11

Parametric curves

Curves as *point loci* with a specific shape are often identified by a proper name, such as *circle*, *parabola*, *hyperbola*, *spiral*, *helix*, etc. For applications of computer-aided design, some classes of *free-form curves* are more interesting, because they are able to satisfy both geometric and esthetic constraints set by designers, depending on the shape design problem at hand. Even more useful, *splines* are piecewise-continuous composite curves used to either interpolate or approximate a discrete set of points. Various different representations of curves are in use. The present chapter is dedicated to discussing the *parametric* representation of *polynomial* and *rational* curves and splines, i.e. the main classes of free-form curves used by CAD applications, and also introduces a straightforward implementation of parametric curves and splines with `PLaSM`, together with several examples.

11.1 Curve representations

Curves may be represented by using different kinds of equations. In particular, it is possible to distinguish representations:

1. *explicit* or Cartesian, where the curve is given as the graph of a function;
2. *implicit*, as the zero set of one or more global algebraic equations;
3. *parametric*, associated with a vector function of one parameter;
4. *intrinsic*, where points locally satisfy differential equations.

Explicit representation

An explicit or Cartesian representation of a plane curve is the graph $(x, f(x))$ of a function $f : \mathbb{R} \to \mathbb{R}$. We may also write, in this case:

$$y = f(x).$$

This representation is not particularly diffuse nor useful in geometric modeling, because it is unusable for closed curves nor, more in general, for curves where more than one value of the dependent variable, say, y, is associated with the same value of the independent variable, say, x. Also, it does not easily support affine transformations.

Geometric Programming for Computer-Aided Design Alberto Paoluzzi
© 2003 John Wiley & Sons, Ltd ISBN 0-471-89942-9

Example 11.1.1 (Cartesian equation of a line)
A simple example of explicit representation in the 2D plane is the well-known Cartesian equation of the line:

$$y = mx + c,$$

where m is called the *angular coefficient* and coincides with the tangent of the angle that the line creates with the x axis, and c is the ordinate of the intersection point between the line and the y axis. According with the previous remark, this representation cannot be used for vertical lines, for which we must conversely use the equation $x = a$.

Implicit representation

The *implicit representation* denotes a plane curve as the locus of points that satisfy an equation, usually algebraic, of type:

$$f(x, y) = 0.$$

The simplest example of a plane curve is given by the implicit representation of the 2D line:

$$ax + by + c = 0.$$

A single equation in three variables instead denotes a surface in three-dimensional space, where a curve is given as the set of simultaneous solutions of two linear equations. More in general, a curve in n-dimensional space is the solution set of $n - 1$ algebraic equations of arbitrary degree. Its existence is always guaranteed only in a complex space.

Example 11.1.2 (Conics curves)
Another example of implicit representation is given by the curves of second degree, as the loci of points which satisfy the general equation

$$ax^2 + by^2 + cxy + dx + ey + f = 0.$$

Such curves are called *conics* because they describe the possible intersections of an indefinite 3D cone with the plane $z = 0$. In particular, the intersection is:

1. a *circle*, when such a plane is perpendicular to the cone axis;
2. an *ellipse*, when it crosses all the cone lines while cutting only one cone sheet;
3. a *parabola*, when the cutting plane is parallel to one line of the cone;
4. a *hyperbola*, when the plane is parallel to two such lines.

Such an intersection may also result in a (double) line or in two lines, in particular when the $z = 0$ plane contains one or two lines of the cone.

Degrees of freedom The number of coefficients, called *degrees of freedom*, of the implicit equation and hence the number of different geometric objects it may represent, grows rapidly with the degree of the equation. As a consequence, the set of lines is a subset of conics, and conics are a subset of curves of degree three, and so on.

Point-set membership The implicit equation of a plane curve enjoys the notable property of partitioning the plane into three subsets of points. They are the set of curve points, and two subspaces at the two sides of the curve, respectively corresponding to points that substituted into the curve equation and return either a positive or a negative number. Hence, to classify a point with respect to an implicit curve is a simple and efficient operation. It will suffice to substitute the point coordinates for the variables in the curve equation and compute the algebraic result. If this is zero, then the point belongs to the curve; otherwise, it belongs to one of the subspaces defined by the curve, depending on the resulting sign. The same property obviously arises for algebraic equations with three or more variables.

Meaning and sensitivity of coefficients A negative aspect of implicit representation arises from small geometric significance of the coefficients of the implicit equations. Except in a few cases,[1] they do not have any explicit geometric meaning. Furthermore, there often exists a strong numerical sensitivity of the curve shape on even a small variation of some coefficients. In other words, a small variation of some coefficients of an implicit equation may produce a large and completely unpredictable variation of the curve geometry.

Parametric representation

With a *parametric* representation, a curve $\boldsymbol{c}$ is given as a *point-valued map* of a single real parameter:

$$\boldsymbol{c} : D \to \mathbb{E}^n \quad \text{such that} \quad \boldsymbol{c}(u) = \boldsymbol{o} + \begin{pmatrix} x_1(u) & \dots & x_n(u) \end{pmatrix}^T$$

where $\boldsymbol{o}$ is the origin of the reference frame, and $D \subset \mathbb{R}$ often coincides with the standard real interval $[0, 1]$. When n is either two or three, the curve is said to be a plane or a space curve, respectively. The component functions $x_i : \mathbb{R} \to \mathbb{R}$, $i = 1, \dots, n$, are called the *coordinate maps* of the curve. The curve as a locus of points should be properly called the *image* of the curve. We would like to remind the reader that several differential concepts about parametric curves were already introduced in Chapter 5.

Polynomial and rational curves When the coordinate maps are polynomial functions of a single variable the curve is called a *polynomial* (parametric) curve. When the coordinate maps are rational functions, i.e. ratio of polynomials, the curve itself is said *rational*. Both polynomial and rational curves are liked by shape designers, mainly because their coefficients have a precise geometric meaning and assess specific esthetic and geometric constraints on the generated free-form curves. By editing such coefficients, normally by using very simple and intuitive graphics tools, the designer is given strong control over shape generation. Metodology and algorithms to execute automatic parametrization of several classes of rational curves and surfaces from implicit representations is given in papers [BA87a, BA87b, BA88, BA89] by Bajaj and Abhyankar.

[1] E.g., in the linear equation in three variables, three of the coefficients are proportional to the "director cosines" of the plane, i.e. to the cosines of the angles between the represented plane and the coordinate axes.

Polynomial and rational splines It is necessary to distinguish between polynomial or rational *curves* and the so-called *splines* of the same kind. The former are single point-valued maps. The latter are composite maps, which can be thought of as piecewise-continuous curves, obtained by joining more curve segments with some proper degree of continuity at the extreme points. In particular, when we need to interpolate or approximate a long sequence of points, it is normally more useful to employ some low-degree spline, rather than using a single high-degree curve with sufficient degrees of freedom. The second alternative, other than requiring higher computational costs, may in fact generate unpredictable oscillations of the produced point set.

Intrinsic representation

Intrinsic representations are mainly used in differential geometry, and describe the curve locally, i.e. independent of its position and orientation with respect to some external reference frame. In particular, an *intrinsic representation* is a pair of relations

$$\begin{aligned}\frac{1}{\rho} &= f(s)\\ \tau &= g(s)\end{aligned}$$

which express the *curvature* $\frac{1}{\rho}$, where ρ is the radius of curvature, and the *torsion* τ as functions of *curvilinear abscissa* s, i.e. of the length of the curve arc. The *natural equation* of a curve is any equation

$$h\left(\frac{1}{\rho}, \tau, s\right) = 0$$

between curvature, torsion and arc length.

A natural equation characterizes a whole class of curves. For example, *plane curves* have the natural equation

$$\tau = 0.$$

Analogously, *straight lines* have the natural equation

$$\frac{1}{\rho} = 0.$$

Two natural equations determine a curve completely, except for position and orientation. This kind of representation of curves has only limited utility in geometric modeling applications, because in this case we are mainly interested in characterizing a curve as a subset of points of an affine space, showing where to apply geometric transformations such as translations and rotations, and also relative to positioning, intersection, blending, etc, of two of more curves.

11.2 Polynomial parametric curves

The use of parametric curves in graphics and CAD started at the end of the 1960s, first within the aerospatial and automotive industries, both in Europe and in the USA, and later in academia, where applied mathematicians were studying mathematical methods for computer-aided geometric design, actually by simulating the hand-crafting of physically modeled curves and surfaces. Previously, various instruments were used in the industry for this purpose, and in particular some mechanical and graphical “ad hoc” tools, from pantograph to a flexible ruler called a *spline*, whose name was soon extrapolated to denote the mathematical interpolation/approximation of sequences of points.

Properties of parametric curves

We summarize below the main benefits of using a parametric representation of curves, by discussing some useful properties of such methods.

Control handles A polynomial parametric curve may be seen as a linear combination, with vector coefficients, of the elements of a suitable polynomial basis.[2] The vector coefficients of such a combination normally have some geometric meaning useful for shape design and editing. In particular, depending on the choice of the polynomial basis, they may correspond either to points interpolated by the curve, or to approximated points, or to tangent vectors, etc. Such vector coefficients are collectively called *geometric handles* or *control handles*, since they are often implemented as handles to be dragged by the designer when interactively editing the shape of the curve.

Multiple points When using explicit equations, it is not possible to represent curves where the same value of the independent variable is associated with multiple values of the dependent variable. No problems of this kind arise with parametric equations, which may easily represent curves which are self-intersecting, even more times. The presence of double or multiple points is not difficult to handle because every point instance is associated with a different value of the generating parameter. Conversely, when using implicit equations, such multiple points must be considered singular points, and a curve with singularities must be reduced to a set of curve segments without singularities.

Affine and projective invariance A polynomial or rational parametric curve may be translated, rotated or scaled by just translating, rotating or scaling its geometric handles. The same property, called *affine invariance*, holds for polynomial and rational splines. Furthermore, a rational curve or spline can be projected by simply projecting its handles. The points of the projected curve can be later computed by just combining with proper rational functions the projected handles. Unfortunately, such a *projective invariance* property does not hold for polynomial curves and splines. The projective invariance is one of main reasons for the great success of rational splines like NURBS (see Section 11.4.2) in graphics libraries.

[2] Let us remember that the set of polynomials of degree less or equal to n is a vector space of dimension $n + 1$.

Local and global control The vector parameters of polynomial and rational curves are used for curve input and control, since they perfectly match the implementation of interactive tools for definition and editing of shape. Such control of shape is *global* for curves, because in this case every variation of one of vector coefficients (control handles) induces some variation of the whole curve point set. Conversely, for splines, shape control is *local*, because a variation of a control handle induces a variation of the only subset of curve segments which are influenced by the changed point or vector, whereas the images of other curve segments do not change. Such local control of shape is very important to the designer, who often wants to make only small local changes to a design profile whose shape otherwise is quite satisfactory to design constraints and/or esthetic criteria.

Variation diminishing The possibility of unpredictable shape oscillations between interpolated points strongly increases with the growth of the degree of the interpolating polynomial. Some of the polynomial curves studied in this chapter, and in particular the Bézier curves, give a full control of this aspect, because they provide the so-called *variation diminishing* property. In particular, this property guarantees that the number of intersections of a curve with a hyperplane of its embedding space — i.e. a line in case of plane curves — is either less or equal to the number of intersections of the hyperplane with the polygon of control handles, also called the control polygon. In other words, this property states that a curve closely resembles its control polygon, which can be considered a linear approximation of the curve shape.

Continuity order and degree elevation A polynomial curve of degree n depends on $n+1$ vector parameters, and has non-zero continuous derivatives until the order $n-1$. If a curve is needed as a vector function with some fixed order of continuity, then it is sufficient to raise its degree as needed. The method for *degree elevation* of a polynomial curve, at least in the Bézier form, is quite simple. See the book by Farin [Far88]. Another reason to elevate the degree of a polynomial curve may arise when generating surfaces from curves. Several methods used for this purpose require the various input curves be of the same degree.

11.2.1 Linear curves

Let us start by discussing the polynomial curves of first degree. The matrix approach we use here, due to Jim H. Clark,[3] will be straightforwardly extended in later sections to polynomial curves of higher degree.

Algebraic form

We know that the line segment between points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ can be written, in any $\mathbb{E}^d$, as:

$$(\boldsymbol{p}_2 - \boldsymbol{p}_1)u + \boldsymbol{p}_1, \qquad u \in [0,1],$$

[3] The founder of SGI and Netscape Corporations.

i.e. as a polynomial with vector coefficients in the u indeterminate.

In general, let consider the polynomial curve of first degree, said *in algebraic form*:

$$\boldsymbol{C}(u) = \boldsymbol{a}u + \boldsymbol{b}$$

and write it as a product of matrices:

$$\boldsymbol{C}(u) = \begin{pmatrix} u & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{pmatrix} = \boldsymbol{U}_1 \ \boldsymbol{M}_1 \tag{11.1}$$

The row matrix $\boldsymbol{U}_1$ contains the elements of the standard *power basis* for the vector space $\mathbb{P}^1[\mathbb{R}]$ of polynomials of degree less or equal to 1, with coefficients in $\mathbb{R}$. The elements of this basis are u^1 and u^0, i.e. u and 1, respectively. Notice that every other polynomial in $\mathbb{P}^1$ can be expressed as a linear combination of such basis elements.

Geometric form

The matrix equation (11.1) contains two vector degrees of freedom, i.e. two "free" vector coefficients $\boldsymbol{a}$ and $\boldsymbol{b}$. In order to specify a given curve, we may specify two vector constraints to be satisfied by $\boldsymbol{a}$ and $\boldsymbol{b}$. We can, for example, require that the curve interpolates, at the extreme values of the parameter, two given points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$:

$$\begin{aligned} \boldsymbol{C}(0) &= \boldsymbol{p}_1; \\ \boldsymbol{C}(1) &= \boldsymbol{p}_2. \end{aligned}$$

By substituting 0 and 1 for u in equation 11.1, we get, respectively:

$$\boldsymbol{p}_1 = \begin{pmatrix} 0 & 1 \end{pmatrix} \boldsymbol{M}_1$$

$$\boldsymbol{p}_2 = \begin{pmatrix} 1 & 1 \end{pmatrix} \boldsymbol{M}_1$$

The above vector equations can be collected into the following matrix equation:

$$\begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \boldsymbol{M}_1,$$

from which we have

$$\boldsymbol{M}_1 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \end{pmatrix},$$

and, by substitution into equation 11.1, we get

$$\boldsymbol{C}(u) = \begin{pmatrix} u & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \end{pmatrix} = \boldsymbol{U}_1 \ \boldsymbol{B}_1 \ \boldsymbol{G}_1. \tag{11.2}$$

Equation 11.2 is called the *geometric form* of the polynomial curve of first degree. In general, there can be several different geometric forms, corresponding to different sets of constraints, usually boundary conditions, on the coefficients of the algebraic form.

Terminology Let us compare some different ways of writing the parametric equation of a polynomial curve. Equation 11.1 is called the *algebraic form* of the curve. Conversely, equation 11.2 is called the *geometric form* of passage through two given points. In particular, the matrix $\boldsymbol{B}_1$ is called the *basis matrix*, whereas the vector of points $\boldsymbol{G}_1$ is called the *geometry vector* or tensor of *control handles* of the geometric form. Finally, the vector of polynomial functions given by

$$\boldsymbol{U}_1\boldsymbol{B}_1 = \begin{pmatrix} u & 1 \end{pmatrix}\begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1-u & u \end{pmatrix} = \begin{pmatrix} b_1(u) & b_2(u) \end{pmatrix}$$

defines the *polynomial basis* of the geometric form, i.e. its basis functions, called also *blending functions* of the curve. The reason for last name is quite obvious, when considering the generic curve point $\boldsymbol{C}(u)$ written as a combination of blending function with control handles, i.e. as a sort of "blend" of such data:

$$\boldsymbol{C}(u) = \sum_{i=0}^{1} b_i(u)\ \boldsymbol{p}_i.$$

11.2.2 Quadratic curves

The approach discussed above is straightforwardly extended to polynomial curves of higher degree. For quadratic curves, we have what follows.

Algebraic form

The algebraic form of the polynomial parametric curves of second degree depends on three free vector parameters:

$$\begin{aligned} \boldsymbol{C}(u) &= \boldsymbol{a}u^2 + \boldsymbol{b}u + \boldsymbol{c} \\ &= \begin{pmatrix} u^2 & u & 1 \end{pmatrix} \boldsymbol{M}_2 \\ &= U_2\ \boldsymbol{M}_2 \end{aligned} \qquad (11.3)$$

Lagrange's geometric form

The passage of the curve for three given points $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ and $\boldsymbol{p}_3$ can be imposed in order to specify the matrix $\boldsymbol{M}_2$. In particular, by substituting in equation (11.3) for $u \in \{0, \frac{1}{2}, 1\}$, we can write:

$$\begin{pmatrix} \boldsymbol{C}(0) \\ \boldsymbol{C}(\frac{1}{2}) \\ \boldsymbol{C}(1) \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ \frac{1}{4} & \frac{1}{2} & 1 \\ 1 & 1 & 1 \end{pmatrix} \boldsymbol{M}_2 = \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{pmatrix},$$

from which we get

$$\boldsymbol{M}_2 = \begin{pmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{pmatrix},$$

and hence

$$\boldsymbol{C}(u) = U_2\boldsymbol{M}_2 = \begin{pmatrix} u^2 & u & 1 \end{pmatrix} \begin{pmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{pmatrix} = U_2\boldsymbol{B}_2\boldsymbol{G}_2.$$

The previous interpolating form is called *Lagrange's geometric form* of the quadratic polynomial curve.

If we want to underline the meaning of this quadratic curve as a polynomial blending of three given points, then we may write:

$$\boldsymbol{C}(u) = (2u^2 - 3u + 1)\boldsymbol{p}_1 + (-4u^2 + 4u)\boldsymbol{p}_2 + (2u^2 - u)\boldsymbol{p}_3.$$

11.2.3 Cubic curves

Polynomial curves of third degree are largely used in CAD systems since they are sufficiently flexible for most applications. The *algebraic form* of their equation contains four vector parameters:

$$\begin{aligned} \boldsymbol{C}(u) &= \boldsymbol{a}u^3 + \boldsymbol{b}u^2 + \boldsymbol{c}u + \boldsymbol{d} \\ &= \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \boldsymbol{M}_3 \\ &= \boldsymbol{U}_3\ \boldsymbol{M}_3 \end{aligned} \qquad (11.4)$$

Every set of four vector constraints that allow specification of the four degrees of freedom defines a different *geometric form* of the cubic curve. Such constraints may impose the passage of the curve through four assigned points, or through two points with assigned tangents (derivatives) in those points, and so on, as we see in the following sections.

Lagrange's geometric form

A Lagrange cubic curve is defined by imposing the passage through four assigned points $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$ and $\boldsymbol{p}_4$, with four equispaced values of the parameter in the unit interval, i.e. with $u \in \{0, \frac{1}{3}, \frac{2}{3}, 1\}$, respectively:

$$\boldsymbol{C}(0) = \boldsymbol{p}_1, \qquad \boldsymbol{C}\left(\tfrac{1}{3}\right) = \boldsymbol{p}_2, \qquad \boldsymbol{C}\left(\tfrac{2}{3}\right) = \boldsymbol{p}_3, \qquad \boldsymbol{C}(1) = \boldsymbol{p}_4,$$

so that, by substitution in equation (11.4) we have

$$\begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \\ \boldsymbol{p}_4 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ \frac{1}{27} & \frac{1}{9} & \frac{1}{3} & 1 \\ \frac{8}{27} & \frac{4}{9} & \frac{2}{3} & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \boldsymbol{M}_3,$$

and hence

$$\boldsymbol{M}_3 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ \frac{1}{27} & \frac{1}{9} & \frac{1}{3} & 1 \\ \frac{8}{27} & \frac{4}{9} & \frac{2}{3} & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \\ \boldsymbol{p}_4 \end{pmatrix} = \boldsymbol{B}_L\ \boldsymbol{G}_L.$$

In conclusion, the Lagrange form of the cubic curve is defined as:

$$\boldsymbol{C}(u) = \boldsymbol{U}_3\ \boldsymbol{B}_L\ \boldsymbol{G}_L \tag{11.5}$$

where $\boldsymbol{U}_3$ is the cubic power basis, $\boldsymbol{B}_L$ is the matrix of the cubic Lagrange's basis and $\boldsymbol{G}_L$ is the Lagrange control tensor.

Hermite's geometric form

In this case the cubic curve segment is forced to have assigned extreme points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ and assigned extreme tangents $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, with $u \in [0,1]$. Such constraints are therefore:

$$\begin{aligned} \boldsymbol{C}(0) &= \boldsymbol{p}_1 \\ \boldsymbol{C}(1) &= \boldsymbol{p}_2 \\ \boldsymbol{C}'(0) &= \boldsymbol{s}_1 \\ \boldsymbol{C}'(1) &= \boldsymbol{s}_2 \end{aligned}$$

From the algebraic form (11.4) of the curve we can compute the derivative with respect to the parameter, which gives the tangents to the curve:

$$\boldsymbol{C}'(u) = \begin{pmatrix} 3u^2 & 2u & 1 & 0 \end{pmatrix} \boldsymbol{M}_3 \tag{11.6}$$

and, by substitution of $0, 1$ for u in equations (11.4) or (11.6), respectively:

$$\begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \boldsymbol{M}_3.$$

So, we get

$$\boldsymbol{M}_3 = \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \end{pmatrix}$$

so that the Hermite geometric form for the cubic polynomial curves results

$$\boldsymbol{C}(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \end{pmatrix}.$$

It can be written in matrix form as

$$\boldsymbol{C}(u) = U_3\ \boldsymbol{B}_h\ \boldsymbol{G}_h, \tag{11.7}$$

where $\boldsymbol{B}_h$ is the matrix of the Hermite Basis and $\boldsymbol{G}_h$ is the Hermite control tensor of cubic curves.

Cubic Hermite basis Let us rewrite the cubic Hermite curve as the polynomial combination of the elements of its control tensor. From equation (11.7) we get

$$\boldsymbol{C}(u) = \begin{pmatrix} h_1(u) & h_2(u) & h_3(u) & h_4(u) \end{pmatrix} \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \end{pmatrix}$$

where the Hermite basis polynomials are:

$$\begin{aligned} h_1(u) &= 2u^3 - 3u^2 + 1 \\ h_2(u) &= -2u^3 + 3u^2 \\ h_3(u) &= u^3 - 2u^2 + u \\ h_4(u) &= u^3 - u^2 \end{aligned}$$

In Figure 11.1 the graphs of such basis functions are shown. Let us notice that not all of them are nonnegative in the interval $[0, 1]$. This fact implies that some of the important properties which hold, e.g., for the Bézier form of a curve conversely do not hold for the Hermite form.

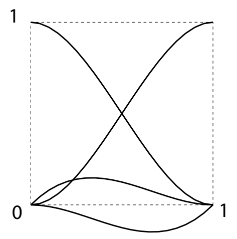

Figure 11.1 Graphs of the cubic polynomial Hermite basis

Example 11.2.1 (Graphs of the Hermite basis)
In Script 11.2.1 we produce the graph of the four polynomials of the Hermite basis in the $[0, 1]$ interval. A function `Intervals:a:n` produces a partition of the real interval $[0, a]$ into n subintervals as a $(1, 1)$-dimensional polyhedral complex. The function `BaseHermite` then generates a polyhedral complex of dimension $(1, 2)$ made by the graphs of the four basis polynomials, that is shown in Figure 11.1. Notice that the Hermite polynomials are translated quite literally into variable-free functions `h1`, `h2`, `h3` and `h4`, using algebraic operators that work in function spaces. Notice also that, for this purpose, the constant multipliers of power terms are translated into constant functions.

Example 11.2.2 (Examples of Hermite curves)
A plane cubic Hermite curve may be written in scalar form, i.e. by giving its coordinate maps as:

$$\begin{aligned} x(u) &= x_1 h_1(u) + x_2 h_2(u) + t_{1x} h_3(u) + t_{2x} h_4(u) \\ y(u) &= y_1 h_1(u) + y_2 h_2(u) + t_{1y} h_3(u) + t_{2y} h_4(u) \end{aligned}$$

Script 11.2.1

```
DEF Intervals (a::IsReal) (n::IsIntPos) = (QUOTE ∼ #:n):(a/n)

DEF BaseHermite =
   STRUCT ∼ [ MAP:[u,h1], MAP:[u,h2], MAP:[u,h3], MAP:[u,h4] ]
   WHERE
      h1 = (k:2)*(u*u*u) - (k:3)*(u*u) + (k:1),
      h2 = (k:3)*(u*u) - (k:2)*(u*u*u),
      h3 = (u*u*u) - (k:2)*(u*u) + u,
      h4 = (u*u*u) - (u*u),
      u = s1
   END;

BaseHermite:(Intervals:1:20)
```

Direct implementation of such equations in given in Script 11.2.2, where a function `Hermite2D` is defined, that, when applied to a quadruple of 2D control handles $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, $t_1 = (t_{1x}, t_{1y})$ and $t_2 = (t_{2x}, t_{2y})$, generates a vector function which can be mapped by the `MAP` operator over a 1D polyhedral partition of the unit segment.

Script 11.2.2

```
DEF Hermite2D (p1,p2,t1,t2::IsSeq) =
   [(x:p1 * h1) + (x:p2 * h2) + (x:t1 * h3) + (x:t2 * h4),
    (y:p1 * h1) + (y:p2 * h2) + (y:t1 * h3) + (y:t2 * h4)]
WHERE
   h1 = (k:2*u*u*u) - (k:3*u*u) + (k:1),
   h2 = (k:3*u*u) - (k:2*u*u*u),
   h3 = (u*u*u) - (k:2*u*u) + u,
   h4 = (u*u*u) - (u*u),
   u = s1, x = (k∼s1), y = (k∼s2)
END;

(STRUCT∼[
   MAP:(Hermite2D:<<1,0>,<1,1>,<-:1,1>,<1,0>>),
   MAP:(Hermite2D:<<1,0>,<1,1>,<-:2,2>,<2,0>>),
   MAP:(Hermite2D:<<1,0>,<1,1>,<-:4,4>,<4,0>>),
   MAP:(Hermite2D:<<1,0>,<1,1>,<-:10,10>,<10,0>>)
]): (Intervals:1:20)
```

Four cubics generated by the `PLaSM` final expression of Script 11.2.2 are shown in Figure 11.2. Notice that the proportional growth of the extreme tangent vectors may produce a self-loop in the curve. This possibility is not actually loved by designers. Thus, Hermite curves are used very carefully in CAD systems.

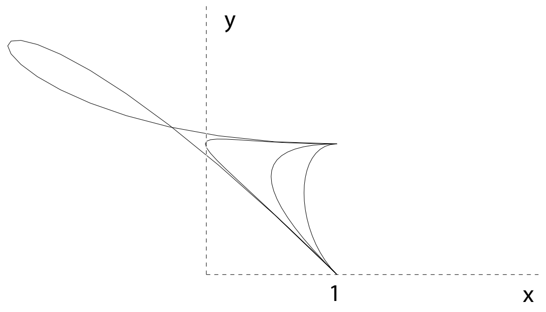

Figure 11.2 Four examples of Hermite curves through the same extreme points, and with proportional extreme tangents

Bézier's geometric form

When the cubic curve is built by blending the ordered sequence of four control points, it is said to be in Bézier's geometric form, or also called the cubic Bézier curve. Such a curve interpolates the two extreme control points, and approximates the remaining two points.

Let us suppose the cubic arc is generated by points $\boldsymbol{q}_0$, $\boldsymbol{q}_1$, $\boldsymbol{q}_2$ and $\boldsymbol{q}_3$, and defined for $u \in [0,1]$. To get its parametric equation we solve the set of constraints that impose (a) the passage through the extreme control points, and (b) the starting and ending tangent vectors parallel to the difference of two control points. Formally:

$$\begin{aligned}
\boldsymbol{C}(0) &= \boldsymbol{q}_0 \\
\boldsymbol{C}(1) &= \boldsymbol{q}_3 \\
\boldsymbol{C}'(0) &= \boldsymbol{s}_0 = 3(\boldsymbol{q}_1 - \boldsymbol{q}_0) \\
\boldsymbol{C}'(1) &= \boldsymbol{s}_1 = 3(\boldsymbol{q}_3 - \boldsymbol{q}_2),
\end{aligned}$$

so that, by using the equation of the Hermite's geometric form, we get:

$$\begin{aligned}
\boldsymbol{C}(u) &= U_3\, \boldsymbol{M}_3 \\
&= U_3\, \boldsymbol{B}_h\, \boldsymbol{G}_h \\
&= U_3\, \boldsymbol{B}_h \begin{pmatrix} \boldsymbol{q}_0 \\ \boldsymbol{q}_3 \\ 3(\boldsymbol{q}_1 - \boldsymbol{q}_0) \\ 3(\boldsymbol{q}_3 - \boldsymbol{q}_2) \end{pmatrix} \\
&= U_3\, \boldsymbol{B}_h \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_0 \\ \boldsymbol{q}_1 \\ \boldsymbol{q}_2 \\ \boldsymbol{q}_3 \end{pmatrix} \\
&= U_3\, \boldsymbol{B}_h\, \boldsymbol{B}_{hb}\, \boldsymbol{G}_b \\
&= U_3\, \boldsymbol{B}_b\, \boldsymbol{G}_b
\end{aligned}$$

where $\boldsymbol{B}_b$ is said matrix of the cubic Bézier basis, and $\boldsymbol{G}_b$ is the Bézier control tensor.

Transformations between geometric forms We have already computed the cubic Bézier form starting from the Hermite form:

$$U_3 \, \boldsymbol{B}_h \, \boldsymbol{G}_h = U_3 \, \boldsymbol{B}_h \, \boldsymbol{B}_{hb} \, \boldsymbol{G}_b = U_3 \, \boldsymbol{B}_b \, \boldsymbol{G}_b$$

where we set

$$\boldsymbol{B}_h \, \boldsymbol{B}_{hb} = \boldsymbol{B}_b$$

with $\boldsymbol{B}_{hb}$ linear transformation from Hermite to Bézier forms. Analogously, we have

$$\boldsymbol{B}_b \, (\boldsymbol{B}_{hb})^{-1} = \boldsymbol{B}_b \, \boldsymbol{B}_{bh} = \boldsymbol{B}_h$$

with $\boldsymbol{B}_{hb}$ linear transformation from Bézier to Hermite.

More in general, every geometric form of a curve can be transformed into another form by a similar approach, since the choice of a new geometric form is simply using a different basis of polygons to blend the control handles of a curve. The transformation matrices discussed above are exactly the matrices of two linear *transformations of coordinates* (from one basis to another one) into the vector space of polynomials of proper degree.

Cubic Bernstein/Bézier basis The cubic Bézier curve may be written as the blending of control points with the polynomials of the Bézier basis, also known as *Bernstein polynomials*:

$$\boldsymbol{C}(u) = \begin{pmatrix} b_0(u) & b_1(u) & b_2(u) & b_3(u) \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_0 \\ \boldsymbol{q}_1 \\ \boldsymbol{q}_2 \\ \boldsymbol{q}_3 \end{pmatrix}$$

where

$$\begin{pmatrix} b_0(u) & b_1(u) & b_2(u) & b_3(u) \end{pmatrix} = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

and hence the Bernstein/Bézier cubic polynomials are

$$\begin{aligned} b_0(u) &= (1-u)^3, \\ b_1(u) &= 3u(1-u)^2, \\ b_2(u) &= 3u^2(1-u), \\ b_3(u) &= u^3. \end{aligned}$$

If we denote with $B_i^3(u)$, $0 \le i \le 3$, the elements of the cubic Bernstein basis, we can write:

$$B_i^3(u) = \binom{3}{i} u^i (1-u)^{3-i}.$$

11.2.4 Higher-order Bézier curves

More in general, a Bézier curve of degree n is a function $\boldsymbol{C} : [0,1] \to \mathbb{E}^d$ defined as a polynomial combination of $n+1$ control points $\boldsymbol{q}_i \in \mathbb{E}^d$:

$$\boldsymbol{C}(u) = \sum_{i=0}^{n} B_i^n(u)\ \boldsymbol{q}_i, \qquad u \in [0,1] \tag{11.8}$$

where the blending functions $B_i^n : \mathbb{R} \to \mathbb{R}$ are the Bernstein polynomials:

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}$$

Bernstein basis of degree n

In order to generate a curve or a function graph we need to map a suitable vector function over some partition of the unit interval $[0,1]$. Such polyhedral decomposition of the unit interval is stored in the `Domain` object of Script 11.2.3, whereas the `Intervals` generating function was given in Script 11.2.1.

Then we need to compute the Bernstein basis of polynomials

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i} \qquad i = 0, \ldots, n$$

for any given n. This can be readily done in `PLaSM`. In particular, the `BernsteinBasis` function, when applied to an integer n, will return the ordered sequence of the $n+1$ Bernstein polynomials of degree n.

In order to compute a `BernsteinBasis` of arbitrary degree n, a small toolbox of utility functions is needed. In particular, the applications `Fact:n`, `Choose:<n,i>` and `Bernstein:n:i` will respectively compute the factorial number $n!$, the binomial number $\binom{n}{i}$ and the Bernstein polynomial B_i^n. The definition of the `Bernstein` function in Script 11.2.3 may seem a little tricky, but the reader should consider that it actually computes a curried function with signature

$$B_i^n : \mathbb{Z} \to \mathbb{Z} \to (\mathbb{R} \to \mathbb{R}) : n \mapsto i \mapsto \left(u \mapsto \binom{n}{i} u^i (1-u)^{n-i}\right)$$

where $\mathbb{Z}$ is the set of nonnegative integers.

Script 11.2.3 (Bernstein/Bézier basis)

```
DEF Domain = Intervals:1:30;

DEF Fact = IF: <C:EQ:0, K:1, * ~ INTSTO >;
DEF Choose = IF:< OR ~ [C:EQ:0 ~ S2, EQ], K:1, Choose ~ AA:(C:+:-1) * / >;
DEF Bernstein (n::IsInt)(i::IsInt) =
   *~[K:(Choose:<n,i>),**~[ID,K:i],**~[-~[K:1,ID],K:(n-i)]]~S1;

DEF BernsteinBasis (n::IsInt) = AA:(Bernstein:n):(0..n);
```

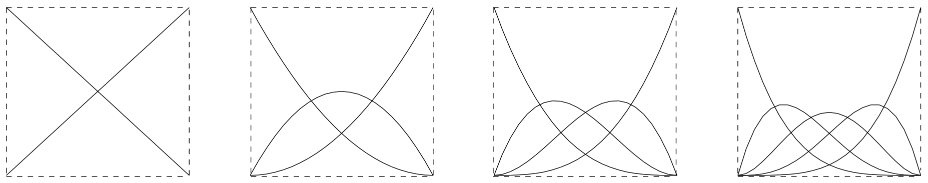

Figure 11.3 The graphs of Bernstein's bases of degrees 1, 2, 3 and 4

The functions `Fact` and `Choose`, for the factorial and the binomial numbers, were introduced in Scripts 2.1.6 and 2.1.10, respectively. They are recalled above for the reader's convenience. The graph of the `BernsteinBasis:4` basis shown in Figure 11.3d can be obtained by evaluating the expression:

```
STRUCT:<MAP:[S1, Bernstein:4:4]: Domain,
        MAP:[S1, Bernstein:4:3]: Domain,
        MAP:[S1, Bernstein:4:2]: Domain,
        MAP:[S1, Bernstein:4:1]: Domain,
        MAP:[S1, Bernstein:4:0]: Domain >;
```

More in general, a `PLaSM` function `BernsteinGraph`, that fully exploits the combinatorial power of the `FL` language, is given in Script 11.2.4. This function is used to generate the graph of the Bernstein's basis of arbitrary degree n.

Script 11.2.4 (SVG exporting of Bezier basis graphs)

```
DEF BernsteinGraph (n::IsIntPos) = STRUCT:(
   (CONS ∼ AA:(MAP ∼ CONS) ∼ DISTL):< S1, BernsteinBasis:n >: Domain
   AR CUBOID:<1,1>
);

DEF out = (STRUCT ∼ CAT ∼ AA:[BernsteinGraph, K:(T:1:1.33)]):(1..4);
svg:out:10:'out.svg';
```

The graphs of the Bernstein/Bézier bases of degrees 1,2,3 and 4 shown in Figure 11.3 are produced by evaluating the `out` symbol of Script 11.2.4. Its geometric value is first exported to a `.svg` file by the last expression of the script, then imported in *Adobe Illustrator* for handling the line size, and finally exported into a `.eps` file to be included by the LaTeX system used to typeset the present book. Most of pictures in this chapter are produced by a similar procedure.

Bézier mapping of degree n

We are finally ready to compute the Bézier mapping of degree n from the parameter domain $[0,1]$ to the target space $\mathbb{E}^d$.

According to equation (11.8), we have to combine the sequence of `ControlPoints` with the `BernsteinBasis` of proper `degree`. For this purpose, the `ControlPoints` are transposed into the sequence `coordinateSeqs` of coordinate sequences, where each number is transformed into a constant function. Finally, each element in `coordinateSeqs`, i.e. each sequence of corresponding coordinates, is linearly combined with the basis polynomials returned by the expression `BernsteinBasis:degree`, resulting in a sequence of *coordinate maps*.

Let us remember that the `LEN` function returns the length of the sequence it is applied to, and that the `InnerProd` function for inner product of vectors (and function vectors) was defined in Script 3.2.4.

Script 11.2.5

```
DEF Bezier (ControlPoints::IsSeq) = (AA:InnerProd ∼ DISTL):
   < BernsteinBasis:degree, coordinateSeqs >
WHERE
   degree = LEN:ControlPoints - 1,
   coordinateSeqs = ((AA ∼ AA):K ∼ TRANS):ControlPoints
END;
```

The `Bezier` function given above may be applied to an arbitrary sequence of control points, thus generating a Bezier's mapping of the appropriate degree, to be applied by the `MAP` function to some partition of the unit segment.

Notice in particular that the `Bezier` function may work for an *arbitrary* number of control points, including one. Furthermore, it can be used either in 2D or in 3D as well as for *every* other dimension of the target space, i.e. for arbitrary number of coordinates of curve points.

Example 11.2.3 (Bézier curves of various degrees)
The curve examples of Figure 11.4 are produced by the `out` object defined in Script 11.2.6. Both the curves and the corresponding control polygons are displayed, for curve degrees increasing from 1 to 4. The four examples are generated by the control point sequences `pol1`, `pol2`, `pol3` and `pol4`. The generated polyhedral complex `out`, including the four curve images and the four control polygons, is finally exported as `.svg` file. The `polyline` primitive was defined in Script 7.2.3.

Script 11.2.6 (Examples of Bézier curves)

```
DEF pol1 = <<0,0>,<10,10>>;
DEF pol2 = <<0,0>,<10,0>,<10,10>>;
DEF pol3 = <<0,0>,<10,0>,<0,10>,<10,10>>;
DEF pol4 = <<0,0>,<10,0>,<10,10>,<0,10>,<0,0>>;

DEF out = STRUCT:<
   MAP:(Bezier:pol1):domain, polyline:pol1, T:1:15,
   MAP:(Bezier:pol2):domain, polyline:pol2, T:1:15,
   MAP:(Bezier:pol3):domain, polyline:pol3, T:1:15,
   MAP:(Bezier:pol4):domain, polyline:pol4
>;

svg:out:10:'out.svg'
```

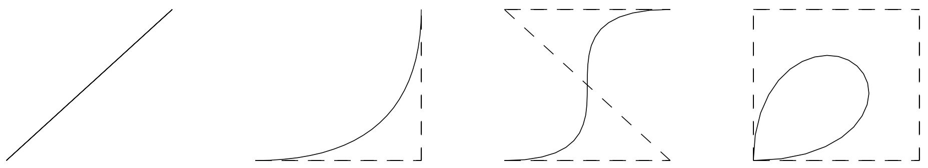

Figure 11.4 Bézier curves of degrees 1, 2, 3 and 4 with their control polygons

Properties of Bézier curves

Bézier curves are greatly used by CAD systems because of their very useful properties, that we briefly summarize below:

1. *Affine invariance*
   A Bézier curve can be translated, rotated or scaled by simply translating, rotating or scaling its control points.
2. *Variation diminishing*
   The number of intersections of a Bézier curve with a hyperplane does not exceed the number of intersections of its control polygon.
3. *Interpolation of extreme points*
   A Bézier curve interpolates the first and last control points. Notice, in fact, that $B_0^n(0) = B_n^n(1) = 1$, whereas $B_i^n(0) = B_i^n(1) = 0$, for $0 < i < n$.
4. *Convex hull containment*
   A Bézier curve of arbitrary degree n is contained in the convex hull of its control points $\boldsymbol{q}_0, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$. The proof is simple:

$$\boldsymbol{C}(u) = \sum_{i=0}^{n} B_i^n(u)\ \boldsymbol{q}_i$$

   is a convex combination of $\boldsymbol{q}_0, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$, for every $u \in [0, 1]$.
   The above statement is true because all polynomials $B_i^n(u)$ are non negative and sum to 1 for every $u \in [0, 1]$. Using Newton's formula for the power of binomials, we get in fact:

$$\sum_{i=0}^{n} B_i^n(u) = \sum_{i=0}^{n} \binom{n}{i} u^i (1-u)^{n-i} = (u + (1-u))^n = 1.$$

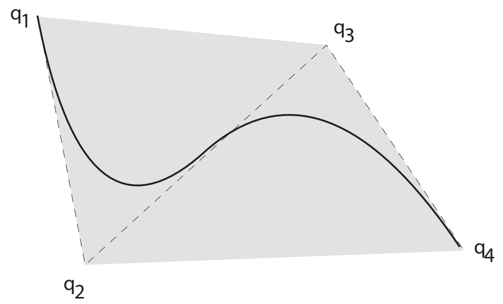

Figure 11.5 Containment of a Bézier curve in the convex hull of control points

11.3 Polynomial splines

A *spline* is a composite curve, defined by joining some adjacent curve segments with an appropriate order of continuity at the joints. We discuss in this section some useful classes of polynomial splines, largely diffused in most graphics and CAD systems, and in particular:

1. *cubic cardinal splines*, made by C^1-continuous Hermite cubic segments, which interpolate all the points in their control sequence;
2. *cubic uniform B-splines*, which only approximate the control point sequence, using cubic curve segments joined with C^2 continuity;
3. *non-uniform B-splines* of arbitrary degree, which are polynomial splines of great flexibility, that enjoy several useful properties.

11.3.1 Cubic cardinal splines

With *cubic cardinal splines*, also called *Catmull-Rom* splines [FvDFH90], each component curve segment is a Hermite cubic. In particular, a cubic cardinal spline is defined by a sequence $\boldsymbol{p}_0, \boldsymbol{p}_1, \dots, \boldsymbol{p}_m$ of $m+1$ control points ($m \geq 3$), that define $m-2$ adjacent cubic segments. The i-th curve segment $\boldsymbol{C}_i(u)$ is defined by the interpolation of points $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$, and by imposing that the tangent vector in a point $\boldsymbol{p}_i$ is parallel to the vector difference between the adjacent points $\boldsymbol{p}_{i+1}$ and $\boldsymbol{p}_{i-1}$. Formally we have:

$$\begin{aligned}
\boldsymbol{C}_i(0) &= \boldsymbol{p}_i \\
\boldsymbol{C}_i(1) &= \boldsymbol{p}_{i+1} \\
\boldsymbol{C}'_i(0) &= h(\boldsymbol{p}_{i+1} - \boldsymbol{p}_{i-1}) \\
\boldsymbol{C}'_i(1) &= h(\boldsymbol{p}_{i+2} - \boldsymbol{p}_i)
\end{aligned}$$

As a consequence of such constraints, a cubic cardinal will interpolate all the points in the subsequence $\boldsymbol{p}_1, \dots, \boldsymbol{p}_{m-1}$. In the intervals between $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$ and between $\boldsymbol{p}_{m-1}$ and $\boldsymbol{p}_m$ the spline is not defined.

The equation of the segment $\boldsymbol{C}_i(u)$ can be derived from the matrix form of Hermite curves as follows:

$$\boldsymbol{C}_i(u) = U_3\ \boldsymbol{M}_h \begin{pmatrix} \boldsymbol{p}_i \\ \boldsymbol{p}_{i+1} \\ h(\boldsymbol{p}_{i+1} - \boldsymbol{p}_{i-1}) \\ h(\boldsymbol{p}_{i+2} - \boldsymbol{p}_i \end{pmatrix}, \qquad u \in [0, 1].$$

Thus, it is:

$$\begin{aligned}
\boldsymbol{C}_i(u) &= \boldsymbol{U}_3\ \boldsymbol{M}_h \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -h & 0 & h & 0 \\ 0 & -h & 0 & h \end{pmatrix} \begin{pmatrix} \boldsymbol{p}_{i-1} \\ \boldsymbol{p}_i \\ \boldsymbol{p}_{i+1} \\ \boldsymbol{p}_{i+2} \end{pmatrix} \\
&= \boldsymbol{U}_3\ \boldsymbol{M}_h\ \boldsymbol{M}_{hc}\ \boldsymbol{G}_i\ =\ \boldsymbol{U}_3\ \boldsymbol{M}_c\ \boldsymbol{G}_i
\end{aligned}$$

where $\boldsymbol{M}_{hc}$ is the matrix of the transformation from the Hermite to the cardinal form, $\boldsymbol{M}_c$ is the matrix of the cardinal basis, and $\boldsymbol{G}_i$ is the control tensor of the i-th spline segment.

Cardinal basis

As we know, the set of polynomials of the cardinal basis can be obtained from the product of the power basis $\boldsymbol{U}_3$ times the matrix $\boldsymbol{M}_c$ of the geometric form. Hence we have:

$$\boldsymbol{C}_i(u) = \boldsymbol{U}_3\ \boldsymbol{M}_c\ \boldsymbol{G}_i = \begin{pmatrix} c_1(u) & c_2(u) & c_3(u) & c_4(u) \end{pmatrix} \begin{pmatrix} \boldsymbol{p}_{i-1} \\ \boldsymbol{p}_i \\ \boldsymbol{p}_{i+1} \\ \boldsymbol{p}_{i+2} \end{pmatrix},$$

where

$$\begin{pmatrix} c_1(u) & c_2(u) & c_3(u) & c_4(u) \end{pmatrix} = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} -h & 2-h & h-2 & h \\ 2h & h-3 & 3-2h & -h \\ -h & 0 & h & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

The expression of the cardinal cubic polynomial, whose graph is given in Figure 11.6, is:

$$\begin{aligned} c_1(u) &= -hu^3 + 2hu^2 - hu \\ c_2(u) &= (2-h)u^3 + (h-3)u^2 + 1 \\ c_3(u) &= (h-2)u^3 + (3-2h)u^2 + hu \\ c_4(u) &= hu^3 - hu^2 \end{aligned}$$

It may be interesting to look at the shape of the four basis polynomials, shown in Figure 11.6, where the graph of each is separately given over the interval $[0, 1]$. Notice that two such polynomials are not positive in the unit interval.

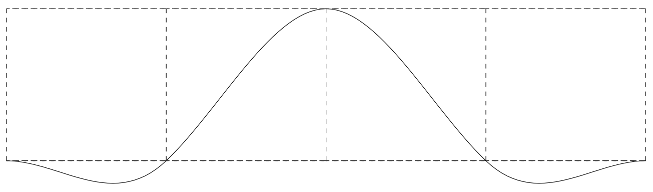

Figure 11.6 From left to right: graphs of polynomials $c_4(u)$, $c_3(u)$, $c_2(u)$ and $c_1(u)$ of the cubic cardinal basis

Implementation The graph of Figure 11.6 is generated by evaluating the expression

```
CubicCardinalGraph:(Intervals:1:20)
```

where the function `CubicCardinalGraph` is given in Script 11.3.1, and defines a 2D structure containing the graphs of the single basis polynomials separated by a unit translation along the first coordinate axes.

Several examples of cubic cardinal splines will be discussed in Section 11.6.1, where a unified implementation of cardinal and uniform B-splines will be given. In that section we also discuss how to produce two more curve segments passing through the extreme points of the control segment.

Script 11.3.1

```
DEF CubicCardinalGraph =
   STRUCT ~ [MAP:[u,c4], k:(T:1:1), MAP:[u,c3], k:(T:1:1),
      MAP:[u,c2], k:(T:1:1), MAP:[u,c1]]
WHERE
   c1 = ((k:0 - h) * u3) + ((k:2 * h) * u2) - (h * u),
   c2 = ((k:2 - h) * u3) + ((h - k:3) * u2) + (k:1),
   c3 = ((h - k:2) * u3) + ((k:3 - k:2*h) *u2) + (h*u),
   c4 = (h * u3) - (h * u2),
   u = s1, u2 = u*u, u3 = u2*u, h = k:1
END;
```

11.3.2 Cubic uniform B-splines

Cubic uniform B-splines are composite curves used to approximate a sequence of $m+1$ control points $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_m$, with $m \geq 3$. Such control points define $m-2$ adjacent segments of cubic polynomial curve. In particular, the i-th curve segment, denoted as $\boldsymbol{Q}_i(u)$, is defined by imposing the geometric continuity and the continuity of first and second derivatives at the extreme points of the segment.

The discussion of cubic B-splines given in this section concerns the so-called *uniform* cubic B-splines, where each curve segment is obtained by a combination of four control points with the *uniform B-spline basis*. Each polynomial in this basis has the interval $[0, 1]$ as its domain. In a later section we discuss non-uniform B-splines, which are parametrized over different intervals of the real line.

Uniform B-spline geometric form

The parametric equation of cubic segment $\boldsymbol{Q}_i(u)$, with $u \in [0, 1]$, can be written in geometric form as follows:

$$\begin{aligned}\boldsymbol{Q}_i(u) &= \boldsymbol{U}_3\,\frac{1}{6}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix}\begin{pmatrix} \boldsymbol{p}_{i-1} \\ \boldsymbol{p}_i \\ \boldsymbol{p}_{i+1} \\ \boldsymbol{p}_{i+2} \end{pmatrix}, \qquad 1 \leq i \leq m-2, \\ &= \boldsymbol{U}_3\ \boldsymbol{B}_s\ \boldsymbol{G}_i\end{aligned}$$

where $\boldsymbol{B}_s$, called the matrix of the cubic uniform B-spline, is derived in a later subsection. Therefore, the uniform cubic B-basis polynomials, given by the product $\boldsymbol{U}_3\boldsymbol{B}_s$ and graphically shown in Figure 11.7, are

$$\begin{aligned} B_1(u) &= \frac{1}{6}(-u^3 + 3u^2 - 3u + 1) \\ B_2(u) &= \frac{1}{6}(3u^3 + -6u^2 + 4) \\ B_3(u) &= \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1) \\ B_4(u) &= \frac{1}{6}u^3 \end{aligned}$$

Uniform B-spline basis

The graph of Figure 11.7 is generated by evaluating the last expression of Script 11.3.2, where the function `UniformBsplineGraph` defines a 2D structure with the graphs of the single basis polynomials inserted within an empty unit box and separated by a unit translation on the first coordinate axis. Notice that all B-spline basis functions, unlike the cardinal case, are non-negative. They also sum to one for each u. According to such properties, each spline segment $\boldsymbol{Q}_i(u)$ is contained in the convex hull of its geometry vector $\boldsymbol{G}_i$.

Script 11.3.2

```
DEF UniformBsplineGraph =
   STRUCT ∼ [Box, MAP:[u,B4], k:(T:1:1), Box, MAP:[u,B3], k:(T:1:1),
      Box, MAP:[u,B2], k:(T:1:1), Box, MAP:[u,B1]]
WHERE
   B1 = a * ((k:3*u2) - (k:1*u3) - (k:3*u) + (k:1)),
   B2 = a * ((k:3*u3) - (k:6*u2) + (k:4)),
   B3 = a * ((k:3*u2) - (k:3*u3) + (k:3*u) + (k:1)),
   B4 = a * u3,
   u = s1, u2 = u*u, u3 = u2*u, a = k:(1/6)
   Box = (K ∼ @1 ∼ CUBOID):<1,1>
END;

DEF out = UniformBsplineGraph:(Intervals:1:30)

SVG:out:10:'out.svg'
```

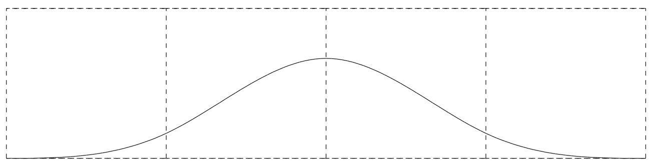

Figure 11.7 From left to right: graphs of polynomials $B_4(u)$, $B_3(u)$, $B_2(u)$ and $B_1(u)$ of the cubic uniform B-spline basis

Global parametrization

When dealing with splines, it may be useful to maintain a bijective correspondence between spline points and parameter values. For this purpose a spline form is often used where the various curve segments are parametrized over subsequent real intervals of unit size.

In order to describe the whole spline and not the single spline segments, we change our notation, with respect both to the parameter and to the indexing of segments. This notation is adopted for the sake of uniformity with the one used in the following section for *non-uniform* B-splines.

As we know, a cubic B-spline approximates a sequence of $m+1$ control points $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_m$, $m \geq 3$, with a sequence of $m-2$ segments of cubic curves, which can be indexed as $\boldsymbol{Q}_3, \boldsymbol{Q}_4, \ldots, \boldsymbol{Q}_m$. With this choice of indices, a $\boldsymbol{Q}_i$ segment is a polynomial combination of points $\boldsymbol{p}_{i-4+k}$, $1 \leq k \leq 4$. Thus, the first segment $\boldsymbol{Q}_3$ is generated by points $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_3$, and the last segment $\boldsymbol{Q}_m$ is generated by points $\boldsymbol{p}_{m-3}, \cdots, \boldsymbol{p}_m$.

The uniform cubic B-spline can be written as a function of a global parameter $3 \leq t \leq m+1$, as

$$\boldsymbol{Q}(t) = \bigcup_{i=3}^{m} \boldsymbol{Q}_i(t-i).$$

Each integer parameter value $t \in \{3, 4, \ldots, m+1\}$ is called a *knot*. The image $\boldsymbol{Q}(t)$ of a knot, where $\boldsymbol{Q}_i(1) = \boldsymbol{Q}_{i+1}(0)$, is called a join point, or *joint*.

Uniform B-spline matrix

We derive in this section the matrix of the cubic uniform B-spline, already given in Section 11.3.2. For this purpose we start from the boundary conditions imposed on pairs of cubic segments. In particular, a pair of adjacent spline segments $\boldsymbol{Q}_i(u)$ and $\boldsymbol{Q}_{i+1}(u)$, $0 \leq u \leq 1$, must be C^2-continuous, i.e. must satisfy the following conditions at the joint:

$$\begin{aligned}
\boldsymbol{Q}_i(1) &= \boldsymbol{Q}_{i+1}(0). \\
\boldsymbol{Q}'_i(1) &= \boldsymbol{Q}'_{i+1}(0), \\
\boldsymbol{Q}"_i(1) &= \boldsymbol{Q}"_{i+1}(0).
\end{aligned}$$

To be more explicit, we can write:

$$\begin{aligned}
\sum_{k=1}^{4} B_k(1)\boldsymbol{p}_{i-4+k} &= \sum_{k=1}^{4} B_k(0)\boldsymbol{p}_{i-3+k} \\
\sum_{k=1}^{4} B'_k(1)\boldsymbol{p}_{i-4+k} &= \sum_{k=1}^{4} B'_k(0)\boldsymbol{p}_{i-3+k} \\
\sum_{k=1}^{4} B"_k(1)\boldsymbol{p}_{i-4+k} &= \sum_{k=1}^{4} B"_k(0)\boldsymbol{p}_{i-3+k}
\end{aligned}$$

Such continuity equations can be written in matrix form as follows:

$$\boldsymbol{B} \begin{pmatrix} \boldsymbol{p}_{i-3} \\ \boldsymbol{p}_{i-2} \\ \boldsymbol{p}_{i-1} \\ \boldsymbol{p}_i \\ \boldsymbol{p}_{i+1} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} \tag{11.9}$$

with

$$\boldsymbol{B} = \begin{pmatrix} B_1(1) & B_2(1) - B_1(0) & B_3(1) - B_2(0) & B_4(1) - B_3(0) & B_4(0) \\ B'_1(1) & B'_2(1) - B'_1(0) & B'_3(1) - B'_2(0) & B'_4(1) - B'_3(0) & B'_4(0) \\ B"_1(1) & B"_2(1) - B"_1(0) & B"_3(1) - B"_2(0) & B"_4(1) - B"_3(0) & B"_4(0) \end{pmatrix}$$

The matrix equation (11.9) states that 3 different linear combinations of 5 displacement vectors in general position must go to zero. This is only possible when all the coefficients, i.e. all the elements in the left-hand matrix, are zero.

Also, remembering that a blending function is the product of the power basis vector times a column of the unknown basis matrix $\boldsymbol{B} = (b_{hk})$, we can write, for $1 \leq k \leq 4$:

$$\begin{aligned}
B_k(u) &= \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} \begin{pmatrix} b_{1k} & b_{2k} & b_{3k} & b_{4k} \end{pmatrix}^T \\
B'_k(u) &= \begin{pmatrix} 3u^2 & 2u & 1 & 0 \end{pmatrix} \begin{pmatrix} b_{1k} & b_{2k} & b_{3k} & b_{4k} \end{pmatrix}^T \\
B"_k(u) &= \begin{pmatrix} 6u & 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} b_{1k} & b_{2k} & b_{3k} & b_{4k} \end{pmatrix}^T
\end{aligned}$$

so that we get, respectively:

$$\begin{aligned}
B_k(0) &= b_{4k} \quad \text{and} \quad & B_k(1) &= b_{1k} + b_{2k} + b_{3k} + b_{4k}, \\
B'_k(0) &= b_{3k} \quad \text{and} \quad & B'_k(1) &= 3b_{1k} + 2b_{2k} + b_{3k}, \\
B"_k(0) &= 2b_{2k} \quad \text{and} \quad & B"_k(1) &= 6b_{1k} + 2b_{2k}.
\end{aligned}$$

By setting to zero each term of the matrix in equation 11.9, we get the following 15 equations among the 16 unknowns b_{hk}, with $1 \leq h, k \leq 4$:

$$\begin{aligned}
B_1(1) &= b_{11} + b_{21} + b_{31} + b_{41} \;=\; 0 \\
B_2(1) - B_1(0) &= b_{12} + b_{22} + b_{32} + b_{42} - b_{41} \;=\; 0 \\
B_3(1) - B_2(0) &= b_{13} + b_{23} + b_{33} + b_{43} - b_{42} \;=\; 0 \\
B_4(1) - B_3(0) &= b_{14} + b_{24} + b_{34} + b_{44} - b_{43} \;=\; 0 \\
B_4(0) &= b_{44} \;=\; 0 \\
B'_1(1) &= 3b_{11} + 2b_{21} + b_{31} \;=\; 0 \\
B'_2(1) - B'_1(0) &= 3b_{12} + 2b_{22} + b_{32} - b_{31} \;=\; 0 \\
B'_3(1) - B'_2(0) &= 3b_{13} + 2b_{23} + b_{33} - b_{32} \;=\; 0 \\
B'_4(1) - B'_3(0) &= 3b_{14} + 2b_{24} + b_{34} - b_{33} \;=\; 0 \\
B'_4(0) &= b_{34} \;=\; 0 \\
B"_1(1) &= 6b_{11} + 2b_{21} \;=\; 0 \\
B"_2(1) - B"_1(0) &= 6b_{12} + 2b_{22} - 2b_{21} \;=\; 0 \\
B"_3(1) - B"_2(0) &= 6b_{13} + 2b_{23} - 2b_{22} \;=\; 0 \\
B"_4(1) - B"_3(0) &= 6b_{14} + 2b_{24} - 2b_{23} \;=\; 0 \\
B"_4(0) &= 2b_{24}
\end{aligned}$$

A 16th equation can be obtained by ensuring that the basis polynomials sum to 1 for each u. So, we set

$$B_1(u) + B_2(u) + B_3(u) + B_4(u) = 1,$$

and, in particular, for $u = 0$ we get:

$$B_1(0) + B_2(0) + B_3(0) + B_4(0) = b_{41} + b_{42} + b_{43} + b_{44} = 1.$$

By solving the above system of equations for the unknown terms of the matrix of cubic uniform B-basis, we get

$$\boldsymbol{B}_s = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix} = \frac{1}{6}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix}$$

11.3.3 Non-uniform polynomial B-splines

The B-splines discussed in this section are called *non-uniform* because different spline segments may correspond to different intervals in parameter space, unlike uniform B-splines. The basis polynomials, and consequently the spline shape and the other properties, are defined by a non-decreasing sequence of real numbers

$$t_0 \leq t_1 \leq \cdots \leq t_n,$$

called the *knot sequence*. Splines of this kind are also named *NUB-splines* in the remainder of this book,[4] where the name stands for Non-Uniform B-splines.

The knot sequence is used to define the basis polynomials which blend the control points. In particular, each subset of $k+2$ adjacent knot values is used to compute a basis polynomial of degree k. Notice that some subsequent knots may coincide. In this case we speak of *multiplicity* of the knots.

Note In non-uniform B-splines the number $n+1$ of *knot values* is greater than the number $m+1$ of control points $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_m$. In particular, the relation

$$n = m + k + 1, \tag{11.10}$$

where k is the *degree* of spline segments, must hold between the number of knots and the number of control points. The quantity $h = k+1$ is called the *order* of the spline. It will be useful when giving recursive formulas to compute the B-basis polynomials. Let us remember, e.g., that a spline of order four is made of cubic segments.

Non-uniform B-spline flexibility Such splines have a much greater flexibility than the uniform ones. The basis polynomial associated with each control point may vary depending on the subset of knots it depends on. Spline segments may be parametrized over intervals of different size, and even reduced to a single point. Therefore, the continuity at a joint may be reduced, e.g. from C^2 to C^1 to C^0 and even to none (see Figure 11.10) by suitably increasing the multiplicity of a knot.

Cubic non-uniform B-splines

According to the previous statement, the knot sequence of cubic non-uniform B-splines can be given as

$$(t_0, \ldots, t_3, \ldots, t_m, \ldots, t_{m+4}), \qquad t_i \leq t_{i+1}.$$

[4] Some authors call them non-uniform non-rational B-splines. We prefer to emphasize that they are polynomial splines.

In particular, the $\boldsymbol{Q}_i(t)$ cubic segment is defined as

$$\boldsymbol{Q}_i(t) = \sum_{\ell=0}^{3} \boldsymbol{p}_{i-\ell} B_{i-\ell,4}(t) \qquad \begin{array}{l} 3 \le i \le m, \\ t \in [t_i, t_{i+1}) \end{array} \tag{11.11}$$

where $B_{i,4}(u)$ is the B-basis polynomial of index i and order 4, which is generated by the knot subsequence $(t_i, \ldots, t_{i+4})$.

For the spline as a whole, we can write:

$$\boldsymbol{Q}(t) = \bigcup_{i=3}^{m} \boldsymbol{Q}_i(t), \qquad t \in [t_3, t_{m+1}).$$

Outside the interval $[t_3, t_{m+1})$ a cubic non-uniform B-spline is not defined. Within this interval, each pair of adjacent knot values is associated with a spline segment. When a t_i knot has multiplicity greater than one, i.e. when $t_i = t_{i+1}$, then the $\boldsymbol{Q}_i(t)$ segment reduces to a point.

Example 11.3.1 (Cubic Bézier)
Let us note that a cubic B-spline with a single curve segment needs 4 control points. As a consequence, the number of knots in this case must be 8, i.e. $4 + 3 + 1$. When parametrizing the spline in the interval $[0, 1]$, the knot sequence will therefore be:

$$(0, 0, 0, 0, 1, 1, 1, 1) \tag{11.12}$$

The four basis polynomials to be combined with control points and the associated knot subsequences are: $B_{0,4}$ and $(0, 0, 0, 0, 1)$, $B_{1,4}$ and $(0, 0, 0, 1, 1)$, $B_{2,4}$ and $(0, 0, 1, 1, 1)$ and finally $B_{3,4}$ and $(0, 1, 1, 1, 1)$.

It is possible to show that the cubic non-uniform B-spline corresponding to the knot sequence (11.12) is the cubic Bézier curve.

Geometric entities

In order to fully understand the construction of a non-uniform B-spline, it may be useful to recall the main inter-relationships among the 5 geometric entities that enter the definition.

Control points are denoted as $\boldsymbol{p}_i$, with $0 \le i \le m$. A non-uniform B-spline usually approximates the control points.

Knot values are denoted as t_i, with $0 \le i \le n$. It must be $n = m + k + 1$, where k is the spline degree. Knot values are used to define the B-spline polynomials. They also define the join points (or joints) between adjacent spline segments. When two consecutive knots coincide, the spline segment associated with their interval reduces to a point.

Spline degree is defined as the degree of the B-basis functions which are combined with the control points. The degree is denoted as k. It is connected to the spline order $h = k + 1$. The most used non-uniform B-splines are either cubic or quadratic. The image of a linear non-uniform B-spline is a polygonal line. The image of a non-uniform B-spline of degree 0 coincides with the sequence of control points.

B-basis polynomials are denoted as $B_{i,h}(t)$. They are univariate polynomials in the t indeterminate, computed by using the recursive formulas of Cox and de Boor (see Section 39). The i index is associated with the first one of values in the knot subsequence $(t_i, t_{i+1}, \ldots, t_{i+h})$ used to compute $B_{i,h}(t)$. The second index is called *order* of the polynomial.

Spline segments are defined as polynomial vector functions of a single parameter. Such functions are denoted as $\boldsymbol{Q}_i(t)$, with $k \leq i \leq m$. A $\boldsymbol{Q}_i(t)$ spline segment is obtained by a combination of the i-th control point and the k previous points with the basis polynomials of order h associated to the same indices. It is easy to see that the number of spline segments is $m - K + 1$.

Non-uniform B-splines of arbitrary degree

In this case the knot sequence may be written as

$$(t_0, \ldots, t_k, \ldots, t_{m+1}, \ldots, t_{m+h}), \qquad t_i \leq t_{i+1}.$$

with the spline defined in the interval $[t_k, t_{m+1})$. The first basis function $B_{0,h}(t)$ is defined by the knot subsequence $(t_0, \ldots, t_k, t_{k+1})$; the last function $B_{m,h}(t)$ is defined by $(t_m, t_{m+1}, \ldots, t_{m+h})$. Notice the bijection between basis polynomials and control points.

The equation of the non-uniform B-spline segment of degree k with $m + 1$ control points may therefore be written as

$$\boldsymbol{Q}_i(t) = \sum_{\ell=0}^{k} \boldsymbol{p}_{i-\ell} B_{i-\ell,h}(t) \qquad \begin{array}{l} k \leq i \leq m, \\ t \in [t_i, t_{i+1}) \end{array} \tag{11.13}$$

A global representation of the non-uniform B-spline as a whole can be given:

$$\boldsymbol{Q}(t) = \bigcup_{i=k}^{m} \boldsymbol{Q}_i(t) = \sum_{i=0}^{m} \boldsymbol{p}_i\, B_{i,h}(t), \qquad t \in [t_k, t_{m+1}),$$

since each basis function $B_{i,h}(t)$ is zero outside the subinterval $[t_i, t_{i+h})$.

If the multiplicity of the first (last) knot value is equal to h, then the spline interpolates the first (last) control point. The interpolation is induced by the fact that such multiplicity induces $B_{0,h}(t_k) = 1$ ($B_{m,h}(t_{m+1}) = 1$, respectively).

Example 11.3.2 (Degree elevation)
In Figure 11.8 we show both a quadratic and a cubic spline interpolating the extreme control points. If we want to raise by one unity the degree of a non-uniform B-spline without changing the control polygon, then we have to:

1. decrease by one the parameter interval;[5]
2. increase by one the multiplicity of both extreme knots.

As a consequence of the degree elevation, the number $m - h$ of segments decreases of one. The `PLaSM` code that generates Figure 11.8 is given in Script 11.3.3. The `DisplayAll` operator, which displays both a spline and its control polygon and a polymarker to show the joints, is given in Script 11.6.4. The `NUBspline` function is given in Script 11.6.14.

Script 11.3.3 (Degree elevation)
```
DisplayAll:NUBspline:
   < <2,<0,0,0,1,2,3,4,5,5,5>>,
      <<1,3>,<-1,2>,<1,4>,<2,3>,<1,1>,<1,2>,<2.5,1>> >

DisplayAll:NUBspline:
   < <3,<0,0,0,0,1,2,3,4,4,4,4>>,
      <<1,3>,<-1,2>,<1,4>,<2,3>,<1,1>,<1,2>,<2.5,1>> >
```

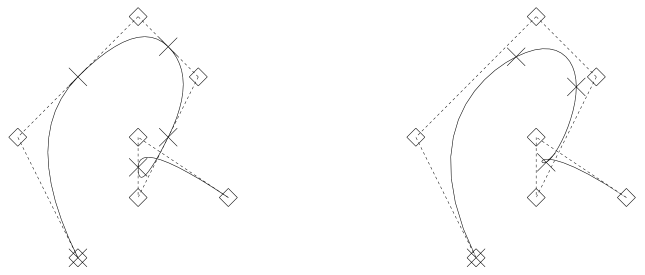

Figure 11.8 (a) Quadratic non-uniform B-spline with 7 control points and $5 = 7 - 2$ spline segments (b) Cubic non-uniform B-spline with same control points and $4 = 7 - 3$ segments

The reader may see, looking at Figure 11.8a, two interesting properties of quadratic non-uniform B-splines: (a) knots with multiplicity 1 are mapped to the middle point of a control polygon segment; (b) the spline is tangent there at the control polygon.

Coox and de Boor formula

To compute the non-uniform B-spline basis polynomials $B_{i,h}(t)$ the recursive formulas derived by Coox [Cox71] and de Boor [dB72] are used. Their formulation was applied by Riesenfeld [Rie73, GR74] to curve definition in CAD. An interesting approach to B-splines can be found in Rogers and Adams [RA76].

The computation of each basis function requires using a subset of knot values. In particular, the basis polynomial $B_{i,h}(t)$, called "with initial value t_i and *order h*", is

[5] In case of knot intervals of integral size.

defined as:

$$\begin{aligned} B_{i,1}(t) &= \begin{cases} 1 & \text{if } t_i \le t \le t_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ B_{i,h}(t) &= \frac{t - t_i}{t_{i+h-1} - t_i} B_{i,h-1}(t) + \frac{t_{i+h} - t}{t_{i+h} - t_{i+1}} B_{i+1,h-1}(t) \end{aligned} \tag{11.14}$$

The above definition is recursive. A basis function of order h and initial value t_i is defined using two basis functions of order $h-1$ and initial values t_i and t_{i+1}. In basic cases $B_{i,1}(t)$ of the recursion, concerning functions of order 1, *step functions* are used, whose value is 1 in the interval $[t_i, t_{i+1}]$, and 0 elsewhere.

Dependence of basis functions on knots Let us develop a simple analysis of the dependence of basis functions on knot values. It is easy to see, from the Cox and de Boor formula, that $B_{i,h}(t)$ depends on the subsequence of knot values

$$(t_i, t_{i+1}, \cdots, t_{i+h}) \tag{11.15}$$

For example, $B_{1,3}(t)$ depends on the 4 knots $(t_1, \cdots, t_4)$, because it depends on $B_{1,2}(t)$ and on $B_{2,2}(t)$. In turn, $B_{1,2}(t)$ depends on $B_{1,1}(t)$ and on $B_{2,1}(t)$, and $B_{2,2}(t)$ depends on $B_{2,1}(t)$ and on $B_{3,1}(t)$. In conclusion, $B_{1,3}(t)$ is defined as a function of $B_{1,1}(t), B_{2,1}(t), B_{3,1}(t)$, and such functions depend on (t_1, t_2, t_3, t_4), according to (11.15). In Figure 11.9 is shown the recursive dependence of a basis function $B_{i,4}$ on the knot values it depends on.

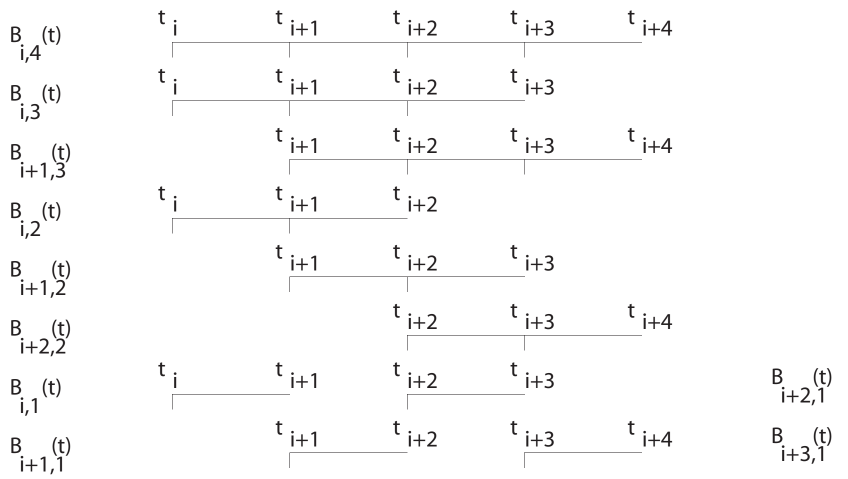

Figure 11.9 Representation of the dependence of $B_{i,4}(t)$ on lower order basis functions and on knot values

Further remarks Let us summarize some properties of non-uniform B-splines.

1. The size of the parameter interval between two adjacent knots is *non-uniform*, i.e. may be variable. Often, integer intervals are used with size

either zero or one. In particular, non-integer knot values may arise when the parametrization of some spline segment must be more or less dense.
2. A non-uniform knot sequence implies that the basis functions may vary from a segment to another. This fact is negative, when considering the computational costs. The evaluation of B-basis polynomials in the general case is quite expensive. Most optimizations require integer knot values.
3. The continuity of segments at a join point may be reduced by raising the multiplicity of a knot value. Correspondingly, a spline segment is reduced to a single point. This fact also implies a reduction of the common subset of control points shared by two spline segments.
4. A control point may easily be interpolated, without introducing linear spline segments, simply by raising the multiplicity of a knot.
5. New control points and knots may easily be inserted, allowing for a powerful local editing and control of the spline shape.

Example 11.3.3 (Continuity reduction at a joint)
In Figure 11.10 an example is shown of reduction of the continuity at a join point by elevation of the multiplicity of a knot. The set of quadratic splines of the figure is generated by Script 11.3.4.

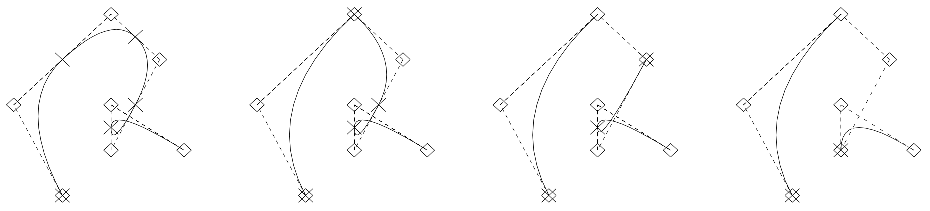

Figure 11.10 Quadratic non-uniform B-spline. Reduction of continuity, initially equal to C^1, by raising the multiplicity of the fourth knot

Script 11.3.4
```
    DEF points = <<0,0>,<-1,2>,<1,4>,<2,3>,<1,1>,<1,2>,<2.5,1>>;

    DisplayNUBspline:< <2,<0,0,0,1,2,3,4,5,5,5>>, points >;
    DisplayNUBspline:< <2,<0,0,0,1,1,2,3,4,4,4>>, points >;
    DisplayNUBspline:< <2,<0,0,0,1,1,1,2,3,3,3>>, points >;
    DisplayNUBspline:< <2,<0,0,0,1,1,1,1,2,2,2>>, points >;
```

11.3.4 Multiresolution Bézier and B-splines

A new set of *multiresolution methods* is becoming popular which utilize a control polygon, as in the Bézier or B-spline case. Instead of using analytic methods to directly calculate points on the curve, such methods successively refine the control polygons into a sequence of control polygons that converge to a curve. Conceptually, multiresolution methods can be described as computing a succession of curves whose image converge to the desired point set. In practice, for a given u, they compute a succession of points that converges to $\boldsymbol{C}(u)$.

de Casteljau algorithm

A typical representative of multiresolution methods for curve generation, also called subdivision methods, is the de Casteljau algorithm for Bézier curves. We first present this method for curves of second degree (parabola), then we discuss its generalization to Bézier curves of arbitrary degree.

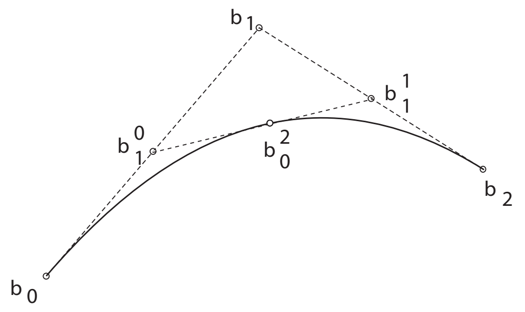

Figure 11.11 de Casteljau algorithm for a curve of second degree

de Casteljau's parabola Let us consider three points $\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3 \in \mathbb{E}^d$ and a scalar $u \in [0,1]$. Define the linear interpolations of point pairs:

$$\begin{aligned} \boldsymbol{b}_0^1(u) &= (1-u)\boldsymbol{b}_0 + u\boldsymbol{b}_1 \\ \boldsymbol{b}_1^1(u) &= (1-u)\boldsymbol{b}_1 + u\boldsymbol{b}_2 \\ \boldsymbol{b}_0^2(u) &= (1-u)\boldsymbol{b}_0^1(u) + u\boldsymbol{b}_1^1(u) \end{aligned}$$

By substituting the first two equations into the third, we get

$$\begin{aligned} \boldsymbol{b}_0^2(u) &= (1-u)\,[(1-u)\boldsymbol{b}_0 + u\boldsymbol{b}_1] + u\,[(1-u)\boldsymbol{b}_1 + u\boldsymbol{b}_2] \\ &= (1-u)^2\boldsymbol{b}_0 + 2u(1-u)\boldsymbol{b}_1 + u^2\boldsymbol{b}_2 \end{aligned}$$

which is the parametric equation of a *parabola*. By considering Figure 11.11, we can see with elementary geometric methods that

$$\begin{aligned} \text{ratio}(\boldsymbol{b}_0, \boldsymbol{b}_0^1, \boldsymbol{b}_1) &= \text{ratio}(\boldsymbol{b}_1, \boldsymbol{b}_1^1, \boldsymbol{b}_2) \\ &= \text{ratio}(\boldsymbol{b}_0^1, \boldsymbol{b}_0^2, \boldsymbol{b}_1^1) \\ &= u/(1-u) \end{aligned}$$

The linear interpolation is affinely invariant, i.e. is invariant with respect to rotation, translation and scaling. The above construction can be fully described by a triangular array of points:

$$\begin{array}{lll} \boldsymbol{b}_0 & & \\ \boldsymbol{b}_1 & \boldsymbol{b}_0^1 & \\ \boldsymbol{b}_2 & \boldsymbol{b}_1^1 & \boldsymbol{b}_0^2 \end{array}$$

General case Consider points $\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in \mathbb{E}^d$ and $u \in [0,1]$. Start by setting $\boldsymbol{b}_i^0(u) = \boldsymbol{b}_i$, and compute a succession of interpolations

$$\boldsymbol{b}_i^r(u) = (1-u)\ \boldsymbol{b}_i^{r-1}(u) + u\ \boldsymbol{b}_{i+1}^{r-1}(u)$$

with $r = 1, \ldots, n$ and with $i = 0, \ldots, n-r$.

It is possible to show that $\boldsymbol{b}_0^n(u)$ will coincide with $\boldsymbol{b}(u)$, i.e. with the point which is the image of the Bézier curve of degree n generated by the control polygon $\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$.

Example 11.3.4 (Cubic de Casteljau scheme)
The triangular array of points used for the generation of a cubic Bézier curve has four rows and four columns. The elements of the first column are the input data, i.e. the original control points. The other columns correspond to algorithm iterations. The element $(4,4)$ of the array is the point generated on the curve for a given u value, as shown below:

$$\begin{array}{llll}
\boldsymbol{b}_0 & & & \\
\boldsymbol{b}_1 & \boldsymbol{b}_0^1 & & \\
\boldsymbol{b}_2 & \boldsymbol{b}_1^1 & \boldsymbol{b}_0^2 & \\
\boldsymbol{b}_3 & \boldsymbol{b}_2^1 & \boldsymbol{b}_1^2 & \boldsymbol{b}_0^3
\end{array}$$

The de Casteljau scheme for a Bézier cubic is given graphically in Figure 11.12.

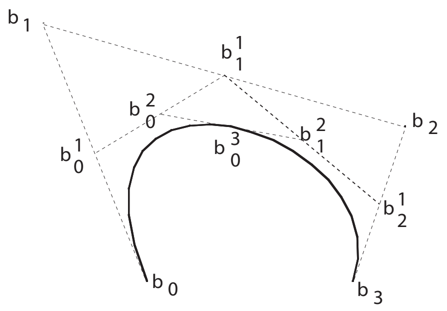

Figure 11.12 The de Casteljau algorithm for a Bézier cubic

11.4 Rational curves and splines

A rational function is a ratio of polynomial functions. A rational curve is a point-valued function of one real parameter, whose coordinate maps are rational. In particular, the coordinate maps of a rational curve or spline in $\mathbb{E}^d$ can be written as ratios of the coordinate maps of a corresponding polynomial curve in $\mathbb{E}^{d+1}$. For example, for a rational curve in $\mathbb{E}^3$ we have

$$\boldsymbol{Q}(u) = \begin{pmatrix} x(u) & y(u) & z(u) \end{pmatrix}^T,$$

with

$$x(u) = \frac{X(u)}{W(u)}, \quad y(u) = \frac{Y(u)}{W(u)}, \quad z(u) = \frac{Z(u)}{W(u)},$$

where $\boldsymbol{C}(u) = \begin{pmatrix} X(u) & Y(u) & Z(u) & W(u) \end{pmatrix}^T$ is a polynomial curve in homogeneous space. The polynomials in homogeneous space are usually in Bézier or B-spline form, although any kind of curve or spline may be used. Non-uniform rational B-splines are called NURB-splines or simply NURBS.

11.4.1 Rational Bézier curves

Rational Bézier curves $\boldsymbol{Q}(u)$ are defined as the projection from the origin on the hyperplane $x_{d+1} = 1$ of a polynomial Bézier curve $\boldsymbol{C}(u)$ in $\mathbb{E}^{d+1}$ homogeneous space. The above hyperplane is easily identified by $\mathbb{E}^d$ by dropping the last coordinate. So, consider

$$\boldsymbol{C}(u) = \boldsymbol{q}_0 B_0^n(u) + \cdots + \boldsymbol{q}_n B_n^n(u),$$

and

$$w(u) = w_0 B_0^n(u) + \cdots + w_n B_n^n(u),$$

where $\boldsymbol{q}_i = (w_i \boldsymbol{b}_i, w_i)^T \in \mathbb{E}^{d+1}$ and $w_i \in \mathbb{R}$. Notice that $\boldsymbol{b}_i \in \mathbb{E}^d$ is the projection of the normalized $\boldsymbol{q}_i$.

It is hence possible to write the rational curve $\boldsymbol{Q}(u)$ of degree n, such that $\boldsymbol{C}(u) = (w(u)\boldsymbol{Q}(u), w(u))$, as a combination of control points $\boldsymbol{b}_i \in \mathbb{E}^d$ with rational functions:

$$\boldsymbol{Q}(u) = \sum_{i=0}^{n} \boldsymbol{b}_i \frac{w_i B_i^n(u)}{w(u)}, \qquad \boldsymbol{b}_i \in \mathbb{E}^d \tag{11.16}$$

The numbers w_i are called *weights* of the control points. When the weights are all unitary, a rational curve coincides with its polynomial counterpart.

From the above it is possible to see that a rational Bézier function is a weighted average of the nonrational basis. It is easy to verify that the rational basis sum to 1 for each u. Rational Bézier curves hence continue to satisfy the properties of nonrational ones, i.e. the affine invariance, the variation diminishing and the containment in the convex hull of control points. Furthermore, rational Bézier curves enjoy also the projective invariance property, that makes them very useful for computer graphics.

Rational Bézier mapping

In the remainder of this section we consider the control points as given in homogeneous space. In particular, for a rational curve in 3D of degree n, we have

$$\boldsymbol{q}_i = (X_i, Y_i, Z_i, W_i)^T, \qquad 0 \le i \le n$$

so that we define a mapping $\boldsymbol{Q} : \mathbb{R} \to \mathbb{E}^3$ as

$$\boldsymbol{Q}(u) = \frac{1}{\sum_{i=0}^n W_i B_i^n(u)} \begin{pmatrix} \sum_{i=0}^n X_i B_i^n(u) \\ \sum_{i=0}^n Y_i B_i^n(u) \\ \sum_{i=0}^n Z_i B_i^n(u) \end{pmatrix} = \frac{1}{\boldsymbol{W} \cdot \boldsymbol{B}^n(u)} \begin{pmatrix} \boldsymbol{X} \cdot \boldsymbol{B}^n(u) \\ \boldsymbol{Y} \cdot \boldsymbol{B}^n(u) \\ \boldsymbol{Z} \cdot \boldsymbol{B}^n(u) \end{pmatrix} \tag{11.17}$$

where $\boldsymbol{B}^n(u)$ is a function vector having as a component the $n+1$ Bernstein basis polynomials, and where $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$ and $\boldsymbol{W}$ are function vectors with $n+1$ constant functions corresponding to the coordinates of control points. So, we have, e.g.

$$\boldsymbol{X} := \big(\ X_i(u) = X_i\ \big), \qquad u \in [0,1],\ 0 \leq i \leq n.$$

Implementation The `PLaSM` implementation of equation (11.17) is straightforward. It will suffice to combine, using a function `RationalBlend` given in the following Script 11.4.1, the sequence of basis functions generated by the expression

```
BernsteinBasis:degree
```

with the sequence of *homogeneous* `ControlPoints`, being the `degree` of the curve equal to the number of control points minus one. The `BernsteinBasis` function is defined in Script 11.2.5.

Script 11.4.1 (Rational Bézier curves)

```
DEF RationalBezier (ControlPoints::IsSeq) =
   RationalBlend:(BernsteinBasis:degree): ControlPoints
WHERE
   degree = LEN:ControlPoints - 1
END
```

Example 11.4.1 (Ellipse as rational Bézier)
We get an exact representation of the quarter of ellipse as a quadratic rational Bézier curve, described [FP80] by homogeneous control points

$$(a,0,1), \quad \frac{\sqrt{2}}{2}(a,b,1), \quad (0,b,1),$$

where a and b are the two radii. Such an ellipse is generated with the center on the origin, the radius of size a in the direction of the x axis and the radius of size b in the direction of the y axis. In Script 11.4.2 the whole curve is generated by four symmetric instances of a quarter of curve.

The ellipse quarter is obtained as a mapping of the rational Bézier quadratic on the interval $[0,1]$. The n parameter establishes the number of segments in a polyhedral approximation of the image of curve quarter. The ellipse may be generated as a whole by a quadratic NURB-spline. See Example 11.4.2. The second order function `Intervals`, used to generate a partition of the interval $[0,a]$ with n segments is given in Script 11.2.1.

The plane ellipse of Figure 11.13a, with radii $\frac{1}{2}$ and 1, respectively, is combined in the last expression above with a 1D polyhedron of length $\frac{1}{2}$, thus generating the elliptical cylinder shown in Figure 11.13b.

11.4.2 NURBS

Rational non-uniform B-splines are normally denoted as NURB splines or simply as NURBS. These splines are very important for both graphics and CAD applications. In particular:

Script 11.4.2

```
DEF ellipse (a,b::IsReal; n::IsINtPos) = STRUCT:< half, S:1:-1, half >
WHERE
   half = STRUCT:< quarter, S:2:-1, quarter >,
   quarter = MAP: mapping: (Intervals:1:n),
   mapping = RationalBezier:<<a,0,1>,<a*c,b*c,c>,<0,b,1>>,
   c = SQRT:2/2
END;

ellipse:<1/2,1,10> * QUOTE:<1/2>
```

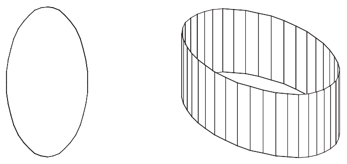
Figure 11.13 (a) Ellipse generated as Bézier curve (b) Surface generated by Cartesian product with an interval

1. Rational curves and splines are invariant with respect to affine and projective transformations. Consequently, to transform or project a NURBS it is sufficient to transform or project its control points, leaving to the graphics hardware the task of sampling or rasterizing the transformed curve.
2. NURBS represent exactly the conic sections, i.e. circles, ellipses, parabolæ, iperbolæ. Such curves are very frequent in mechanical CAD, where several shapes and geometric constructions are based on such geometric primitives.
3. Rational B-splines are very flexible, since (a) the available degrees of freedom concern both degree, control points, knot values and weights; (b) can be locally interpolant or approximant; (c) can alternate spline segments with different degree; and (d) different continuity at join points.
4. They also allow for local variation of "parametrization velocity", or better, allow for modification of the norm of velocity vector along the spline, defined as the derivative of the curve with respect to the arc length. For this purpose it is sufficient to properly modify the knot sequence. This fact allows easy modification of the sampling density of spline points along segments with higher or lower curvature, while maintaining the desired appearance of smoothness.

As a consequence of their usefulness for applications, NURBS are largely available when using geometric libraries or CAD kernels.

Rational B-splines of arbitrary degree

A rational B-spline segment $\boldsymbol{R}_i(t)$ is defined as the projection from the origin on the hyperplane $x_{d+1} = 1$ of a polynomial B-spline segment $\boldsymbol{P}_i(u)$ in $\mathbb{E}^{d+1}$ homogeneous space.

Using the same approach adopted when discussing rational Bézier curves, where $\boldsymbol{q}_i = (w_i\boldsymbol{p}_i, w_i) \in \mathbb{E}^{d+1}$ are the $m+1$ homogeneous control points, the equation of the rational B-spline segment of degree k with $n+1$ knots, may be therefore written as

$$\boldsymbol{R}_i(t) = \sum_{\ell=0}^{k} w_{i-\ell}\, \boldsymbol{p}_{i-\ell} \frac{B_{i-\ell,k+1}(t)}{w(t)} = \sum_{\ell=0}^{k} \boldsymbol{p}_{i-\ell} N_{i-\ell,k+1}(t) \tag{11.18}$$

with $k \le i \le m$, $t \in [t_i, t_{i+1})$, and

$$w(t) = \sum_{\ell=0}^{k} w_{i-\ell} B_{i-\ell,k+1}(t),$$

where $N_{i,h}(t)$ is the non-uniform rational B-basis function of initial value t_i and order h. A global representation of the NURB spline can be given, due to the local support of the $N_{i,h}(t)$ functions, i.e. to the fact that they are zero outside the interval $[t_i, t_{i+h})$. So:

$$\boldsymbol{R}(t) = \bigcup_{i=k}^{m} \boldsymbol{R}_i(t) = \sum_{i=0}^{m} \boldsymbol{p}_i\, N_{i,h}(t), \qquad t \in [t_k, t_{m+1}).$$

NURB splines can be computed as non-uniform B-splines by using homogeneous control points, and finally by dividing the Cartesian coordinate maps times the homogeneous one. This approach will be used in the NURBS implementation given later in this chapter. A more efficient and numerically stable variation of the Cox and de Boor formula for the rational case is given by Farin [Far88], p. 196.

Example 11.4.2 (Circumference as quadratic NURBS)
A circumference may be generated exactly as a closed quadratic NURB spline [FP80] with 9 control points, where the last control point is equal to the first one. In this case the knot sequence must have $9+2+1=12$ elements. To guarantee the passage for the extreme control points the first and last knot must have multiplicity 3. Notice that to guarantee the passage for three other control points, three other knots must have multiplicity 2. As a consequence, using integral knots, the curve is globally parametrized on the $[0,4]$ interval, with each one of the four nondegenerate spline segments parametrized over a unit interval. The `DisplayNURBspline` function is given in Script 11.6.20.

Script 11.4.3 (Circumference)
```
DEF knots = <0,0,0,1,1,2,2,3,3,4,4,4>;
DEF controlPoints = <
   <-1,0,1>, <-:c,c,c>, <0,1,1>, <c,c,c>,
   <1,0,1>, <c,-:c,c>, <0,-1,1>, <-:c,-:c,c>, <-1,0,1> >
WHERE
   c = SQRT:2/2
END;

DEF MarkerSize = 0.10;
DisplayNURBspline:<<2, knots>, controlPoints>;
```

A circumference implementation as a quadratic NURBS is given in Script 11.4.3. The resulting geometry, including a polyline through the control points and two polymarkers through the control and join points, respectively, is shown in Figure 11.14.

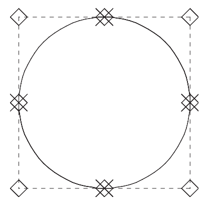
Figure 11.14 Circumference generated as a quadratic NURB spline

Example 11.4.3 (Varying parametrization)
In Script 11.4.4 we vary the number of points sampled on a spline segment, by varying the size of the parameter interval associated with the segment. For this purpose it is sufficient to suitably modify the knot sequence, so that the size of the $[t_i, t_{i+1})$ interval becomes proportional to the number of points sampled on the curve. Let us compare the circumference samples in Figure 11.15. They are respectively produced by:

```
DEF knots = <0,0,0,1,1,2,2,3,3,4,4,4>,
DEF knots = <0,0,0,1,1,2,2,5,5,6,6,6>,
```

Notice that the symbol `splineSampling`, introduced for NURBS in Script 11.6.19, defines the number of sampling subintervals for unit in parameter space of the spline. The `markerSize` and the `markers` operator were defined in Scripts 7.2.8 and 5.1.3, respectively. The `NURBspline` operator will be given in Script 11.6.19.

Script 11.4.4
```
[Polymarker on NURBS circle] DEF markerSize = 0.025;
DEF splineSampling = 6;

(markers ∼ NURBspline:<2,knots>): controlPoints;
```

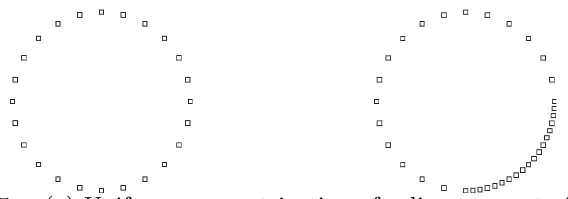
Figure 11.15 (a) Uniform parametrization of spline segments (b) Triply dense parametrization in the third segment (non-reduced to a single point)

11.5 Examples

A quite complex example of shape generation with a NURB spline is presented, and the design of various form features by knot insertion, control point duplication and weight variation is discussed. Then we bring a step further the "umbrella" example already introduced in Section 8.5.15, by substituting some linear rods with curved rods, with a curvature depending on the opening angle.

11.5.1 Shape design with a NURB spline

Let us build a NURB spline example in Script 11.5.1 by discussing the shape design of a sort of "shoe". The profile of Figure 11.16 is generated by evaluating the last expression of Script 11.5.1. The control points $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{11}$ are 12, and the spline has degree $k = 2$. Therefore the knots must be $12 + 2 + 1 = 15$, i.e $t_0, t_1, \ldots, t_{14}$.

The interpolation of first and last control points is enforced by setting t_0 and t_{14} with multiplicity 3. Another interpolation constraint through point $\boldsymbol{p}_4$, i.e. through the "heel tip" of the shoe, is enforced by setting $t_5 = t_6 = 3$. In this way the fourth spline segment $\boldsymbol{R}_5(t)$, $t_5 \leq t < t_6$, is reduced to a point. Since at the joints of this segment the continuity is reduced by one unit, and hence from C^1 to C^0, a sharp-cornered point is obtained.

Script 11.5.1 (NURBS example)

```
DEF splineSampling = 10;
DEF MarkerSize = 0.15;
DEF homCoord = 1;

DisplayNURBspline: < 2, <0,0,0,1,2,3,3,4,5,6,7,8,9,9,9>,
   < <0,5,1>,<4,5,1>,<5,5,1>,<5,4,1>,<5,0,1>,
   <4,0,1>,<4,3,1>,<2,1,1>,<-1,0,1>,<-6,0,homCoord>, <0,3,1>,<0,5,1> >
>;
```

First, remember that the numbering of spline segments $\boldsymbol{R}_i(t)$ goes for $k \leq i \leq m$. To understand why $\boldsymbol{R}_5(t_5) = \boldsymbol{R}_5(t_6) = \boldsymbol{p}_4$, i.e. why the 4-th spline segment reduces to the 5-th control point, consider what follows.

By continuity of spline segments, the last point of segment $\boldsymbol{R}_4(t)$ is equal to the first point of segment $\boldsymbol{R}_5(t)$, i.e. $\boldsymbol{R}_4(t_5) = \boldsymbol{R}_5(t_5)$, so that

$$\boldsymbol{R}_5(t_5) \in \mathrm{conv}\,\{\boldsymbol{p}_3, \boldsymbol{p}_4\} = (\mathrm{conv}\,\{\boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4\} \cap \mathrm{conv}\,\{\boldsymbol{p}_3, \boldsymbol{p}_4, \boldsymbol{p}_5\})$$

Analogously it is $\boldsymbol{R}_5(t_6) = \boldsymbol{R}_6(t_6)$, so that

$$\boldsymbol{R}_6(t_6) \in \mathrm{conv}\,\{\boldsymbol{p}_4, \boldsymbol{p}_5\} = (\mathrm{conv}\,\{\boldsymbol{p}_3, \boldsymbol{p}_4, \boldsymbol{p}_5\} \cap \mathrm{conv}\,\{\boldsymbol{p}_4, \boldsymbol{p}_5, \boldsymbol{p}_6\}).$$

As a consequence of $t_5 = t_6$ we have:

$$\boldsymbol{R}_5(t_5) = \boldsymbol{R}_5(t_6) \in (\mathrm{conv}\,\{\boldsymbol{p}_3, \boldsymbol{p}_4\} \cap \mathrm{conv}\,\{\boldsymbol{p}_4, \boldsymbol{p}_5\}) = \{\boldsymbol{p}_4\}.$$

Summarizing: the spline segment $\boldsymbol{R}_5(t)$, $t_5 \leq t < t_6$, reduced to a single point when $t_5 = t_6$, must coincide with the control point $\boldsymbol{p}_4$. This effect is achieved by simply duplicating a knot value. More in general: by raising the multiplicity of a knot, the spline is attracted towards an internal control point. This is interpolated when the knot multiplicity equals the degree. A multiplicity equal to the order is needed to interpolate the extreme control points.

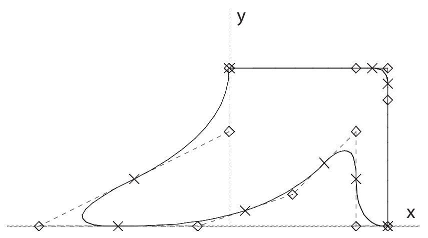

Figure 11.16 Quadratic NURB spline generated by 12 control points and by a suitable knot sequence

Also consider the segments $\boldsymbol{R}_2(t)$ and $\boldsymbol{R}_4(t)$ on the top and right of the shape, respectively, as generated by two *aligned* triples of control points. Since $\boldsymbol{R}_2(t) \subset \operatorname{conv}\{\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2\}$ and $\boldsymbol{R}_4(t) \subset \operatorname{conv}\{\boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4\}$, they must necessarily be line segments.

Example 11.5.1 (Attraction towards a control point)
In Example 11.5.1 all the control points had unit w_i weight. When a control point is edited from $\boldsymbol{p}_i = (x_i, y_i, 1)$ to $\boldsymbol{q}_i = (x_i, y_i, w_i)$, it is actually changed to $\boldsymbol{q}_i = (\frac{x_i}{w_i}, \frac{y_i}{w_i}, 1)$.

The geometric effect of this editing on the homogeneous coordinate of a control point is shown in Figure 11.17, generated by Script 11.5.1 when to the `homCoord` parameter are assigned values $\frac{1}{2}, 1, 2$. Such weights produce the long, medium and short "shoe", respectively.

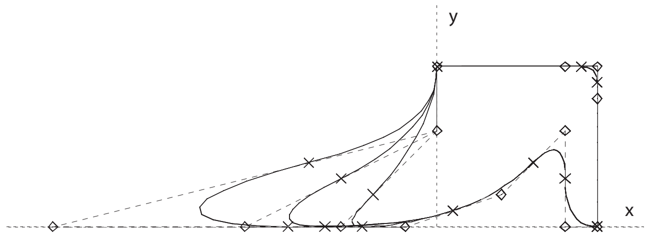

Figure 11.17 Three NURB splines that differ for the weights of a control point

Points and knots insertion When editing a NURB spline it is often necessary to insert a new control point and a new knot without modifying the spline shape, in order to introduce some further degree of freedom in local editing of the shape.

This insertion is easily obtained, in a quadratic NURBS, by adding a new point corresponding to the middle point of a segment of the control polygon, and by doubling the corresponding knot value. Actually, this one is not a good way to add a new knot, as it affects the continuity by introducing a degenerate spline segment. A much better method is the Oslo algorithm [CLR80], or Boehm's method [Böh80]. Notice that, because the reelation between numbers of knots and control points must continue to hold, when a new knot is inserted, one more control point is generated. Actually, adding a knot causes two old control points to be replaced by three new control points.

11.5.2 Umbrella modeling (2): curved rods

We start in this section the step-wise refinement of the geometric modeling of a simplified umbrella already introduced in Section 8.5.3. In particular, the function `RodPair` is here redefined so that the previous `Rod1` component is split into two components `Rod10` and `Rod11`, where `Rod11` is generated as a 1D polyhedral approximation with 4 segments of a quadratic Bézier curve. This one is tangent at the join point with `Rod10` for every opening angle. The `Bezier` mapping generating operator is defined in Script 11.2.5. The definition of the `Intervals` function is given in Script 11.2.1. To fully understand the meaning of the `RodCurve` function, the reader should compare the code below with Script 8.5.14.

Script 11.5.2 (Umbrella (2))

```
DEF RodCurve (len,beta,n::IsReal) = MAP:
   (Bezier:<<0,0,0>,<0,0,len/2>,<len/2 * sin:beta,0,len>>):
      (Intervals:1:n);

DEF RodPair (h, alpha::Isreal) = STRUCT:<
   T:3:h, R:<3,1>:(-:alphaRad), Rod10, T:3:(-:AB): Rod11,
   T:3:(-:AB), R:<3,1>:(2*alphaRad), Rod2 >
WHERE
   alphaRad = alpha*PI/180,
   Rod10 = S:3:-1:(Rod:(AB)),
   Rod11 = S:3:-1:(RodCurve:<AB,-:alphaRad/4,4>),
   Rod2 = S:3:-1:(Rod:AB),
   AB = h*4/10
END;
```

Two different values of the geometric object resulting from the evaluation of the `umbrella` function for two different angles are shown in Figure 11.18. Notice that this function is defined in Script 8.5.15.

11.6 Splines implementation

The `PLaSM` language allows for fast prototyping of parametric curves and splines, for two main reasons.

1. First, since it allows the easy combination with algebraic operators of numbers, points and functions. As the reader knows, a parametric curve is a linear combination of polynomials with control points or vectors.
2. Second, the implementation of parametric curves may be based on the primitive operator `MAP`. For this purpose we only need to define a vector function of a real parameter, which is represented exactly, i.e. symbolically, by `PLaSM`, and a suitable decomposition of the spline domain. A polyhedral

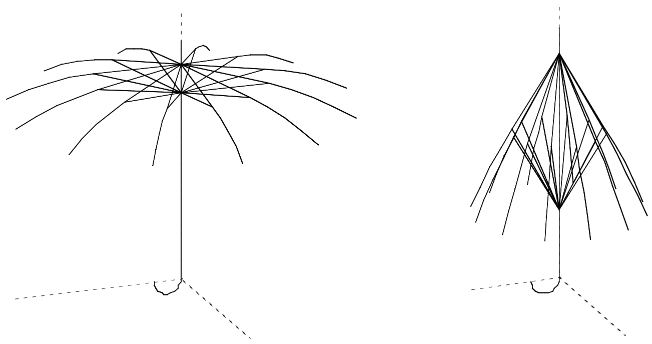

Figure 11.18 New values of `umbrella:<10,80>` and `umbrella:<10,30>` after the redefinition of the function `RodPair`

approximation of the curve image is thus automatically generated, based on a sampling of the curve points.

Indeed, the conceptual elegance of this approach has a trade-off in terms of computational costs. The evaluation of a spline in a functional environment, where the coordinate maps are computed by using algebraic operators in functional spaces, although closer to the mathematical definition of the spline itself, certainly requires higher computation times than its evaluation using imperative languages.

Conversely, the higher evaluation time in a functional environment has a positive revenue in terms of compactness, robustness and readability of the functional code. In particular, the authors believe that the proposed approach may be useful both to teach and fully understand the underlying mathematics of parametric geometry, and to quickly prototype new shape generation tools, as we show in next chapters.

11.6.1 Cubic uniform splines

A `PLaSM` implementation of cubic *cardinal splines* (see Section 11.3.1) and cubic *uniform B-splines* (see Section 11.3.2) is discussed in this section. As usual, our approach is dimension-independent, in order to generate spline images in arbitrary $\mathbb{E}^d$ space. Splines are more often used in 2D or 3D, but higher dimensional splines may be of great interest in some applications, e.g. in robotics, to model a motion as a curve in configuration space.

The design of the `PLaSM` code given in Script 11.6.1 has the specific aim of unifying the evaluation of cubic uniform splines, both cardinal and B-splines. In particular, the `domain` symbol contains a partition of the interval $[0, 1]$ with a `splineSampling` number of subintervals, so that each spline segment is polyhedrally approximated with that number of linear segments. To modify the approximation precision it will suffice to redefine that parameter.

The `Blend` function is used to linearly combine a sequence of basis polynomials with a sequence of control points, thus generating the sequence of coordinate maps associated with a spline segment. The `CubicCardinal` and `CubicUBspline` functions generate a segment image, and are used as partial functions in the implementation of the `Spline` operator given in Script 11.6.2.

Script 11.6.2 defines two sequences of basis polynomials, named `CubicCardinalBasis`

Script 11.6.1 (Cardinal and uniform B-spline maps)

```
DEF splineSampling = 20;
DEF domain = (Intervals:1:splineSampling);

DEF Blend (Basis::IsSeq) (ControlPoints::IsSeq) =
   (AA:innerProd ∼ DISTL):
      < Basis, (AA:(AA:K) ∼ TRANS):ControlPoints >;

DEF CubicCardinal (domain::IsPol) (q1,q2,q3,q4::IsSeq) =
   MAP:(CONS:(Blend:CubicCardinalBasis:<q1,q2,q3,q4>)):domain;

DEF CubicUBspline (domain::IsPol) (q1,q2,q3,q4::IsSeq) =
   MAP:(CONS:(Blend:CubicUBsplineBasis:<q1,q2,q3,q4>)):domain;
```

and `CubicUBsplineBasis`, respectively. In both cases the arithmetics is strictly functional, so that all the sub-expressions produce a function. For this purpose it is necessary to convert each scalar coefficient of basis polynomials into a constant function.

Script 11.6.2 (Cubic cardinal and uniform B-spline splines)

```
DEF CubicCardinalBasis = < C0,C1,C2,C3 >
WHERE
   C0 = ((k:0 - a) * u3) + ((k:2 * a) * u2) - (a * u),
   C1 = ((k:2 - a) * u3) + ((a - k:3) * u2) + (k:1),
   C2 = ((a - k:2) * u3) + ((k:3 - k:2 * a) * u2) + (a * u),
   C3 = (a * u3) - (a * u2),
   u = s1, u2 = u * u, u3 = u2 * u, a = k:1
END;

DEF CubicUBsplineBasis = < B0,B1,B2,B3 >
WHERE
   B0 = a * ((k:3 * u2) - (k:1 * u3) - (k:3 * u) + (k:1)),
   B1 = a * ((k:3 * u3) - (k:6 * u2) + (k:4)),
   B2 = a * ((k:3 * u2) - (k:3 * u3) + (k:3 * u) + (k:1)),
   B3 = a * u3,
   u = s1, u2 = u * u, u3 = u2 * u, a = k:1/k:6
END;

DEF Spline (Curve::IsFun) = STRUCT ∼ AA:Curve ∼ subsets:4

DEF geometryVector =
<<-3,6>,<-4,2>,<-3,-1>,<-1,1>,<1.5,1.5>,<3,4>,<5,5>,<7,2>,<6,-2>,<2,-3>>;

Spline:(CubicUBspline:domain):geometryVector;
```

The `Spline` function given above has the purpose of generating a polyhedral approximation of a spline image, as the `STRUCT` aggregation of the $(1, d)$-polyhedra generated by the `Curve` vector function applied to all subsets of control points of length 4. Notice that `Curve` is the formal parameter of the `Spline` function, and is applied to all point quadruples generated by the `subsets:4` function, in turn applied, at run time, to a control point sequence.[6] The definition of the `subsets` operator is given in Script 11.6.11.

Computation and drawing of joints In Script 11.6.3 we compute the join points between spline segments. The approach used may look interesting:

1. the `knotzero` symbol is loaded with the $(0,1)$-complex made by the single point $(0) \in \mathbb{E}$;
2. `knotzero` is used as domain of the formal parameter `TheSpline`;
3. the desired spline (ether `CubicCardinal` or `CubicUBspline`) is passed as actual argument of the `Joints` function; so that the expression `Spline:(TheSpline:knotzero)` computes the $(0,d)$-complex
$$\{Q_i(0) | 1 \leq i \leq m-2\};$$
4. the internal data structure representing this complex is "unpacked" by the `UKPOL` primitive;
5. its vertices, i.e. the join points, are passed to the `polymarker:2` function.

Script 11.6.3
```
    DEF knotzero = MKPOL:<<<0>>,<<1>>,<<1>>>;

    DEF Joints (TheSpline::IsFun) =
       polymarker:2 ~ S1 ~ UKPOL ~ Spline:(TheSpline:knotzero);

    Joints:CubicUBSpline:geometryVector;
```

The `Joints` function is used with a double application, as in Script 11.6.3.

Display function In Script 11.6.4 a function `DisplayAll` is given as user interface for uniform splines. It draws at the same time the spline, the control polygon and two polymarkers, respectively corresponding to control points and to the join point between spline segments.

Let us evaluate the last expression of the script. The generated geometric value is shown in Figure 11.19a. The reader might note that such a geometric result is too "bumped" when using as tangent vectors in $\boldsymbol{p}_i$ the $\boldsymbol{s}_i = \boldsymbol{p}_{i+1} - \boldsymbol{p}_{i-1}$ differences. In Figure 11.19b is shown the cubic cardinal spline generated with extreme tangents $\boldsymbol{s}_i = \frac{1}{2}(\boldsymbol{p}_{i+1} - \boldsymbol{p}_{i-1})$. The `geometryVector` is the one given in Script 11.6.2. Notice that in a cardinal spline the joints coincide with the (internal) control points.

Interpolation of extreme points As we know, a cardinal spline is not defined at the extreme control points, and the uniform B-spline normally passes for none of control points. To force the passage for $\boldsymbol{p}_0$ and $\boldsymbol{p}_m$ it is necessary to set their multiplicity to 2 in case of cardinal, and to 3 for uniform cubic B-spline.

[6] A mixed programming approach is used here, with both a formal parameter in λ-style, and an implicit parameter, in proper `FL`-style.

Script 11.6.4

```
DEF DisplayAll (MySpline::IsFun)(points::IsSeq) = (STRUCT ∼ [
   Spline:(MySpline:domain),
   Joints:MySpline,
   polyline,
   polymarker:1
]): points;

DisplayAll:CubicCardinal:geometryVector;
```

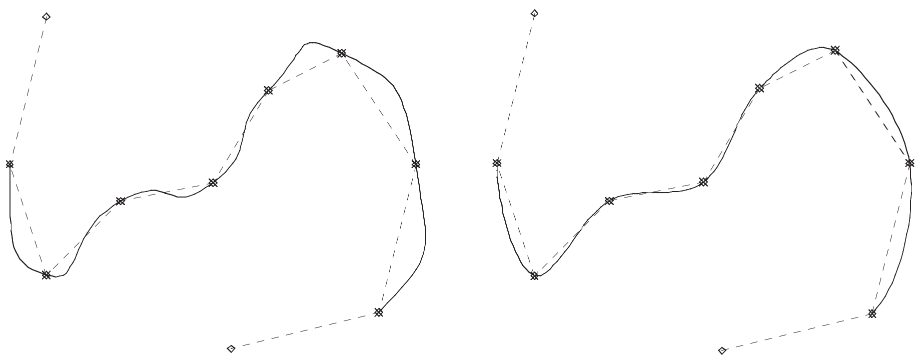

Figure 11.19 (a) Cubic cardinal spline with scalar $h = 1$ multiplying tangents at control points (b) Same spline with $h = \frac{1}{2}$

Therefore, in Script 11.6.5 we give a `MultipleExtremes` function which does the job. The two geometries generated by last expressions of the script are shown in Figure 11.20.

Script 11.6.5

```
DEF MultipleExtremes (multiplicity::IsIntPos) =
   CAT ∼ [ Firsts, ID, Lasts ]
WHERE
   Firsts = #:(multiplicity - 1) ∼ FIRST,
   Lasts = #:(multiplicity - 1) ∼ LAST
END;

DEF geometryVector = <<3,0>,<6,0>,<7,3>,<6,6>,<3,7>,<0,6>,<0,3>>;
DisplayAll:CubicCardinal: (MultipleExtremes:2:geometryVector);
DisplayAll:CubicUBSpline: (MultipleExtremes:3:geometryVector);
```

Example 11.6.1 (Closed cubic uniform B-spline)
Two adjacent spline segments $\boldsymbol{Q}_i(t)$ and $\boldsymbol{Q}_{i+1}(t)$ share three common control points. So, to enforce a C^2-continuity between the first and last segments $\boldsymbol{Q}_3(t)$ and $\boldsymbol{Q}_m(t)$ of a cubic B-spline it is sufficient to set the first three points equal to the last three, i.e. , to require:

$$\boldsymbol{p}_0 = \boldsymbol{p}_{m-3}, \qquad \boldsymbol{p}_1 = \boldsymbol{p}_{m-2}, \qquad \boldsymbol{p}_2 = \boldsymbol{p}_{m-1}.$$

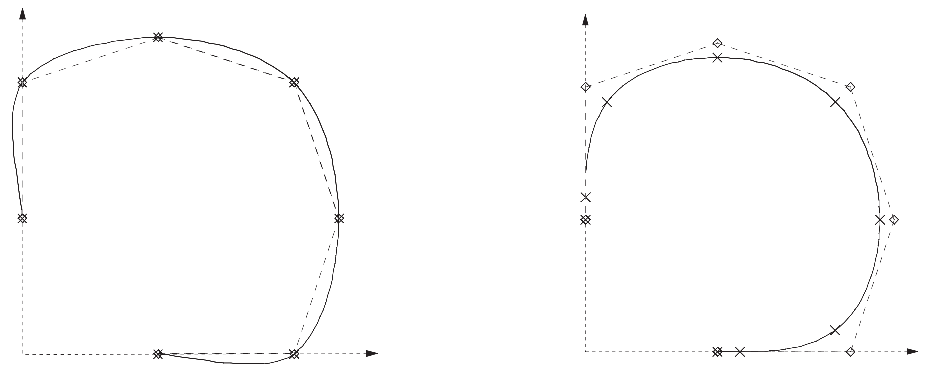

Figure 11.20 Interpolation of extreme control points: (a) cubic cardinal (b) cubic uniform B-spline. Notice the presence of linear segments

The generated spline is shown in Figure 11.21a.

```
    DisplayAll:CubicUBSpline:
     <<1,-1>,<-1,2>,<1,4>, <2,3>,<0,0>,<2.5,1>,<3,-1>, <1,-1>,<-1,2>,<1,4>>;
```

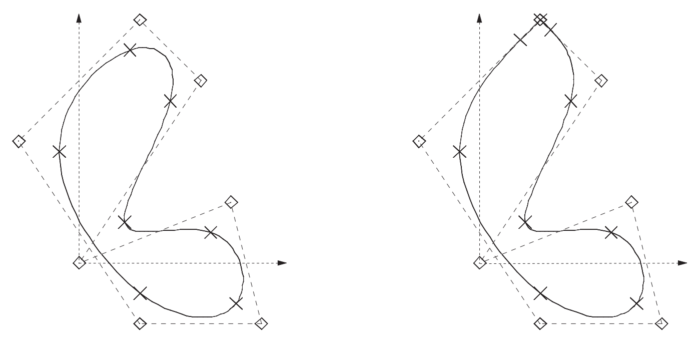

Figure 11.21 (a) Closed cubic uniform B-spline (b) Interpolation of a control point

Example 11.6.2 (Interpolation of a control point)

Consider the closed cubic uniform B-spline of the previous example, and enforce a constraint of interpolation of an internal control point, e.g. $\boldsymbol{p}_2$. This interpolation is achieved by triplicating the point. Therefore, two new spline segments are created (one for each new point instance). Since (a) both segments are generated by the triplet point and by another point, and (b) they must be contained in the convex hull of their control points, then it follows that they are line segments. A discontinuity on the second derivative hence arises at their joints.

```
    DisplayAll:CubicUBSpline:<<1,-1>,<-1,2>, <1,4>,<1,4>,<1,4>, <2,3>,
       <0,0>,<2.5,1>,<3,-1>, <1,-1>,<-1,2>,<1,4>>;
```

The generated spline is shown in Figure 11.21b.

11.6.2 Non-uniform B-splines

In this section we discuss a possible PLaSM implementation of non-uniform polynomial B-splines. In particular, we first approach the Cox and de Boor formula (11.14) for computation of B-basis functions up to order 4, then we abstract such implementation to B-bases of arbitrary order. Finally, the combinatorial power of a FL-like programming environment is exploited to combine control points and basis functions.

Basis functions up to order 4

We start by giving explicit versions of Cox and de Boor recursive formula for orders $1, 2, 3$ and 4. The implementation of such non-recursive approach will give the insight needed for a recursive implementation of arbitrary order. The explicit formulas to compute $B_{i,h}(t)$, with $1 \le h \le 4$, follow:

$$B_{i,1}(t) = \begin{cases} 1, & \text{if } t_i \le t \le t_{i+1} \\ 0, & \text{otherwise,} \end{cases}$$

$$B_{i,2}(t) = \frac{t - t_i}{t_{i+1} - t_i} B_{i,1}(t) + \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} B_{i+1,1}(t)$$

$$B_{i,3}(t) = \frac{t - t_i}{t_{i+2} - t_i} B_{i,2}(t) + \frac{t_{i+3} - t}{t_{i+3} - t_{i+1}} B_{i+1,2}(t)$$

$$B_{i,4}(t) = \frac{t - t_i}{t_{i+3} - t_i} B_{i,3}(t) + \frac{t_{i+4} - t}{t_{i+4} - t_{i+1}} B_{i+1,3}(t)$$

In Script 11.6.6 four B-basis functions respectively denoted Bi1, Bi2, Bi3 and Bi4 are given. To each $B_{i,h}$ implementation is associated as formal parameter the subsequence $(t_i, \ldots, t_{i+h})$ of knots it depends on. In particular, the $t_i, t_{i+1}, t_{i+2}, t_{i+3}, t_{i+4}$ parameters are denoted as ti0,ti1,ti2,ti3,ti4, respectively. The division is operated by a special-purpose myDiv operator, since we need to manage the division by zero, that arises when knots have a multiplicity greater then one.

Special division An *ad hoc* functional division operator MyDiv is given in Script 11.6.7. In particular, MyDiv is applied to a and b parameters, where a is a function and $b = (b_1, b_2)$ is a pair of numbers, applying the following rules when the denominator $\underline{b_1} - \underline{b_2}$ is zero: $a/\underline{0} = \underline{0}$ if $a \ne \underline{0}$, else $\underline{0}/\underline{0} = \underline{1}$.

Example 11.6.3
The implementation of $B_{i,h}$ as a mapping $\mathbb{R} \to \mathbb{R}$ in Script 11.6.6 allows one to MAP each generated mapping over a polyhedral decomposition of the parameter interval. For this purpose it must be applied to domain points given as *sequences* of coordinates, actually with only one coordinate. So, we may write:

```
Bi4:< 0,0,1,2,2 > ≡ An-Anonymous-Function
Bi4:< 0,0,1,2,2 >:< 1.5 > ≡ 0.25
```

Script 11.6.6

```
DEF Bi1 (ti0,ti1::IsReal) =
   IF:< AND ~ [GE:ti0 ~ S1,LT:ti1 ~ S1], K:1, K:0 >;

DEF Bi2 (ti0,ti1,ti2::IsReal) =
   (((S1 - K:ti0) MyDiv <ti1,ti0>) * Bi1:<ti0,ti1>) +
   (((K:ti2 - S1) MyDiv <ti2,ti1>) * Bi1:<ti1,ti2>);

DEF Bi3 (ti0,ti1,ti2,ti3::IsReal) =
   (((S1 - K:ti0) MyDiv <ti2,ti0>) * Bi2:<ti0,ti1,ti2>) +
   (((K:ti3 - S1) MyDiv <ti3,ti1>) * Bi2:<ti1,ti2,ti3>);

DEF Bi4 (ti0,ti1,ti2,ti3,ti4::IsReal) =
   (((S1 - K:ti0) MyDiv <ti3,ti0>) * Bi3:<ti0,ti1,ti2,ti3>) +
   (((K:ti4 - S1) MyDiv <ti4,ti1>) * Bi3:<ti1,ti2,ti3,ti4>);
```

Script 11.6.7

```
DEF MyDiv (a::IsFun; b::IsSeq) =
   IF:< EQ ~ [s1,s2], StrangeValues, K:(a/c) >:b
WHERE
   StrangeValues = filter ~ a, % precondition a/0       %
   filter =                    % if a = 0 then 1 else 0 %
      IF:<testOnZero, (K ~ K):1, (K ~ K):0>,
   testOnZero = (LT:precision) ~ (ABS ~ - ~ [ID,K:0]),
   precision = 1E-12,
   c = (K ~ -):b
END;
```

Basis functions of arbitrary order

Recursive formula (11.14) is implemented in Script 11.6.8 as a function named `DeBoor`, with a formal parameter `knots` which is a subsequence of knot values. The `DeBoor` function recognizes the basic case of order 1 from the length 2 of the input sequence. In such case a step function of constant value 1 is generated in the parameter interval, and 0 outside. Formula (11.14) is directly implemented for the recursive case.

Example 11.6.4 (Linear and quadratics B-basis functions)
We show in this example some non-uniform B-basis functions of order 2 and 3, i.e. of first and second degree, generated by the knot sequence $(0, 0, 0, 1, 2, 3, 4, 5, 5, 5)$. The domain of B-basis functions up to degree 2 generated by this knot sequence is $[0, 5]$. The graphs given in Figure 11.22 are generated by mapping these functions over a domain partition into 100 subintervals.

Let us first derive the second-order B-basis functions $B_{5,2}$ and $B_{6,2}$. The first one is generated by knot subsequence $(t_5, t_6, t_7) = (3, 4, 5)$; the second one by subsequence $(t_6, t_7, t_8) = (4, 5, 5)$. Their graphs (together with those of their first-order generating functions) are given in Figure 11.22a. The generating expressions of the graphs are given in Script 11.6.9.

In Figure 11.22b we show the graphs of third-order B-basis functions $B_{4,3}$, $B_{5,3}$ and $B_{6,3}$, together with those of their second-order generating maps. In particular, $B_{4,3}$

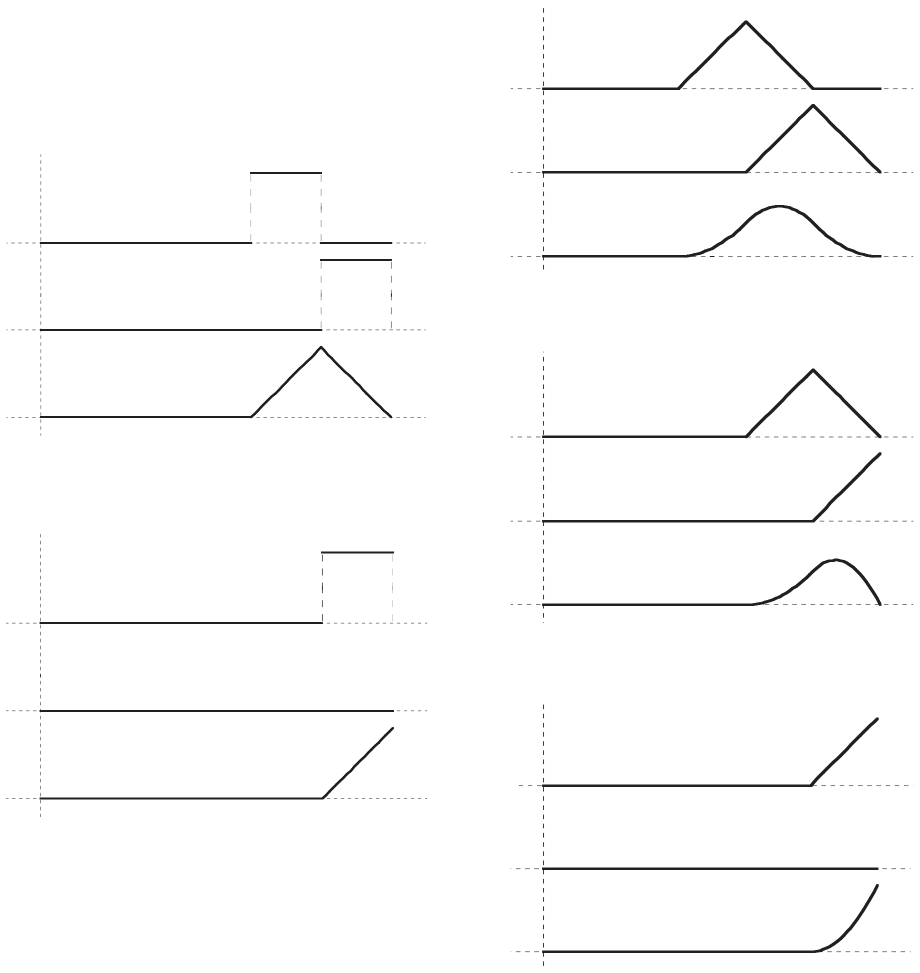

Figure 11.22 Graphs of some non-uniform B-spline functions generated by knot sequence $(0, 0, 0, 1, 2, 3, 4, 5, 5, 5)$. (a) Right, from top to bottom: graphs of $B_{5,1}(t)$, $B_{6,1}(t)$ and of $B_{5,2}(t)$ generated by them; graphs of $B_{6,1}(t)$, $B_{7,1}(t)$ and of $B_{6,2}(t)$ derived from them. (b) Left, from top to bottom: graphs of $B_{4,2}(t)$, $B_{5,2}(t)$ and of the generated $B_{4,3}(t)$; graphs of $B_{5,2}(t)$, $B_{6,2}(t)$ and of the generated $B_{5,3}(t)$; functions $B_{6,2}(t)$, $B_{7,2}(t)$ and $B_{6,3}(t)$ derived from them.

Script 11.6.8
```
DEF DeBoor (knots::IsSeqOf:IsReal) =
   IF:< C:EQ:2 ~ LEN, basicCase, recursiveCase >:knots
WHERE
   ui0 = S1:knots,
   ui1 = S2:knots,
   ui3 = (LAST ~ leftKnots):knots,
   ui4 = LAST:knots, u = S1,
   rightKnots = TAIL,
   leftKnots = REVERSE ~ TAIL ~ REVERSE,
   basicCase = K:(IF:<AND ~ [GE:ui0 ~ S1, LT:ui1 ~ S1], K:1, K:0>),
   recursiveCase = + ~ [
      K:((u - K:ui0) MyDiv <ui3, ui0>) RAISE:* (DeBoor ~  leftKnots),
      K:((K:ui4 - u) MyDiv <ui4, ui1>) RAISE:* (DeBoor ~ rightKnots)
   ]
END;
```

Script 11.6.9
```
DEF domain = Intervals:5:100;
STRUCT:<
   MAP:[s1,DeBoor:<3,4>]:domain, T:2:-1.25,
   MAP:[s1,DeBoor:<4,5>]:domain, T:2:-1.25,
   MAP:[s1,DeBoor:<3,4,5>]:domain >;
STRUCT:<
   MAP:[s1,DeBoor:<4,5>]:domain, T:2:-1.25,
   MAP:[s1,DeBoor:<5,5>]:domain, T:2:-1.25,
   MAP:[s1,DeBoor:<4,5,5>]:domain >;
```

is generated by the knot subsequence $(t_4, t_5, t_6, t_7) = (2, 3, 4, 5)$. $B_{5,3}$ corresponds to $(t_5, t_6, t_7, t_8) = (3, 4, 5, 5)$; $B_{6,3}$ is produced starting from $(t_6, t_7, t_8, t_9) = (4, 5, 5, 5)$. The generating expressions are in Script 11.6.10.

Subsequences In Script 11.6.11 a `subsets` operator is defined that, starting from every `seq` sequence, returns the set of subsequences of length `h`.

Non-uniform B-basis of arbitrary order

The function `BsplineBasis` in Script 11.6.12 returns the set of basis functions $\{B_{i,h}(t)\}$, $0 \leq i \leq m$ for arbitrary `order` and `knots` sequence.

Example 11.6.5 (Graphs of B-spline basis)
Let us build the graphs of B-bases of orders 3 (quadratics) and 2 (linear) generated by the $(0, 0, 0, 1, 2, 3, 4, 5, 5, 5)$ knot sequence, by evaluating the `PLaSM` code in Script 11.6.13. Notice that two B-basis functions of order 2 on the given knot sequence are identically zero.

Script 11.6.10

```
STRUCT:<
   MAP:[s1,DeBoor:<2,3,4>]:domain, T:2:-1.25,
   MAP:[s1,DeBoor:<3,4,5>]:domain, T:2:-1.25,
   MAP:[s1,DeBoor:<2,3,4,5>]:domain >;
STRUCT:<
   MAP:[s1,DeBoor:<3,4,5>]:domain, T:2:-1.25,
   MAP:[s1,DeBoor:<4,5,5>]:domain, T:2:-1.25,
   MAP:[s1,DeBoor:<3,4,5,5>]:domain >;
STRUCT:<
   MAP:[s1,DeBoor:<4,5,5>]:domain, T:2:-1.25,
   MAP:[s1,DeBoor:<5,5,5>]:domain, T:2:-1.25,
   MAP:[s1,DeBoor:<4,5,5,5>]:domain >;
```

Script 11.6.11 (Subsequences)

```
DEF subsets (h::IsIntPos)(seq::IsSeq) =
   (CONS ~ AA:(AS:SEL ~ FROMTO ~ [ID - K:h + K:1,ID])):(h..LEN:seq):seq;

subsets:5:<0,0,0,1,1,2,2,3,3,3> ≡ <<0,0,0,1,1>,<0,0,1,1,2>,
   <0,1,1,2,2>,<1,1,2,2,3>,<1,2,2,3,3>,<2,2,3,3,3>>
```

Programming interface

In this section we discuss the programming interface to non-uniform B-splines of arbitrary order. In particular, we give in Script 11.6.14 the function `Bspline`, which is the kernel of the implementation, and some user-callable functions. The `Bspline` function is a good summary of the theory discussed in the above sections. We illustrate its behavior by discussing the meaning of the symbols, both formal parameters and local functions, in its functional environment.

`points` is the sequence of control points $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_m$;

`knots` is the knot sequence $t_0, \ldots, t_n$;

`degree` is the spline degree k. No check is performed to enforce the constraint $n = m + k + 1$, which is left to the user. This choice seems preferable to requiring the only input of control points and knots sequences;

`dom` is a polyhedrally-type parameter, usually a suitable partition of the interval $[0, 1]$, which is affinely transformed to provide the domain for the mapping of each spline segment;

`order` is the spline order $h = k + 1$;

`basis` is the non-uniform B-spline basis $\{B_{i,h}(t)\}$ of `order` h, $0 \leq i \leq m$. It is generated by invoking the `BsplineBasis` function;

Script 11.6.12 (Non-uniform B-spline basis)

```
DEF BsplineBasis (order::IsInt) (knots::IsSeqOf:IsReal) =
   (AA:DeBoor ~ subsets:(order+1)): knots;
```

Script 11.6.13 (Graphs of quadratic and linear B-bases)

```
DEF domain = Intervals:5:100;

(STRUCT ∼ (CONS ∼ AA:(MAP ∼ CONS) ∼ DISTL):
   <s1,BsplineBasis:3:<0,0,0,1,2,3,4,5,5,5>>):domain

(STRUCT ∼ (CONS ∼ AA:(MAP ∼ CONS) ∼ DISTL):
   <s1,BsplineBasis:2:<0,0,0,1,2,3,4,5,5,5>>):domain
```

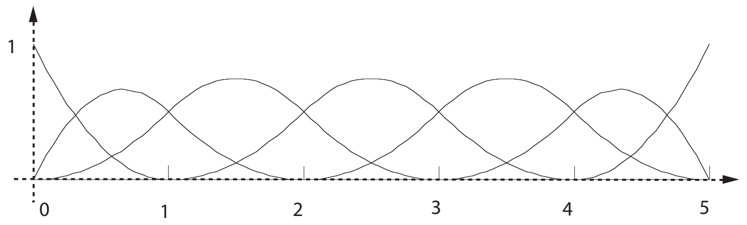

Figure 11.23 B-spline basis functions of order 3 (quadratics) on the $(0, 0, 0, 1, 2, 3, 4, 5, 5, 5)$ knot sequence

`segments` contains the sequence of polyhedral approximations of spline segments $\boldsymbol{Q}_i(t)$, $t_i \leq t \leq t_{i+1}$, $k \leq i \leq m$. It is generated starting from a triplet of sequences. This triplet contains $m - k + 1$ copies of the `MAP` operator, $m - k + 1$ vector functions of one parameter (curves) stored in `segmentmaps`, and $m - k + 1$ polyhedral domains the curves must be applied on, stored in the `poldoms` sequence. This triplet of sequences is transposed into a sequence of triples; then the ones non-degenerating into a single point (when $t_i = t_{i+1}$) are selected and evaluated;

`segmentmaps` is the sequence of point-valued functions $\boldsymbol{Q}_i : \mathbb{R} \to \mathbb{E}^d$ used to generate the spline segments. Each $\boldsymbol{Q}_i$ is generated as product

$$\begin{pmatrix} B_{i-k,h} & \cdots & B_{i,h} \end{pmatrix}\begin{pmatrix} \boldsymbol{p}_{i-k} & \cdots & \boldsymbol{p}_i \end{pmatrix}^T.$$

by the `Blend` function given in Script 11.6.1. In particular, such sequence of maps is generated by transposing the `<basis, points>` pair, then by generating all the ordered subsets of `order` length, and finally by `Blend`-ing all the generated pairs of subsets;

`domain` contains the subset of knots $(t_k, \ldots, t_{m+1})$. Externally to its extreme values, the spline is not defined;

`subdomains` is the sequence of all adjacent pairs of knots extracted from the spline `domain`, i.e. $((t_k, t_{k+1}), (t_{k+1}, t_{k+2}), \ldots, (t_m, t_{m+1}))$.

`poldoms` is the sequence of polyhedral domains generated by scaling the `dom` object with parameter $t_{i+1} - t_i$, and by translating with parameter t_i, for each pair in `subdomains`;

`non-empty` is a `PLaSM` function that, applied to a set pairs, returns as output a filter function that, when applied to another sequence of the same length,

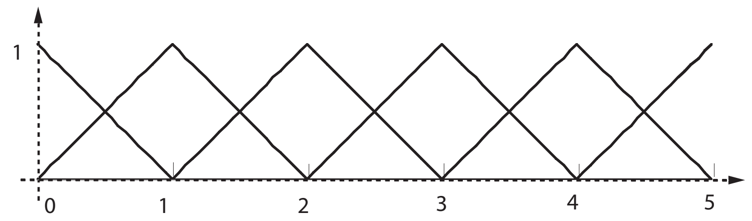

Figure 11.24 B-spline basis functions of order 2 (linear) on the $(0, 0, 0, 1, 2, 3, 4, 5, 5, 5)$ knot sequence

returns the ordered subset corresponding to positions where the former pair has no coincident elements;

`nondegenerate` is the filter function returned by the `non-empty:subdomains` expression. It is used to filter out, i.e. to not compute the spline segments reducing to a point, which otherwise would be produced in vain with a multiplicity equal to the number of point samples in the segment domain. We remember that a degenerate segment is produced by each pair of equal subsequent knots in `subdomains`.

The `intervals` and `knotzero` function are given in Scripts 11.2.1 and 11.6.3, respectively. Finally, `NUBspline` and `NUBsplineJoints` are two partial functions derived from `Bspline`.

Script 11.6.14
```
DEF splineSampling = 10;
DEF NUBspline = Bspline:(intervals:1:splineSampling);
DEF NUBsplineJoints = Bspline:Knotzero;

DEF Bspline (dom::Ispol)(degree::Isint)(knots::IsSeq)(points::IsSeq) =
   STRUCT:segments
WHERE
   order = degree + 1,
   basis = BsplineBasis:order:knots,
   segments = (AA:(INSL:APPLY) ~ nondegenerate ~ TRANS):
      < #:(LEN:segmentmaps):MAP, segmentmaps, poldoms >
   segmentmaps = (AA:(INSL:APPLY ~ AL) ~ DISTL):
      < CONS ~ Blend, (AA:TRANS ~ subsets:order ~ TRANS):< basis, points >>,
   domain = AS:SEL:(order..(LEN:points+1)): knots,
   subdomains = subsets:2:domain,
   poldoms = AA:(STRUCT ~ [T:1~S1, S:1~(S2 - S1), K:dom]):subdomains,
   non-empty = AS:SEL ~ CAT ~ AA:(IF:< EQ~S1, K:<>, [S2] >)
      ~ TRANS ~ [ID,INTSTO ~ LEN],
   nondegenerate = non-empty: subdomains,
END;
```

Display function A non-uniform B-spline may be displayed by using the `DisplayNUBspline` function given in Script 11.6.15. The generated geometry contains the spline, its control polygon and two polymarkers, marking both the control points and the join points among spline segments.

Script 11.6.15

```
DEF DisplayNUBspline (degree::Isint; knots::IsSeq ; points::IsSeq ) =
(STRUCT ~ [
   IF:< K:(GT:0:degree), NUBspline:degree:knots, polymarker:3 >,
   polymarker:2 ~ S1 ~ UKPOL ~ NUBsplineJoints:degree:knots ,
   polyline,
   polymarker:1
]): points;
```

Example 11.6.6 (Nonuniform B-splines)
An example of quadratic non-uniform B-spline in $\mathbb{E}^3$ is given below. The generated curve is shown in Figure 11.25. Notice that control `points` may be defined as a finite subset of $\mathbb{E}^d$, with arbitrary d.

```
DEF points = <<0.1,0>,<2,0>,<6,1.5>,<6,4>,<2,5.5>,<2,6>,<3.5,6.5>>;
DEF out = DisplayNUBspline:<2, <0,0,0,1,2,3,4,5,5,5>, points >;
SVG:out:10:'out.svg';
```

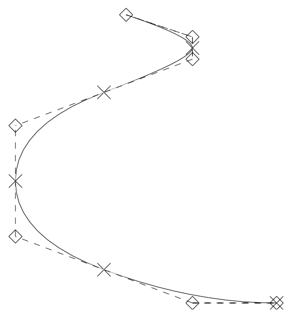

Figure 11.25 Non-uniform quadratic B-spline in $\mathbb{E}^2$

Example 11.6.7 (Degree raising)
We show in Figure 11.26 four non-uniform B-splines of different degrees defined by the same control polygon, generated by Script 11.6.16. It is required that all the splines interpolate the extreme control points.

Notice that the quadratic spline is tangent to the control polygon in the Meanpoint of each segment, but in the extreme segments. It can be shown (see Farin [Far88]) that such a spline, defined over knot intervals of size either 0 or 1, is a sequence of quadratics Bézier curves defined by a control polygon of type

$$\ldots, \boldsymbol{p}_i, \frac{\boldsymbol{p}_i + \boldsymbol{p}_{i+1}}{2}, \boldsymbol{p}_{i+1}, \ldots$$

Script 11.6.16

```
DEF MarkerSize = 0.15;
DEF points = <<0,0>,<-1,2>,<1,4>,<2,3>,<1,1>,<1,2>,<2.5,1>>;

DEF out = STRUCT:<
   DisplayNUBspline:< 0,<0,1,2,3,4,5,6,7>, points >, T:1:4,
   DisplayNUBspline:< 1,<0,0,1,2,3,4,5,6,6>, points >, T:1:4,
   DisplayNUBspline:< 2,<0,0,0,1,2,3,4,5,5,5>, points >, T:1:4,
   DisplayNUBspline:< 3,<0,0,0,0,1,2,3,4,4,4,4>, points >
>;
```

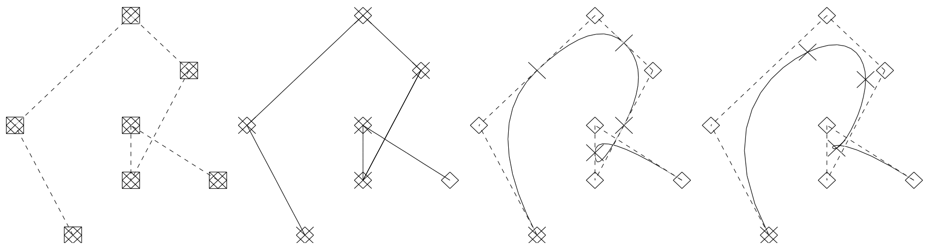

Figure 11.26 Non-uniform B-splines of degree $0, 1, 2$ with 7 points. Notice the reduction of number of segments $(6, 5, 4$, respectively) while raising the degree

Example 11.6.8 (Continuity reduction at joints)
We give finally an example of continuity reduction at a joint by reducing the multiplicity of a knot value. Let us use the same control `points` sequence defined in Example 11.6.7. It results that two adjacent B-spline segments of degree k are C^{k-r} at the image of a knot with r multiplicity, as shown in Figure 11.27.

```
DisplayNUBspline:< <2,<0,0,0,1,2,3,4,5,5,5>>, points >;
DisplayNUBspline:< <2,<0,0,0,1,1,2,3,4,4,4>>, points >;
DisplayNUBspline:< <2,<0,0,0,1,1,1,2,3,3,3>>, points >;
```

11.6.3 Non-uniform rational B-splines

In this last section we quickly discuss a straightforward PLaSM implementation of NURB splines. As always, our approach is dimension-independent, so that the given operators may be applied to homogeneous control points with $d+1$ coordinates, with

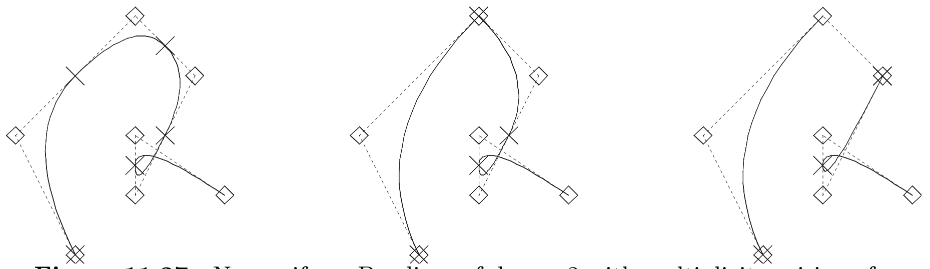

Figure 11.27 Non-uniform B-splines of degree 2 with multiplicity raising of a knot. Notice that the number of non-degenerate segments decreases

arbitrary d, so getting splines of dimension $(1, d)$. The user interface, as well as the implementation itself are extremely similar to those of non-uniform polynomial splines.

NURB mapping First define a `RationalBlend` operator to linearly combine a sequence of blending functions and a sequence of control points. The evaluation of the `RationalBlend` given in Script 11.6.17 returns a `CONS`-ed vector function of coordinate maps that is usable in mapping a partition of a real interval.

Script 11.6.17

```
DEF RationalBlend=
   Rationalize:( Blend:Basis:ControlPoints ) ;
```

Notice that the above `RationalBlend` operator differs from the `Blend` function of Script 11.6.1, only because the introduction of a `Rationalize` function, given below, in the computational pipeline.

Rationalize operator The `Rationalize` operator, given in Script 11.6.18, rationalizes a sequence of either numbers of functions, by dividing the other elements by the last element of the sequence , which is finally dropped out. The function `RTAIL` to eliminate the last element of a sequence, is given in Script 4.2.5.

Script 11.6.18

```
DEF Rationalize (functions::IsSeq) =
   (RTAIL ~ AA:NoExceptionDiv ~ DISTR):<functions,nth>
WHERE
   nth = LAST:functions,
   NoExceptionDiv = IF:<EQ ~ [S2,K:0], S1, (S1 RAISE:/ S2)>
END;
```

Notice that the division is performed in any case, even with zero denominator. In such cases the first components remain invariant by definition, as we show below.

```
Rationalize:<1, 2, 3, 4, 5> ≡ <1/5, 2/5, 3/5, 4/5>
Rationalize:<1, 2, 3, 4, 0> ≡ <1, 2, 3, 4>
```

User interface It is the same as in the non-rational case discussed in some sections above. Only the names of the given functions change slightly. The only difference in the coding of the implementation concerns the invoking of the function `RationalBlend` instead than the `Blend` operator in the `RationalBspline` function of Script 11.6.19. This would otherwise be a perfect copy of the `Bspline` function given in Script 11.6.14.

Analogous changes occur in the `DisplayNURBspline` function given below. Just notice that before be passed to the `polyline` or `polymarker` operators, the homogeneous control points of the NURB spline must be rationalized.

Script 11.6.19

```
DEF NURBspline = RationalBspline:(intervals:1:splineSampling);
DEF NURBsplineJoints = RationalBspline:Knotzero;

DEF RationalBspline
 (dom::Ispol)(degree::Isint)(knots::IsSeq)(points::IsSeq) =
   STRUCT:segments
WHERE
    ...
   segmentmaps = (AA:(INSL:APPLY ∼ AL) ∼ DISTL):
      < CONS ∼ RationalBlend,
         (AA:TRANS ∼ subsets:order ∼ TRANS):<basis, points> >,
    ...
END;
```

Script 11.6.20

```
DEF DisplayNURBspline (degree::Isint; knots::IsSeq ; points::IsSeq ) =
(STRUCT ∼ [
   IF:< K:(GT:0:degree), NURBspline:degree:knots, polymarker:3 >,
   polymarker:2 ∼ S1 ∼ UKPOL ∼ NURBsplineJoints:degree:knots ,
   polyline ∼ AA:Rationalize,
   polymarker:1 ∼ AA:Rationalize
]): points;
```