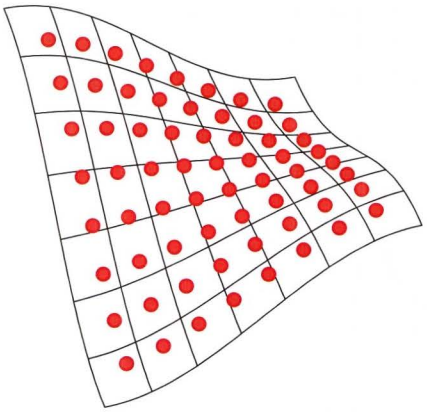
Parametric surfaces and curves share mathematical structure. Surfaces are more complex than curves so, naturally, their representation must be more involved. Rather than describe how surfaces work mathematically and parametrically, this section describes their behaviour from a modeling perspective.

Parametric surfaces comprise a point $\dot{p}(u, v)$ that moves along the surface as the parameters u and v change. (See Figure 6.90.) By convention curves have a parameter t, so surfaces get the next two letters of the alphabet.



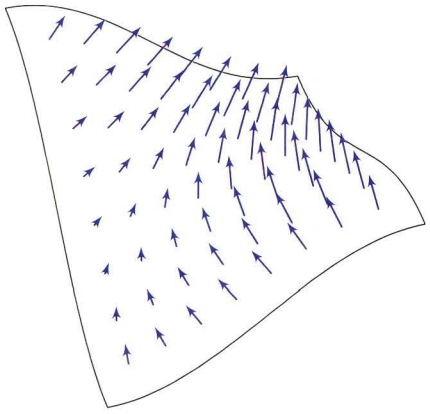
6.90: A uv point on a surface $\dot{p}(u, v)$.

Like curves, movement is not linear. (See Figure 6.90.) Unlike curves, there is no general way to make spacing uniform. This leads to many hard problems in subdividing surfaces.



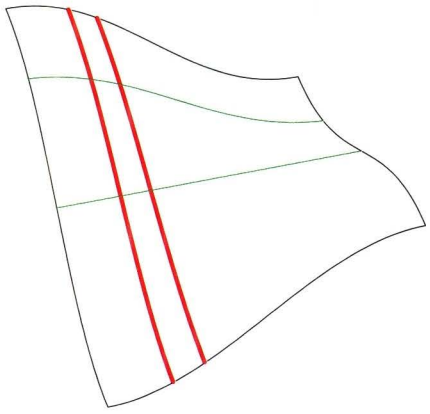
6.91: An array of parametrically equally spaced uv points on a surface. It is easy to see that the geometric spacing varies between pairs of points.

With exceptions similar (but more complex) to those for curves, Figure 6.92 shows that every point on a surface has a unique unit surface normal.



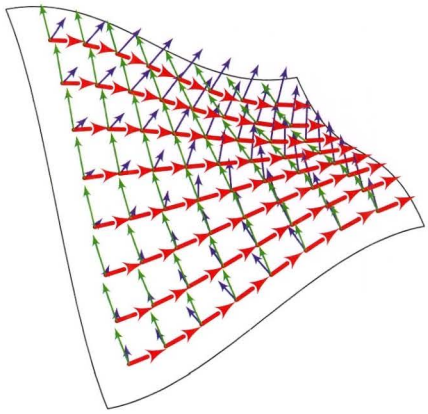
6.92: The unit surface normals at the uv points on the surface from Figure 6.90.

Lines in u and v parameter space map to curves on the surface. When either u or v is held constant, the line in parameter space is parallel to the parameter axes. The square uv parameter space is mapped to the surface, stretching like a rubber sheet in the process. The curve in geometry space stretches with the sheet, so lies on the surface in rough proportion to its position in parameter space. Such curves, where one of u or v is held constant, are called *isocurves*. Figure 6.93 shows four such curves.



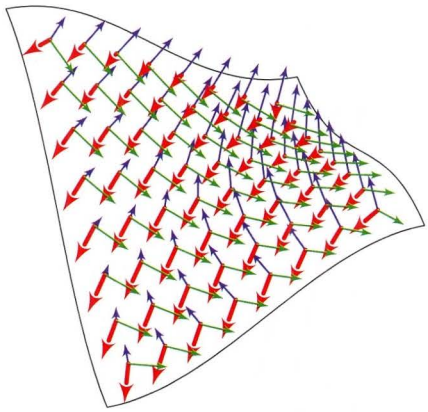
6.93: Isocurves with $u = 0.2$, $u = 0.3$, $v = 0.5$ and $v = 0.8$ on the surface from Figure 6.90.

At almost every point on a surface there is a coordinate system comprising the surface normal (z-axis), a vector in the local u-direction (x-axis) and a vector in the local v-direction (y-axis). Such a system is called a uv-coordinate system. Figure 6.94 renders an array of such systems at equal parametric intervals.



6.94: An array of uv-coordinate systems on a surface.

There is another coordinate system as well, shown in Figure 6.95. This points not along the uv isocurves but along the *lines of principal curvature*. At almost every point on a surface (excepting oddities like the sphere and the plane), there exist two planes at right angles to both the plane of the surface normal and to each other. Both planes intersect the surface. One holds the curve of maximum curvature; the other the curve of minimum curvature. The directions of these planes are called the *principal directions* of the surface at the point.



6.95: An array of principal direction coordinate systems on a surface.