Some Mathematical Elements of Graphics (lecture notes for Advanced Graphics)

Neil Dodgson*

University of Cambridge Computer Laboratory

Overview

The course is in two halves. The first taught by Dr Alex Benton, the second by Prof. Neil Dodgson. These notes are for Prof. Dodgson's part of the course.

Lecture handouts and supervision material Some of the lecture course material is available on the web. This material is also printed out to provide these lecture notes.

Book list and their abbreviations The following books were used in preparation of these notes. Each is preceded by the abbreviation used in these notes to refer to that book. The B-spline part of the course is based on material from **R&A** and **P&M**.

- **S&M** Shirley, P. & Marschner, S. (2009).
  Fundamentals of Computer Graphics.
  A K Peters Ltd (3rd ed.).
  A good, up-to-date, general computer graphics text book. Chapter 15 covers curves (Bézier, B-spline, and NURBS). Chapter 17 covers computer animation in more detail than this course. For Dr Benton's part of the course, it also covers ray tracing (Ch. 4, 13), and implicit modelling (Ch. 16).

- **R&A** Rogers, D.F. & Adams, J.A. (1990).
  Mathematical Elements for Computer Graphics.
  McGraw-Hill (2nd ed.).
  A good coverage of the mathematics of the 2D and 3D representation of shape as it was understood in the year of publication. Explains Bézier, B-spline, and NURBS curves and surfaces in great detail. Also covers conics and quadrics.

- **P&M** Patrikalakis, N. M. & Maekawa, T. (2002).
  Shape Interrogation for Computer Aided Design and Manufacturing.
  Springer.
  This book, available online, is one of the textbooks for the graduate course "Computational Geometry" at MIT. Much of the book goes well beyond what is required for

*Written 10/99, modifications made 09/00, 10/02, 09/04, 04/06, 03/07, 01/10, 03/12, 01/14. Thanks to Malcolm Sabin, Alex Benton, and Jiří Kosinka for useful advice.

this course, but Chapter 1 provides a good alternative explanation of the material on Bézier and B-spline curves and surfaces.

- **Buss** Buss, S.R. (2003).
  3-D Computer Graphics.
  Cambridge University Press.
  A book that has the best description of radiosity (Ch. XI) that I have ever read. It also contains chapters on Bézier curves (VII), B-Splines (VIII), ray tracing (IX and X) and animation (XII).

- **W&W** Warren, J. & Weimer, H. (2002).
  Subdivision Methods for Geometric Design.
  Morgan Kaufmann.
  The first book devoted entirely to subdivision methods.

- **P&R** Peters, J. & Reif, U. (2008).
  Subdivision Surfaces.
  Springer.
  The second book on subdivision. Much more concerned with mathematical proof than **W&W**.

- **Sabin** Sabin, M. (2010).
  Analysis and Design of Univariate Subdivision Schemes.
  Springer.
  Somewhere between **W&W** and **P&R** in the mathematical fluency needed to understand it.

Note on copyright material I have included, in this handout, two extracts from textbooks. The extract from **Buss** is his excellent chapter on radiosity. The extract from Rogers and Adams (**R&A**) comprises *parts* of sections 5–8 (Bézier curves), 5–9 (B-splines), and 5–13 (NURBS).

These extracts are provided under the University of Cambridge's license from the Copyright Licensing Agency. This allows us to make one copy for each student and supervisor ("tutor") on the course *within certain limits*. These are: no more than three works and no more than 5% or one whole article or chapter from each work. This material is provided solely for the student's own study. Further copying of this handout is a breach of copyright.

Be warned: to fit inside the limits I have had to heavily edit the extracts from **R&A**. In particular, I have included none of the worked examples. To thoroughly understand the material I suggest that you read this extract and then borrow a copy of **R&A** in order to go through the examples. There should be a copy in your College's library.

1 The polygon

Today, almost all computer graphics rendering is done via the graphics processing unit (GPU) on a dedicated graphics card. These processors are optimised for drawing triangles. Therefore almost all computer graphics comprises drawing a large number of small

triangles very quickly. Modern graphics cards are capable of a wide range of operations on both vertices and pixels and therefore an enormous range of visual effects can be achieved that are far more than just "filling a triangle."

A consequence of this is that anything that must be drawn has to be converted into a mesh of triangles. However, it would be enormously inconvenient to a designer to have to edit a triangle mesh directly. Owing to the huge number of triangles involved in any good model, it is practically impossible to edit a triangle mesh by individually moving the vertices of the triangles. Therefore some higher-level representation is needed.

We will spend six lectures considering the two principal mechanisms by which we represent arbitrary curved surfaces for computer graphics: splines and subdivision. These mechanisms have been developed to make it easy for a designer to achieve the shape she wants. That is, the mechanisms allow the designer to affect the shape at the appropriate level of resolution: not so fine that a ridiculous number of manipulations are required (as would be the case with operating directly on the triangle mesh) and not so coarse that the designer cannot get the shape she wants.

2 Introduction to splines

While polygons are good for rendering, a designer cannot be expected to manipulate, directly, the millions of polygons that comprise the rendered model. We need some better way of generating curved surfaces. We need a general way of specifying arbitrary curved surfaces, which can then be converted to polygons for rendering. Ideally, we want a mechanism which allows us to specify any smooth curved surface which we desire. This problem was first faced in the 1960s for the design of aeroplanes and cars. We will look at two solutions: (1) B-spline curves and surfaces (including NURBS) and (2) subdivision curves and surfaces. Both methods were invented in the 1970s and they are today the two industry standard representations. The Computer-Aided Design (CAD) industry uses NURBS surfaces as its standard definition mechanism. The visual effects and animation industry uses both NURBS surfaces and subdivision surfaces.

A cubic B-spline is shown in Figure 1. The curve is generated from a sequence of *control points* and a mathematical function (Equation 1). The sequence of points can be thought of as the vertices of a *polygon* or *polyline*. The curve can be thought of as a refinement of that polyline. The designer is able to move the control points, and this changes the shape of the curve.

The basic formula defining any such curve is:

$$\mathbf{P}(t) = \sum_{i=1}^{n} N_i(t)\mathbf{P}_i \tag{1}$$

for n control points, $\mathbf{P}_i$. The $N_i(t)$ are called *basis functions*.

A fundamental property of any valid set of basis functions is that they *partition unity*. That is:

$$\sum_{i=1}^{n} N_i(t) = 1, \forall t. \tag{2}$$

This is because Equation 1 must be invarient under translation of all of the control points. So, if we move all of the control points a fixed distance, $\mathbf{P}'_i = \mathbf{P}_i + \mathbf{D}$, then we

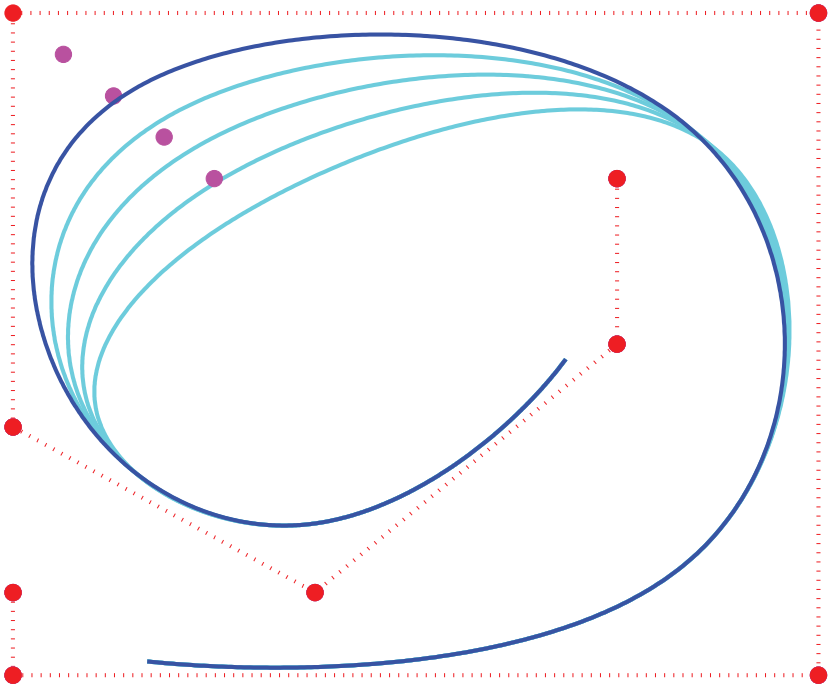


Figure 1: The dark blue curve is a uniform cubic B-spline curve. It is generated from the control points, shown as red circles, taken in the order indicated by the dotted red line. The purple points show different positions of one of the control points. The pale blue curves show how moving that control point changes the curve.

want the resulting curve to move by the same fixed distance, $\mathbf{P}'(t) = \mathbf{P}(t) + \mathbf{D}$.

$$\begin{aligned}
\mathbf{P}'(t) &= \sum_{i=1}^{n} N_i(t)(\mathbf{P}_i + \mathbf{D}) \\
&= \sum_{i=1}^{n} N_i(t)\mathbf{P}_i + \sum_{i=1}^{n} N_i(t)\mathbf{D} \\
&= \sum_{i=1}^{n} N_i(t)\mathbf{P}_i + \mathbf{D}, \text{ if } \sum_{i=1}^{n} N_i(t) = 1, \forall t
\end{aligned}$$

3 Bézier curves and surfaces

Bézier curves and surfaces were covered in the Part IB *Computer Graphics and Image Processing* course.

Béziers are a particular type of spline. They were developed in the 1960s for use in designing the bodies of cars. They are one of the earliest forms of spline implemented on computer for geometric design. Most textbooks (including **R&A** and **P&M**) introduce Béziers before B-splines and NURBS, because they provide a good introduction to the mathematics and the concepts involved.

This section gives some of the mathematical details, as does **R&A** Section 5-8. An extract from this Section of **R&A** is included in the handout. Please read that extract before continuing.

If you want to experiment with Bézier curves then there are a number of on-line tutorials. One such is available from the Technion.

A Bézier curve is a weighted sum of $n+1$ control points, $\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n$, where the weights are the Bernstein polynomials:

$$\mathbf{P}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i, 0 \leq t \leq 1 \tag{3}$$

The Bézier curve of order $n+1$ (degree n) has $n+1$ control points. Below are the first three orders of Bézier curve definitions.

$$\text{linear} \quad \mathbf{P}(t) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \tag{4}$$

$$\text{quadratic} \quad \mathbf{P}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2 \tag{5}$$

$$\text{cubic} \quad \mathbf{P}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \tag{6}$$

3.1 Ways of thinking about Bézier curves

There are several useful ways in which you can think about Bézier curves. Here are the ones that I use.

Linear interpolation. Equation 4 is obviously a linear interpolation between two points. Equation 5 can be rewritten as a linear interpolation between linear interpolations between points:

$$\mathbf{P}(t) = (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2] \tag{7}$$

Equation 6 can be rewritten as a linear interpolation between linear interpolations between linear interpolations between points. This is left as an exercise for the reader.

Weighted average. A Bézier curve can be seen as a weighted average of all of its control points. Because all of the weights are positive, and because the weights sum to one, the Bézier curve is guaranteed to lie within the convex hull of its control points.

Refinement of the control polygon. A Bézier curve can be seen as some sort of refinement of the polygon made by connecting its control points in order. The Bézier curve starts and ends at the two end points and its shape is determined by the relative positions of the $n-1$ other control points, although it will generally not pass through any of these other control points. The tangent vectors at the start and end of the curve pass through the end point and the immediately adjacent point.

R&A list the properties of the Bézier curve on page 291.

3.2 Continuity

One of the most important problems of using Bézier curves (and surfaces) is getting different pieces of curve (or patches of surface) to connect smoothly together, that is: with continuity of position ($C0$), slope ($C1$) and curvature ($C2$). These are continuity of the function, its first and its second derivatives, respectively. Much of the ensuing discussions consider how to achieve such continuity.

You should note that each Bézier curve is independent of any other Bézier curve. If we wish two Bézier curves to join with any type of continuity, then we must explicitly position the control points of the second curve so that they bear the appropriate relationship with the control points in the first curve.

Any Bézier curve is infinitely differentiable within itself, and is therefore continuous to any degree (C^n-continuous, $\forall n$). We therefore only need concern ourselves with continuity across the joins between curves. Assume that we have two Bézier curves of the same order: $\mathbf{P}(t)$, defined by $(\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n)$, and $\mathbf{Q}(t)$, defined by $(\mathbf{Q}_0, \mathbf{Q}_1, \ldots, \mathbf{Q}_n)$. C^0-continuity (continuity of position) can be achieved by setting $\mathbf{P}(1) = \mathbf{Q}(0)$. This gives a formula for $\mathbf{Q}_0$ in terms of the $\mathbf{P}_i$s:

$$\mathbf{Q}_0 = \mathbf{P}_n. \tag{8}$$

Similarly for C^1-continuity, we need C^0-continuity and $\mathbf{P}'(1) = \mathbf{Q}'(0)$, giving:

$$\mathbf{Q}_1 - \mathbf{Q}_0 = \mathbf{P}_n - \mathbf{P}_{n-1} \tag{9}$$

Combining Equations 9 and 8 gives a formula for $\mathbf{Q}_1$ in terms of the $\mathbf{P}_i$s:

$$\begin{aligned} \mathbf{Q}_1 &= 2\mathbf{P}_n - \mathbf{P}_{n-1} && (10)\\ &= \mathbf{P}_n + (\mathbf{P}_n - \mathbf{P}_{n-1}) && (11) \end{aligned}$$

Continuing in this vein, we find that the requirements for C^2-continuity (i.e. C^1-continuity and $\mathbf{P}''(1) = \mathbf{Q}''(0)$) give:

$$\mathbf{Q}_2 - 2\mathbf{Q}_1 + \mathbf{Q}_0 = \mathbf{P}_n - 2\mathbf{P}_{n-1} + \mathbf{P}_{n-2} \tag{12}$$

Combining Equations 12, 9, and 8 gives a formula for $\mathbf{Q}_2$ in terms of the $\mathbf{P}_i$s:

$$\begin{aligned} \mathbf{Q}_2 &= 4\mathbf{P}_n - 4\mathbf{P}_{n-1} + \mathbf{P}_{n-2} && (13)\\ &= \mathbf{P}_{n-2} + 4(\mathbf{P}_n - \mathbf{P}_{n-1}) && (14) \end{aligned}$$

3.3 Bézier surfaces

We learnt in the IB course that a Bézier surface is constructed by taking something known as the *tensor product* of Bézier curves. A tensor product Bézier surface of order $n+1$ is defined by $(n+1)^2$ control points. It is called a Bézier patch.

$$\mathbf{P}(s,t) = \sum_{i=0}^{n} \binom{n}{i} (1-s)^{n-i} s^i \sum_{j=0}^{n} \binom{n}{j} (1-t)^{n-j} t^j \mathbf{P}_{i,j} \tag{15}$$

You can think about this as moving the control points of one Bézier curve along a set of Bézier curves to sweep out a surface. Continuity across a boundary between two Bézier patches is only guaranteed if each of the Bézier curves across the join obey the curve continuity conditions. Again, this was covered in the IB course.

3.4 Exercises

1. Explain what $C0$-, $C1$-, $C2$-, Cn-continuity mean.
2. Equations (9) and (12) give the constraints on control point positions which ensure that Bézier curves join with $C1$- and $C2$-continuity. Derive these equations for two quartic Bézier curves by using the fact that the end-points must be identical, $\mathbf{Q}_0 = \mathbf{P}_n$, and then setting $\mathbf{P}'(1) = \mathbf{Q}'(0)$ and $\mathbf{P}''(1) = \mathbf{Q}''(0)$.

4 B-splines

B-spline curves are the general case of a particular type of curve that was originally derived to meet certain mathematical properties. In particular, they maximise the *continuity* of the curve while minimising the *support*. The *support* is the length of curve that is modified by moving a single control point. The importance of minimising support is that is means any change made by a designer is *local*. For example, if a designer is changing the shape of the nose of an animated character, she wants that change to be local to the control point she is moving. She does not want that change to affect, for example, the feet of the character. **R&A** section 5–9 (extract included in the handout) discusses B-splines curves.

In a B-spline curve each control point is associated with a *basis function*, $N_{i,k}(t)$.

$$\mathbf{P}(t) = \sum_{i=1}^{n} N_{i,k}(t)\mathbf{P}_i,\ t_{\min} \leq t < t_{\max} \tag{16}$$

There are n control points[1], $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_n$. The $N_{i,k}$ basis functions are of *order* k. They are *piecewise polynomial* functions. That is, they are made up of pieces that are each polynomial but that each piece is a different polynomial to the pieces on either side. Figure 2 shows an example.

The *degree* of the polynomial pieces is one less than the order, so they are of degree $k-1$. k must be at least 2 (linear), and can be no more than n (the number of control points). The order of the curve (2 [linear], 3 [quadratic], 4 [cubic],...) is not dependent on the number of control points. This is one important way in which B-splines differ from Béziers and means that we do not need to worry about the continuity between pieces of the same B-spline curve, because the continuity is guaranteed automatically (see details later).

Equation 16 defines a piecewise polynomial function. The $N_{i,k}$ are defined by a *knot vector*, $[t_1, t_2, \ldots, t_{k+n}]$, which we consider in detail below. The knot vector specifies the values of the parameter t at which the pieces of curve join, by analogy to knots joining bits of string. It is necessary that the t_i form a non-decreasing sequence of real numbers:

$$t_i \leq t_{i+1}, \forall i \tag{17}$$

The $N_{i,k}$ depend *only* on the value of k and the values, t_i. $N_{i,k}$ is defined recursively as:

$$N_{i,1}(t) = \begin{cases} 1, & t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

[1]Note that this differs by one from **R&A**, who use $n+1$ control points labelled from 1 to $n+1$.

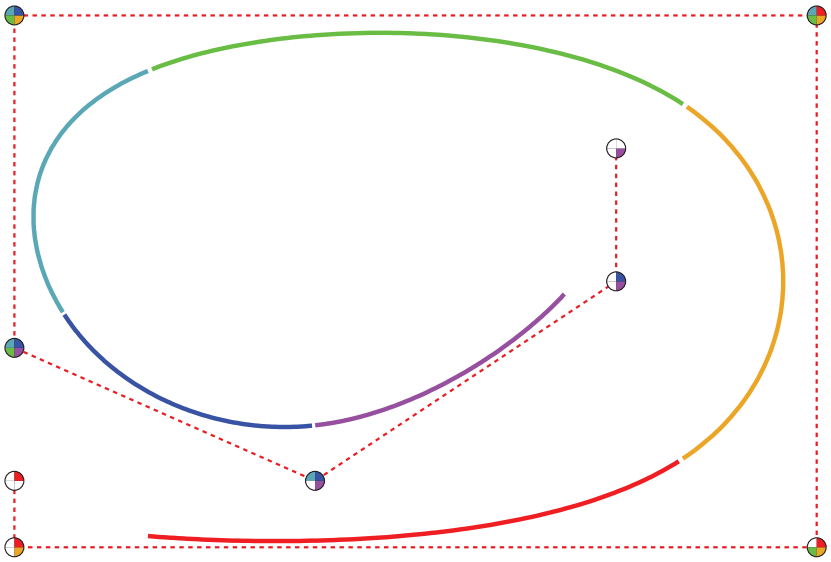


Figure 2: The same example as Figure 1 showing the piecewise nature of the curve. The curve is a uniform cubic B-spline curve generated from the control points. Each piece of the curve is shown in a different colour. Each piece of the curve is a cubic. Each piece of the curve is generated from four control points. The colours within each control point show which pieces of the curve are affected by moving that control point. You can see that each control point affects up to four pieces of curve. The points at which we transition from one curve to another are the knot points. At such a point, the position of the curve is controlled by only those control points common to the pieces either side. Therefore, in this case (cubic), the knot point's position depends on only three control points.

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) \tag{19}$$

When computing this recursive function it is possible for the denominator to be zero. In these cases the numerator is also always zero and we use the convention that $0/0 = 0$. This can be justified formally by considering limits as the spacing between knots approaches zero.

Figure 3 shows the generation of functions, up to the cubic ($n = 4$) from the knot vector $[1, 2, 3, 4, 5]$.

Let us work through the mathematical derivation of some of these functions. Consider the first two functions, $N_{1,1}$ and $N_{2,1}$. These are generated directly from the base case:

$$N_{1,1}(t) = \begin{cases} 1, & 1 \leq t < 2 \\ 0, & \text{otherwise} \end{cases}$$

$$N_{2,1}(t) = \begin{cases} 1, & 2 \leq t < 3 \\ 0, & \text{otherwise} \end{cases}$$

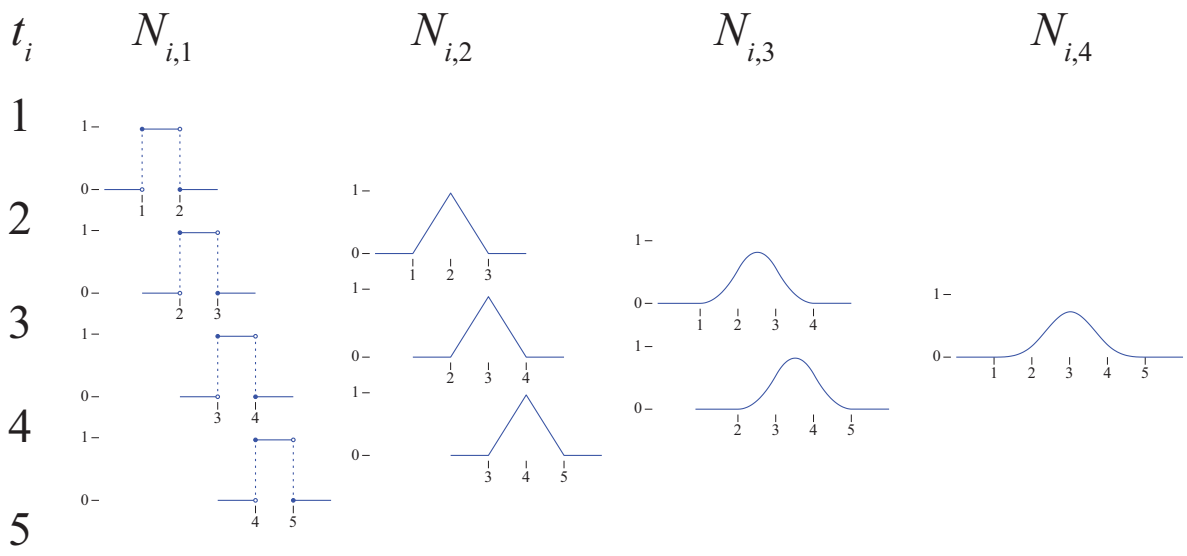


Figure 3: The knot vector on left, $[1, 2, 3, 4, 5]$, generates the $N_{i,1}$ basis functions using the base case of the recursive definition. The higher order functions are generated by the recursive part of the definition. Notice that, for these uniformly-spaced knots, each basis function for a given order is simply a shifted version of all the others of that same order.

The next function up, $N_{1,2}$ is generated from these two by the recursive case:

$$\begin{aligned} N_{1,2}(t) &= \frac{t-t_1}{t_2-t_1}N_{1,1}(t) + \frac{t_3-t}{t_3-t_2}N_{2,1}(t) \\ &= \frac{t-1}{2-1}N_{1,1}(t) + \frac{3-t}{3-2}N_{2,1}(t) \\ &= (t-1)N_{1,1}(t) + (3-t)N_{2,1}(t) \end{aligned}$$

Notice two things: first, the multiplier in front of each function is simply a straight line that varies between 0 and 1 over the range for which the function is non-zero. Figure 4 shows these two linear functions. For $N_{1,1}$, the multiplier increases linearly from 0 to 1 over the non-zero range of $N_{1,1}$. For $N_{2,1}$, the multiplier decreases linearly from 1 to 0 over the non-zero range of $N_{2,1}$. This pattern for the multiplication factors is the same at every level and for every function.

Second, notice that $N_{1,1}$ and $N_{2,1}$ are piecewise functions, therefore the definition of $N_{1,2}$ is also going to be piecewise. Its pieces are drawn from the ranges of the two simpler functions. There will be three separate sections to the function: a piece from $t = 1$ to $t = 2$, a piece from $t = 2$ to $t = 3$, and something to handle the "otherwise" case.

$$N_{1,2}(t) = \begin{cases} t-1, & 1 \le t < 2 \\ 3-t, & 2 \le t < 3 \\ 0, & \text{otherwise} \end{cases}$$

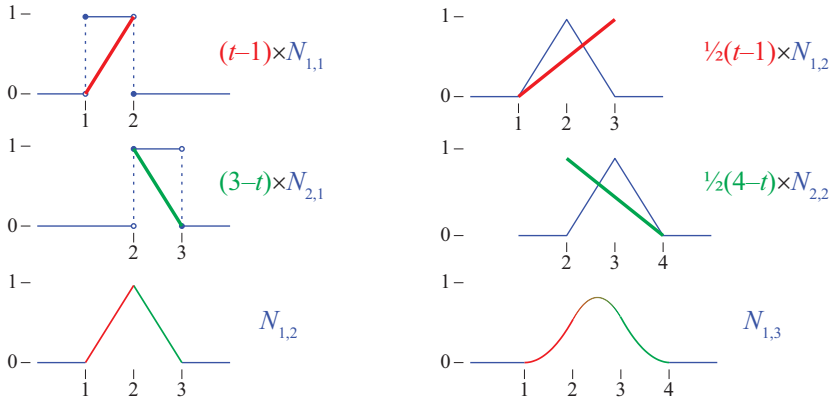


Figure 4: Two examples of the recursive step in Equation 19. In each case the two functions of order $k-1$ are combined to produce one function of order k. The combining factors are simple linear functions that rise from 0 to 1 or fall from 1 to 0 over the range for which the $N_{i,k-1}$ is non-zero.

$N_{2,2}$ is generated by a similar process and, as you can see, $N_{2,2}(t) = N_{1,2}(t-1)$:

$$N_{2,2}(t) = \begin{cases} t-2, & 2 \le t < 3 \\ 4-t, & 3 \le t < 4 \\ 0, & \text{otherwise} \end{cases}$$

We now turn to the case of generating a $N_{1,3}$: a piecewise quadratic function. That is, it will be comprised of several pieces, each of which will be a quadratic. $N_{1,3}$ is generated from the two piecewise linear functions above:

$$\begin{aligned} N_{1,3}(t) &= \frac{t-t_1}{t_3-t_1}N_{1,2}(t) + \frac{t_4-t}{t_4-t_2}N_{2,2}(t) \\ &= \frac{t-1}{3-1}N_{1,2}(t) + \frac{4-t}{4-2}N_{2,2}(t) \\ &= \frac{t-1}{2}N_{1,2}(t) + \frac{4-t}{2}N_{2,2}(t) \end{aligned}$$

Again, notice two things. First, Figure 4 confirms that the multipliers on the two functions are simply linear ramps ranging between 0 and 1 for the range over which the function is non-zero.

Second, this is a piecewise function. It needs to have pieces that match each piece of the two functions that combine to create it. Of course, those functions are generated from one knot vector, so the places at which the pieces join will be knot values. In this case, the function will be of this form:

$$N_{1,3}(t) = \begin{cases} ?, & 1 \le t < 2 \\ ?, & 2 \le t < 3 \\ ?, & 3 \le t < 4 \\ 0, & \text{otherwise} \end{cases}$$

with three quadratic functions to derive, one for each of the three question marks.

Using the appropriate ranges from $N_{2,1}$ and $N_{2,2}$ we get this:

$$N_{1,3}(t) = \begin{cases} \frac{t-1}{2}(t-1) & + \frac{4-t}{2} \times 0, & 1 \le t < 2 \\ \frac{t-1}{2}(3-t) & + \frac{4-t}{2}(t-2), & 2 \le t < 3 \\ \frac{t-1}{2} \times 0 & + \frac{4-t}{2}(4-t), & 3 \le t < 4 \\ 0, & \text{otherwise} & \end{cases}$$

$$= \begin{cases} \frac{(t-1)^2}{2} & + 0, & 1 \le t < 2 \\ \frac{(t-1)(3-t)}{2} & + \frac{(4-t)(t-2)}{2}, & 2 \le t < 3 \\ 0 & + \frac{(4-t)^2}{2}, & 3 \le t < 4 \\ 0, & \text{otherwise} & \end{cases}$$

$$= \begin{cases} \frac{1}{2}(t-1)^2, & 1 \le t < 2 \\ \frac{1}{2}(t-1)(3-t) + \frac{1}{2}(4-t)(t-2), & 2 \le t < 3 \\ \frac{1}{2}(4-t)^2, & 3 \le t < 4 \\ 0, & \text{otherwise} \end{cases}$$

At this point it would be instructive for you to work out $N_{1,1}$, $N_{2,1}$, $N_{3,1}$, $N_{1,2}$, $N_{2,2}$, $N_{1,3}$ for the knot vector $[0, 2, 3, 6]$. It helps if you draw the graphs for these functions.

4.1 Features of B-splines

Earlier, I mentioned that B-splines had originally been derived mathematically as the set of basis functions that minimise support while maximising continuity. These are both important properties of B-splines. There are several properties that we should note about these equations:

1. Each $N_{i,k}(t)$ depends only on the $k+1$ knot values from t_i to t_{i+k}. In particular, remember that $N_{i,k}(t)$ does not depend on the positions of the control points.

2. $N_{i,k}(t) = 0$ for $t < t_i$ or $t \ge t_{i+k}$ so $\mathbf{P}_i$ only influences the curve for $t_i \le t < t_{i+k}$.

3. $N_{i,k}(t)$ is a polynomial of order k (degree $k-1$) on each interval $t_i < t < t_{i+1}$.

4. $\mathbf{P}(t)$ is continuous in all its derivatives *between* the knots. This property follows from the fact that each of the $N_{i,k}(t)$ is a polynomial between each pair of knots because a polynomial can be differentiated infinitely many times.

5. Across the knots $\mathbf{P}(t)$ is C^{k-2}-continuous. This is because the $N_{i,k}(t)$ are continuous in $k-2$ derivatives across the knots but they have discontinuities in the $k-1$ derivative. Notice that the base case (Equation 18, $k = 1$) has discontinuities and therefore is not even $C0$. You can consider each step of recursion (Equation 19) to add one extra level of continuity to the function, so $k = 2$ is $C0$, with discontinuities in the first derivative; $k = 3$ is $C1$, with discontinuities in the second derivative; and so on.

6. Equation 16 shows that $\mathbf{P}(t)$ is only valid for $t_{\min} \leq t < t_{\max}$. The values of these are: $t_{\min} = t_k$ and $t_{\max} = t_{n+1}$. This is because a weighted sum of points only makes sense if the weights sum to one. It is therefore validly defined only where

$$\sum_{i=1}^{n} N_{i,k}(t) = 1.$$

It thus starts at t_k, the lowest knot value for which there are k basis functions that are not in their "otherwise" condition and ends at t_{n+1} the equivalent knot value at the other end of the knot vector (i.e., t_{n+1} is k knots away from the end of the vector, which is at t_{n+k}).

4.2 The knot vector

The above should have led you to realise that the knot vector is important. The knot vector can be any sequence of numbers provided that each one is greater than or equal to the preceding one. There is a limit on how many knots can be coincident (i.e., have equal values). The *multiplicity* of a knot is limited to the degree of the curve; since a higher multiplicity would split the curve into disjoint parts and it would leave control points unused.

The shapes of the $N_{i,k}$ basis functions are determined entirely by the *relative* spacing between the knots. Scaling ($t'_i = \alpha t_i, \forall i$) or translating ($t'_i = t_i + \Delta t, \forall i$) the knot vector has no effect on the shapes of the $N_{i,k}$ nor on the shape of the actual curve $\mathbf{P}(t)$.

Some types of knot vector are more useful than others. Knot vectors are generally placed into one of three categories: uniform, open uniform, and non-uniform.

Uniform. These are knot vectors for which

$$t_{i+1} - t_i = \text{constant}, \forall i \tag{20}$$

For example:

$$[1, 2, 3, 4, 5, 6, 7, 8]$$
$$[0, 1, 2, 3, 4, 5]$$
$$[0, 0.25, 0.5, 0.75, 1.0]$$
$$[-2.5, -1.4, -0.3, 0.8, 1.9, 3.0]$$

A basis function's shape depends only on the *relative* spacing of the knots involved. Because all the knots are equally spaced, all of the basis functions, $N_{i,k}(t)$ for a given k, are just shifted versions of one another, as can be seen in Figure 5.

Open Uniform. These are uniform knot vectors which have k equal knot values at each end:

$$\begin{array}{rl} t_i = t_k, & i \leq k \\ t_{i+1} - t_i = \text{constant}, & k \leq i < n+1 \\ t_i = t_{n+1}, & n+1 \leq i \end{array} \tag{21}$$

For example:

$$[0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4] \quad (k = 4)$$
$$[1, 1, 1, 2, 3, 4, 5, 6, 6, 6] \quad (k = 3)$$
$$[0.1, 0.1, 0.1, 0.1, 0.1, 0.3, 0.5, 0.7, 0.7, 0.7, 0.7, 0.7] \quad (k = 5)$$

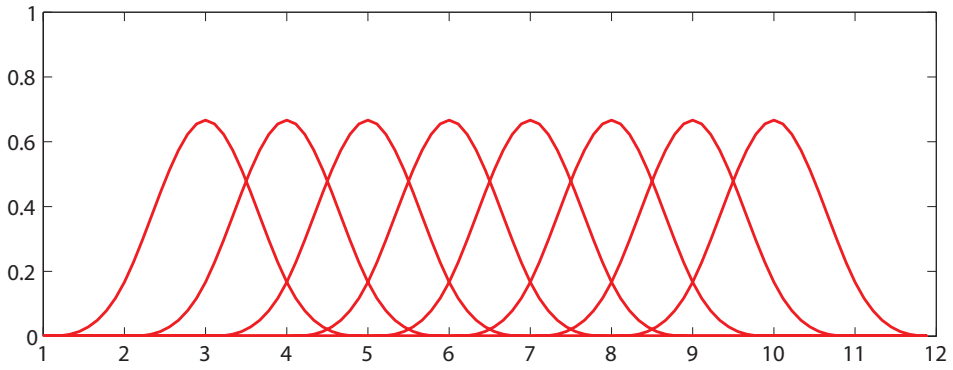


Figure 5: An example of a set of uniform basis functions. These cubic basis functions are generated from the knot vector $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$. They are simply shifted versions of one another: $N_{b,4}(t) = N_{a,4}(t - b + a)$.

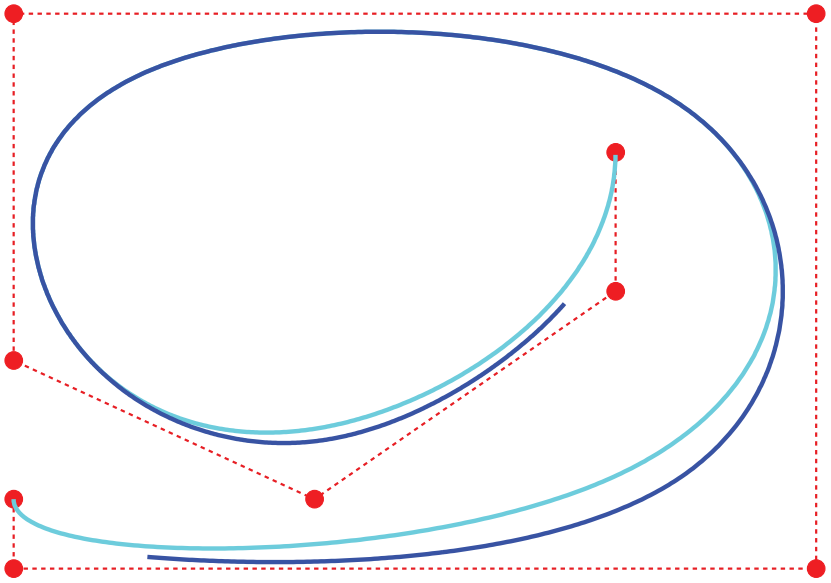


Figure 6: An example of the practical difference between a *uniform* and an *open uniform* knot vector. The curve (light blue) generated by the open uniform vector passes through the two end points, while the curve (dark blue) generated by the uniform knot vector does not. In this cubic case, the difference between the two curves is limited to the two pieces of curve at each end. All of the other pieces of curve are identical.

This is a simple modification to the uniform case that allows the curve to go through its two end points. This arrangement of knots at the ends is known as "Bézier end-conditions." Figure 6 shows an example of the practical difference between uniform and open uniform knot vectors. Figure 7 shows the similarities and differences between the basis functions.

A Bézier curve of order n is a special case of an open uniform B-spline curve with

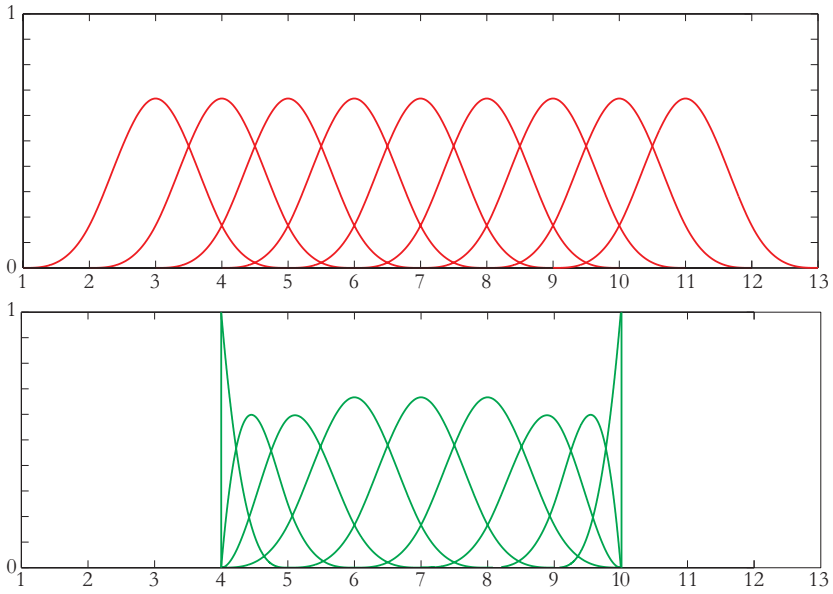


Figure 7: The basis functions for a *uniform* (red) and an *open uniform* (green) knot vector. Both are cubic with order $k = 4$ and a for $n = 9$ control points. The knot vector for the uniform case is $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]$. The knot vector for the open uniform case is $[4, 4, 4, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]$. Both define valid curves between knots $t_4 = 4$ and $t_{10} = 10$. Notice that, in this cubic case, only the three basis functions at each end are different. The basis functions in the middle are identical.

the knot vector comprising n zeros followed by n ones. For example, the cubic Bézier has the knot vector $[0, 0, 0, 0, 1, 1, 1, 1]$.

Non-uniform. This is the general case, the only constraint being the standard $t_i \leq t_{i+1}, \forall i$ (Equation 17). For example:

$$[1, 3, 7, 22, 23, 23, 49, 50, 50]$$
$$[1, 1, 1, 2, 2, 3, 4, 5, 6, 6, 6, 7, 7, 7]$$
$$[0.2, 0.7, 0.7, 0.7, 1.2, 1.2, 2.9, 3.6]$$

The above gives a description of the various types of knot vector but it does not give you any insight into how the knot vector determines the shape of the curve. The following subsections look at the different types of knot vector in more detail. However, the best way to get to feel for these is to derive and draw the basis functions yourself.

4.2.1 Uniform knot vector

For simplicity, let $t_i = i$ (this is allowable given that the scaling or translating the knot vector has no effect on the shapes of the $N_{i,k}$). The knot vector thus becomes $[1, 2, 3, \ldots, k + n]$ and Equation 19 simplifies to:

$$N_{i,1}(t) = \begin{cases} 1, & i \leq t < i + 1 \\ 0, & \text{otherwise} \end{cases}$$

$$N_{i,k}(t) = \frac{t-i}{k-1}N_{i,k-1}(t) + \frac{i+k-t}{k-1}N_{i+1,k-1}(t) \quad (22)$$

You should be easily able to graph the first few of these for yourself. The principle thing to note about the uniform basis functions is that, for a given order k, the basis functions are all simply shifted versions of one another. See Figure 7(top) and **R&A** Figure 5-36.

With a uniform B-spline, you obviously cannot change the basis functions (they are fixed because all the knots are equispaced). However you can alter the shape of the curve by modifying a number of other things:

Moving control points. Moving the control points obviously changes the shape of the curve. This is the usual way in which a designer will change the shape of a B-spline curve.

Multiple control points. Sticking two adjacent control points on top of one another causes the curve to pass closer to that point. Stick enough adjacent control points on top of one another and you can make the curve pass through that point (**R&A**, Figure 5-45).

Order. Increasing the order k increases the continuity of the curve at the knots, increases the smoothness of the curve, and tends to move the curve farther from its defining polygon. (**R&A**, Figure 5-44).

Joining the ends. You can join the ends of the curve to make a closed loop. Say you have M points, $\mathbf{P}_1, \ldots, \mathbf{P}_M$. You want a closed B-spline defined by these points. For a given order, k, you will need $M + (k-1)$ control points (repeating the first $k-1$ points): $\mathbf{P}_1, \ldots, \mathbf{P}_M, \mathbf{P}_1, \ldots, \mathbf{P}_{k-1}$. Your knot vector will thus have $M + 2k - 1$ uniformly spaced knots. An example is shown in Figure 8.

4.2.2 Open uniform knot vector

The final paragraph in the previous section tells you that uniform B-splines can be used to describe closed curves: all you have to do is join the ends as described above. If you do not want a closed curve, and you use a uniform knot vector, you find that you need to specify control points at each end of the curve which the curve does not go near (e.g. Figure 6, the dark blue curve, and **R&A**, Figure 5-44, the order 4 curve).

If you wish your B-spline to start and end at your first and last control points then you need an open uniform knot vector (e.g. Figure 6, the light blue curve, and **R&A**, Figure 5-41). The only difference between this and the uniform knot vector being that the open uniform version has k equal knots at each end.

As mentioned earlier, an order k open uniform B-spline with $n = k$ points is the Bézier curve of order k. It would be a useful exercise for you to prove this for $k = 3$. For ease of calculation take the knot vector to be $[0, 0, 0, 1, 1, 1]$.

4.2.3 The difference between uniform and open uniform

It may help, at this stage, to compare a particular uniform and an equivalent open uniform knot vector. This is a uniform knot vector for $n = 7$, $k = 3$:

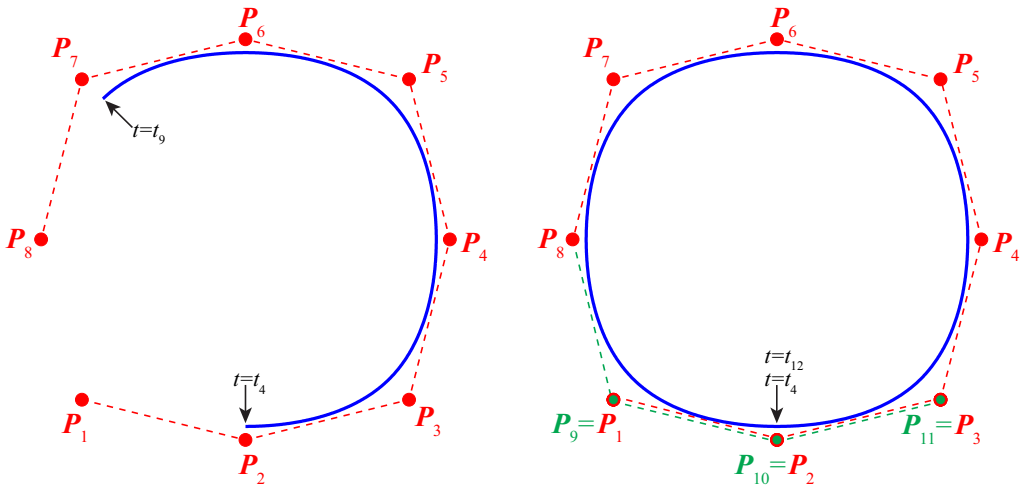
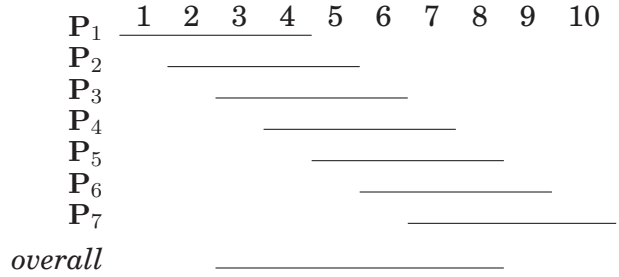
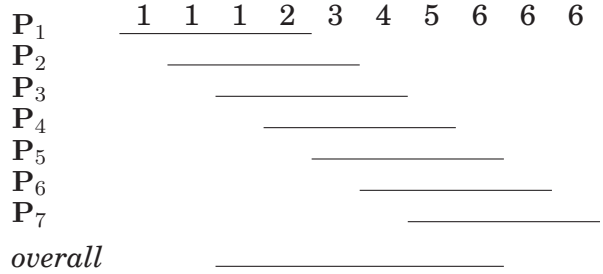


Figure 8: Two curves defined by a uniform knot vector. At left is the curve defined by eight points. At right is the equivalent *closed* curve. In order to get a closed curve, we duplicate the first $k-1$ control points at the end of the list of control points. In this cubic case, $k = 4$, so the first three points are copied to the end of the list, producing a control polygon with eleven points and a curve that is closed because its end, at $t = t_{n+1} = t_{12}$, perfectly meets its start, at $t = t_k = t_4$.



The lines show the range of t over which each $N_{i,k}$ is non-zero. The B-spline itself (the *overall* line in the diagram) is defined over the range $t_3 \leq t < t_8$, i.e. over the range $3 \leq t < 8$.

By comparison an open uniform knot vector for $n = 7$, $k = 3$ is:



The B-spline itself is defined over the range $t_3 \leq t < t_8$, i.e. over the range $1 \leq t < 6$. By the definition of a open uniform knot vector $t_3 = t_1$ and $t_8 = t_{10}$ and so an open uniform

B-spline is defined over the *full* range of t from t_1 to t_{k+n}.

4.2.4 Non-uniform knot vector

Any B-spline whose knot vector is neither uniform nor open uniform is non-uniform. Non-uniform knot vectors allow any spacing of the knots, including multiple knots (adjacent knots with the same value). We need to know how this non-uniform spacing affects the basis functions in order to understand where non-uniform knot vectors could be useful. Owing to the knot vector being scale- and translation-invariant, there are only three cases to consider: (1) multiple knots (adjacent knots equal); (2) adjacent knots more closely spaced than the next knot in the vector; and (3) adjacent knots less closely spaced than the next knot in the vector. Obviously, case (3) is simply case (2) turned the other way round.

Multiple knots. A multiple knot reduces the degree of continuity at that knot value. Across a normal knot the continuity is C^{k-2}. Each extra knot with the same value reduces continuity at that value by one. This is the only way to reduce the continuity of the curve at the knot values. If there are $k-1$ (or more) equal knots then you get a discontinuity in the curve.

Close knots. As two knots' values get closer together, relative to the spacing of the other knots, the curve moves closer to the related control point.

Distant knots. As two knots' values get further apart, relative to the spacing of the other knots, the curve moves further away from the related control point.

Standard procedure is to use uniform or open uniform B-splines unless there is a good reason not to do so.

There are some who advocate using a form of non-uniform knot spacing where knots are spaced relative to the spacing of the associated control points. Figure 9 shows an example. Having said that, moving two knots closer together or further apart tends to change the curve only slightly and so there seems little advantage from having this non-uniform spacing.

Today, the main use of non-uniform B-splines seems to be to allow for multiple knots, which adjust the continuity of the curve at the knot values.

However, non-uniform B-splines are the general form of the B-spline, incorporating both open uniform and uniform B-splines as special cases. Thus we will talk about *non-uniform B-splines* when we mean the general case, incorporating both uniform and open uniform.

4.2.5 What can you do to control the shape of a B-spline?

- Move the control points.
- Add or remove control points.
- Use multiple control points.
- Change the order, k.

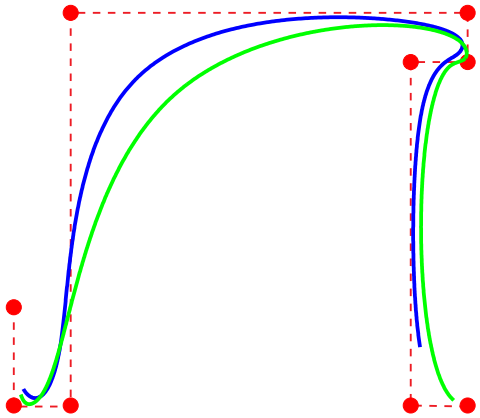


Figure 9: Two curves illustrating the difference between a uniform knot vector (blue) and a knot vector based on the relative spacing between control points (green). The principal differences to notice are in the region where the control points are far apart (top left), where the non-uniform version is farther from the control polygon, and the region where the control points are close together (top right) where the non-uniform version is closer to the control polygon. This is a cubic B-spline, so $k = 4$. The relative spacing between control points is $[2, 1, 8, 7, 1, 1, 7, 1]$, so the non-uniform knot vector used here is $[0, 0, 0, 2, 3, 11, 18, 19, 20, 27, 28, 28, 28]$. Note that there are $n - 1$ intervals between control points but $n + k - 1$ intervals between knots, therefore k knot values are not determined by the intervals between control points. For this example, I have simply duplicated the knots at each end of the knot vector until there are enough knot values.

- Change the type of knot vector.
- Change the relative spacing of the knots.
- Use multiple knot values in the knot vector.

4.2.6 What should the defaults be?

If there are no pressing reasons for doing otherwise, your B-spline should be defined by:

- $k = 4$ (cubic);
- no multiple control points;
- uniform (for a closed curve) or open uniform (for an open curve) knot vector.

4.3 Evaluating a B-spline

A B-spline curve can be evaluated at parameter value, t, using the *de Boor* algorithm. This is essentially a rewriting of Equation 16 using the recursion in Equation 19. For a value of t such that $t_j \leq t < t_{j+1}$:

$$\mathbf{P}(t) = \sum_{i=j-k+1}^{j} N_{i,k}(t)\mathbf{P}_i, \qquad t_j \leq t < t_{j+1} \tag{23}$$

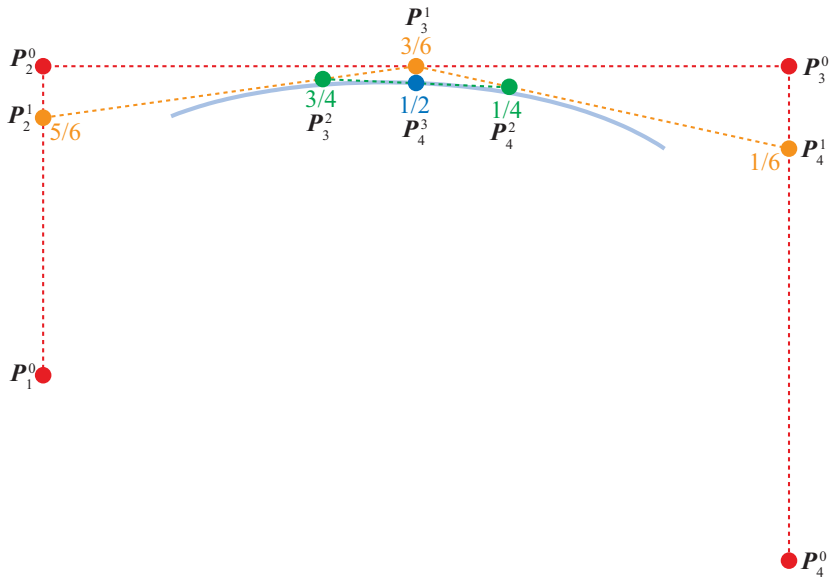


Figure 10: An example of the construction of a cubic uniform B-spline. This example shows the construction of the central point in the piece of curve. The knot vector for this piece of curve is $[1,2,3,4,5,6,7,8]$, the curve is defined in the range $t_4 \leq t < t_5$, the construction is shown for the point $t = 4\frac{1}{2}$. Keeping in mind that, in this example, $t_i = i$, we can see that the weights, α_i^m, are: $\alpha_2^1 = \frac{t-2}{5-2} = \frac{5}{6}$, $\alpha_3^1 = \frac{t-3}{6-3} = \frac{3}{6}$, $\alpha_4^1 = \frac{t-4}{7-4} = \frac{1}{6}$; $\alpha_3^2 = \frac{t-3}{5-3} = \frac{3}{4}$, $\alpha_4^2 = \frac{t-4}{6-4} = \frac{1}{4}$; $\alpha_4^3 = \frac{t-4}{5-4} = \frac{1}{2}$.

$$= \sum_{i=j-k+1+m}^{j} N_{i,k-m}(t)\mathbf{P}_i^m(t), \quad t_j \leq t < t_{j+1},\ 0 \leq m \leq j-1 \tag{24}$$

$$= \mathbf{P}_j^{k-1}(t), \quad t_j \leq t < t_{j+1} \tag{25}$$

where:

$$\mathbf{P}_i^m(t) = (1-\alpha_i^m)\mathbf{P}_{i-1}^{m-1}(t) + \alpha_i^m \mathbf{P}_i^{m-1}(t) \tag{26}$$

$$\mathbf{P}_i^0(t) = \mathbf{P}_i \,\forall t \tag{27}$$

$$\alpha_i^m = \frac{t-t_i}{t_{i+k-m}-t_i} \tag{28}$$

Figure 10 shows an example of this construction, where you should be able to see that the construction is simply combinations of linear blends of two points (Equation 26). Figure 11 provides an example comparing how the evaluation of a point on the curve relates to the construction of the basis functions that define the curve.

When implementing B-spline curve drawing, it is convenient to rescale the knot vector to the range [0,1], that is so that $t_{\min} = 0$ and $t_{\max} = 1$. This is claimed to improve numerical accuracy in floating point arithmetic computation owing to the higher desinty of floating point numbers in this interval.

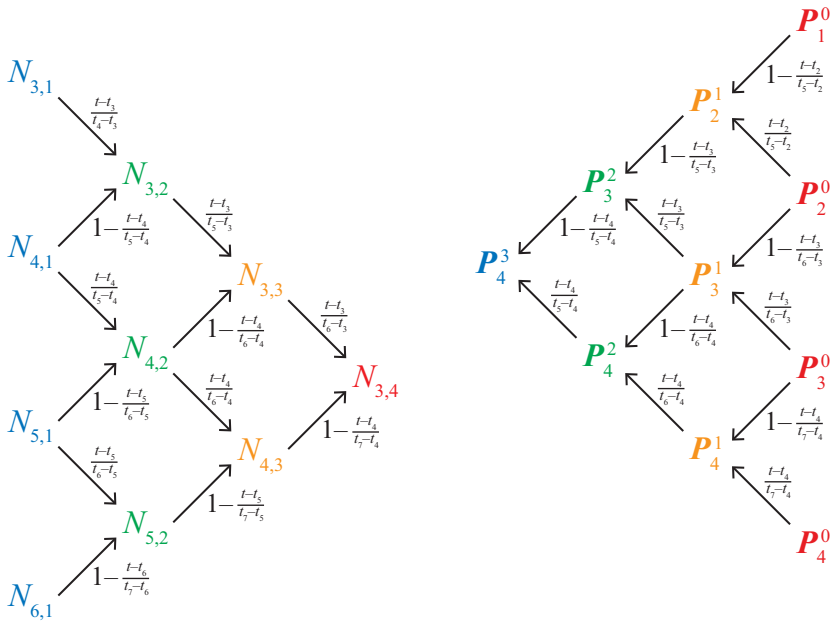


Figure 11: A comparision of the construction of basic functions (left, using Equation 19) and the evaluation of a point on the curve (right, using Equation 26). At left, we see the construction of basis function $N_{3,4}$ from the knots $[t_3, t_4, t_5, t_6, t_8]$. At right, we see the evaluation of a point $\mathbf{P}_4^3$, which is in the range $[t_4, t_5)$. The important thing to notice is that the weights on the arrows are identical for equivalent arrows in the two diagrams. For example, $\mathbf{P}_4^3$ receives a contribution of $\alpha_4^3 = \frac{t-t_4}{t_5-t_4}$ from $\mathbf{P}_4^2$, which is exactly the same contribution that $N_{4,2}$ receives from $N_{4,1}$.

4.4 Knot insertion

When a designer is editing a B-spline curve, she may wish to add extra detail in an area that has insufficient control points to represent the detail. It is therefore useful to be able to add extra control points. The initial insertion of a new point should not change the shape of the existing curve: it should simply provide one extra control that the designer can subsequently edit. Knot insertion allows us to provide this capabiity.

Knot insertion inserts a new knot into the knot vector, which causes a new control point to be inserted into the control polygon and also (except in the case $k = 2$) causes some of the existing control points to move.

A B-spline of order k over knot vector $[t_1, t_2, \ldots, t_j, t_{j+1}, \ldots, t_{k+n}]$ is (Equation 1):

$$\mathbf{P}(t) = \sum_{i=1}^{n} N_i(t)\mathbf{P}_i.$$

Let us now insert a new knot, $\hat{t}$, between t_j and t_{j+1}. That is: $t_j \leq \hat{t} \leq t_{j+1}$. This creates a new knot vector with $k + n + 1$ knots, $[t_1, t_2, \ldots, t_j, \hat{t}, t_{j+1}, \ldots, t_{k+n}]$, and a new B-spline curve:

$$\hat{\mathbf{P}}(t) = \sum_{i=1}^{n+1} \hat{N}_i(t)\hat{\mathbf{P}}_i.$$

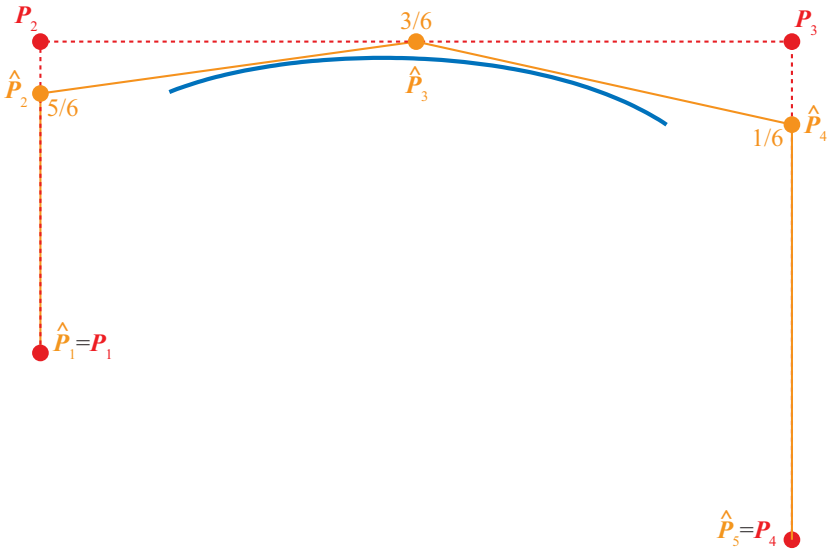


Figure 12: An example of the insertion of a new knot in a cubic uniform B-spline. This example shows the insertion of a new knot half-way along the curve. The knot vector for the original curve is $[1,2,3,4,5,6,7,8]$, the curve is defined in the range $t_4 \leq t < t_5$, and there are four control points. The new knot is inserted at $\hat{t} = 4\frac{1}{2}$, making the new knot vector $[1,2,3,4,4\frac{1}{2},5,6,7,8]$. There are five control points in the new control polygon; the fractions in the diagram show the weights used to generate the new points from the two old points either side. Both control polygons produce exactly the same curve. You should be able to see the relationship between knot insertion (here) and the recursive construction of points on the curve (Figure 10).

All we now need to do is work out the locations of the new points so that $\hat{\mathbf{P}}(t) = \mathbf{P}(t)$. This is straightforward, and the new points are blends of the old points in a way reminiscent of Equation 26:

$$\hat{\mathbf{P}}_i = \begin{cases} \mathbf{P}_i, & i \leq j-k+1 \\ \dfrac{t_{i+k-1}-\hat{t}}{t_{i+k-1}-t_i}\mathbf{P}_{i-1} + \dfrac{\hat{t}-t_i}{t_{i+k-1}-t_i}\mathbf{P}_i, & j-k+1 < i \leq j \\ \mathbf{P}_{i-1}, & i \geq j+1 \end{cases} \tag{29}$$

That is: most of the control points do not move, and $k-2$ existing points are replaced by $k-1$ points around the location of the inserted knot. Figure 12 shows an example of this construction, where you should be able to see that the new points are made as simple linear blends of two points (Equation 29).

4.5 B-spline patches

We generalise from B-spline curves to B-spline surfaces by taking what is called the "tensor product" of two versions of Equation 16.

$$\mathbf{P}(s,t) = \sum_{i=1}^{m}\sum_{j=1}^{n} \mathbf{P}_{i,j} N_{i,k}(s) N_{j,l}(t), s_{\min} \leq s < s_{\max}, t_{\min} \leq t < t_{\max} \tag{30}$$

where it is usual for the patch to have the same order (i.e. $k = l$) in both directions. Patches are thus defined by a quadrilateral grid of control points of size $m \times n$.

4.6 Why B-splines?

B-splines have many nice properties when compared to other families of curves which could be used. They:

- minimise the order of the polynomial pieces (order k)
- maximise the continuity between pieces (continuity $C(k-2)$)
- minimise the number of control points controlling a piece (k points)
- have positive basis functions
- have basis functions which partition unity, implying that each piece lies inside its control points' convex hull
- are invariant with respect to affine transforms

4.7 Exercises

1. How many control points are required for a quartic Bézier and how many for a quartic B-spline?
2. Why are cubics the default for B-spline use?
3. Explain the difference between Uniform, Open Uniform, and Non-Uniform knot vectors. What are the advantages of each type?
4. Work out $N_{1,1}$, $N_{2,1}$, $N_{3,1}$, $N_{1,2}$, $N_{2,2}$, $N_{1,3}$ for the knot vector $[0,2,3,6]$. Draw the graphs of these functions. [This is the exercise on page 11.]
5. [2000/9/4] (b) A B-spline has knot vector $[1,2,4,7,8,10,12]$. Derive the first of the third order (second degree) basis functions, $N_{1,3}(t)$, and graph it.
   If this knot vector were used to draw a third order B-spline, how many control points would be required? [7 marks]
6. [2001/8/4] (a) For a given order, k, there is only one basis function for uniform B-splines. Every control point uses a shifted version of that one basis function. How many different basis functions are there for open-uniform B-splines of order k with n control points, where $n >= 2k-2$? [6 marks]
   (b) Explain what is different in the cases where $n < 2k-2$ compared with the cases

where $n \geq 2k - 2$. [3 marks]
(c) Sketch the different basis functions for $k = 2$ and $k = 3$ (when $n \geq 2k - 2$). [4 marks]
(d) Show that the open-uniform B-spline with $k = 3$ and knot vector $[0, 0, 0, 1, 1, 1]$ is equivalent to the quadratic Bézier curve. [7 marks]

7. [2002/7/9] (d) Derive the formula of and sketch a graph of $N_{3,3}(t)$, the third of the quadratic B-spline basis functions, for the knot vector $[0, 0, 0, 1, 3, 3, 4, 5, 5, 5]$. [6 marks]

5 NURBS

NURBS are covered below and in some detail in **R&A** Section 5-13. An extract of this Section of **R&A** are included in the handout. Please read that before continuing here.

Non-uniform rational B-splines are the industry-standard for computer-aided design (CAD). In most cases, you would actually use the special case of non-rational B-splines (those described in the previous section) but it is useful to have the more general rational versions available for certain types of curve and surface.

NURBS surfaces are usually rendered by converting them to lots of small polygons and then using polygon scan conversion. They can also by ray traced, but a general analytic ray-NURBS intersection algorithm is challenging, so numerical techniques are used to find the intersection point.

NURBS curves incorporate – as special cases – uniform B-splines, non-rational B-splines, Bézier curves, lines, and conics. NURBS surfaces incorporate planes, quadrics, and tori. Note that this does not quite mean what it says. It is tricky to get NURBS to represent *infinite* surfaces, but they can certainly represent finite sections of infinite surfaces such as planes, paraboloids, and hyperboloids.

If you want to experiment with NURBS curves then there are a number of on-line tutorials. One such is available from the Technion.

Rational B-splines have all of the properties of non-rational B-splines plus the following two useful features:

- They produce the correct results under projective transformations (while non-rational B-splines only produce the correct results under affine transformations).
- They can be used to represent lines, conics, non-rational B-splines; and, when generalised to patches, can represent planes, quadrics, and tori.

In this case *rational* means "one polynomial divided by another" (see Equation 31). The antonym of *rational* is *non-rational* (i.e. a non-rational B-spline is just a polynomial, see Equation 16). Non-rational B-splines are a special case of rational B-splines, just as uniform B-splines are a special case of non-uniform B-splines. Thus, *non-uniform rational B-splines* encompass almost every other possible 3D shape definition. *Non-uniform rational B-spline* is a bit of a mouthful and so it is generally abbreviated to *NURBS*.

We have already learnt all about the the *B-spline* bit of *NURBS* and about the *non-uniform* bit. So now all we need to know is the meaning of the *rational* bit and we will fully understand NURBS.

Rational B-splines are defined simply by applying the B-spline equation (Equation 16) to homogeneous coordinates, rather than normal 3D coordinates. We discussed homogeneous coordinates in the IB course. You will remember that these are 4D coordinates where the transformation from 4D to 3D is:

$$(x', y', z', w) \rightarrow \left(\frac{x'}{w}, \frac{y'}{w}, \frac{z'}{w}\right) \tag{31}$$

Last year we said that the inverse transform was:

$$(x, y, z) \rightarrow (x, y, z, 1) \tag{32}$$

This year we are going to be more cunning and say that:

$$(x, y, z) \rightarrow (xh, yh, zh, h) \tag{33}$$

Thus our 3D control point, $\mathbf{P}_i = (x_i, y_i, z_i)$ gains an extra variable, h_i, and becomes the homogeneous control point, $\mathbf{C}_i = (x_i h_i, y_i h_i, z_i h_i, h_i)$.

A NURBS curve is thus defined as:

$$\mathbf{P}_H(t) = \sum_{i=1}^{n} N_{i,k}(t)\mathbf{C}_i, t_{\min} \leq t < t_{\max} \tag{34}$$

Compare Equation 34 with Equation 16 to see just how easy this is.

We now want to see what a NURBS curve looks like in normal 3D coordinates, so we need to apply Equation 31 to Equation 34. In order to better explain what is going on, we first write Equation 34 in terms of its individual components. Equation 34 is equivalent to:

$$x'(t) = \sum_{i=1}^{n} x_i h_i N_{i,k}(t) \tag{35}$$

$$y'(t) = \sum_{i=1}^{n} y_i h_i N_{i,k}(t) \tag{36}$$

$$z'(t) = \sum_{i=1}^{n} z_i h_i N_{i,k}(t) \tag{37}$$

$$h(t) = \sum_{i=1}^{n} h_i N_{i,k}(t) \tag{38}$$

Equation 31 tells us that, in 3D:

$$x(t) = x'(t)/h(t) \tag{39}$$

$$y(t) = y'(t)/h(t) \tag{40}$$

$$z(t) = z'(t)/h(t) \tag{41}$$

Thus the 4D to 3D conversion gives us the curve in 3D:

$$\mathbf{P}(t) = \frac{\sum_{i=1}^{n} N_{i,k}(t)\mathbf{P}_i h_i}{\sum_{i=1}^{n} N_{i,k}(t) h_i}, t_{\min} \leq t < t_{\max} \tag{42}$$

This looks a lot more fierce than Equation 34, but is simply the same thing written a different way.

So now, we need to define an additional parameter, h_i, for each control point, $\mathbf{P}_i$. The default is to set $h_i = 1, \forall i$. This results in the denominator of Equation 42 becoming one (because the $N_{i,k}(t)$ partition unity), and the NURBS equation (Equation 42) therefore reducing to the non-rational B-spline equation (Equation 16).

Increasing h_i pulls the curve closer to point $\mathbf{P}_i$. Decreasing h_i pushes the curve farther from point $\mathbf{P}_i$. Setting $h_i = 0$ means that $\mathbf{P}_i$ has no effect on the curve at all. See **R&A** Figure 5-58 for an example, and play with an on-line NURBS tutorials such as the one mentioned above.

5.1 An example: a circle defined by NURBS

This subsection provides an example of a shape which cannot be represented by non-rational B-splines: a circle. A non-rational B-spline or a Bézier curve cannot exactly represent a circle. An interesting exercise is to place a cubic Bézier curve's end points at $(0, 1)$ and $(1, 0)$, with the other control points at $(\alpha, 1)$ and $(1, \alpha)$. Now see how close this "quarter circle" comes to the real quarter circle defined by $x^2 + y^2 = 1$, i.e. what is the value of α for which the Bézier curve most closely matches the quarter circle. You will find that you can get a match which is almost, but not quite, circular.

NURBS *can* be used to represent circles, and all of the other conics. NURBS surfaces can be used to represent quadric surfaces. As an example, let us consider one way in which NURBS can be used to describe a true circle. **R&A** cover this on pages 371–375. The ways in which this is done require the designer to specify several things correctly at the same time.

The method is as follows. Construct eight control points in a square. Let $\mathbf{P}_1$, $\mathbf{P}_3$, $\mathbf{P}_5$, and $\mathbf{P}_7$ be the vertices of the square. Let $\mathbf{P}_0$, $\mathbf{P}_2$, $\mathbf{P}_4$, and $\mathbf{P}_6$ be the midpoints of the respective sides, so that the vertices are numbered sequentially as you proceed around the square. Finally, you need a ninth point to join up the curve, so let $\mathbf{P}_8 = \mathbf{P}_0$.

Use a quadratic B-spline basis function with the knot vector $[0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 4]$. This means that the curve will pass through $\mathbf{P}_0$, $\mathbf{P}_2$, $\mathbf{P}_4$, $\mathbf{P}_6$ and $\mathbf{P}_8$, and allows us to essentially treat each quarter of the circle independently. That is, we can just examine $\mathbf{P}_0$, $\mathbf{P}_1$, and $\mathbf{P}_2$, along with the knot vector $[0, 0, 0, 1, 1, 1]$. If this makes a quarter circle then the other three quarters will also be correct.

We finally need to specify the homogeneous co-ordinates. As a circle is symmetrical it should be obvious that that $h_1 = h_3 = h_5 = h_7 = \alpha$ and $h_0 = h_2 = h_4 = h_6 = h_8 = \beta$. As we would like the curve to pass through the even numbered points, the easiest thing to do is set $\beta = 1$. All we therefore need to determine is α, the value of the odd numbered homogeneous co-ordinates.

If $\alpha = 1$ then the NURBS curve will bulge out more than a circle. If $\alpha = 0$, it will bow in. This gives us limits on the value of α. To find the exact value we take the NURBS curve definition for the quarter circle:

$$\mathbf{P}(t) = \frac{(1-t)^2\mathbf{P}_0 + 2\alpha t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2}{(1-t)^2 + 2\alpha t(1-t) + t^2}, 0 \leq t < 1 \tag{43}$$

Assume now that $\mathbf{P}_0 = (0, 1)$, $\mathbf{P}_1 = (1, 1)$, and $\mathbf{P}_2 = (1, 0)$. Insert Equation 43 into the

equation for the unit circle ($x(t)^2 + y(t)^2 = 1$). The resulting equation is:

$$\frac{((1-t)^2 + 2\alpha t(1-t))^2 + (2\alpha t(1-t) + t^2)^2}{((1-t)^2 + 2\alpha t(1-t) + t^2)^2} = 1, 0 \leq t < 1 \tag{44}$$

Now solve this for α. Equation 44 is essentially:

$$\frac{a_N t^4 + b_N t^3 + c_N t^2 + d_N t + e_N}{a_D t^4 + b_D t^3 + c_D t^2 + d_D t + e_D} = 1, 0 \leq t < 1 \tag{45}$$

From this we can conclude that we require $a_N = a_D$, $b_N = b_D$, $c_N = c_D$, $d_N = d_D$, and $e_N = e_D$. The first three all solve to give the result that $\alpha = 1/\sqrt{2}$, while the last two cancel out totally to give the tautology $0 = 0$. Thus[2] $\alpha = 1/\sqrt{2}$.

An alternative way to calculate α is to realise that it is the only degree of freedom. Therefore, you need simply find the value of α that makes the curve pass through a fixed point on the circle, such as the intersection of the circle with 45 degree line.

This derivation is not intuitive and similar cleverness is required to handle representations of other conics. The beauty of NURBS is that they allow us to do this sort of thing and unify all shapes into a single representation. The difficulty is that, in order to achieve this unification, we need to have this rather complicated but general mathematical mechanism.

5.2 Exercises

1. Review from IB: What are homogeneous coordinates and what are they used for in computer graphics?

2. Explain how to use homogeneous coordinates to get rational B-splines given that you know how to produce non-rational B-splines.

3. What are the advantages of NURBS over Bézier curves? (i.e. why have NURBS, in general, replaced Bézier curves in CAD?)

4. Show that you understand why NURBS includes Uniform B-splines, Non-Rational B-splines, Béziers, lines, conics, quadrics, and tori.

5. [1998/7/12] Consider the design of a user interface for a NURBS drawing system. Users should have access to the full expressive power of the NURBS representation. What things should users be able to modify to give them such access and what effect does each have on the resulting shape? [6 marks]

6. For each of the items (in the previous question) that the user can edit: (i) Give sensible default values; (ii) Explain how they would be constrained if a 'demo' version of the software was to be limited to cubic Uniform Non-rational B-Splines.

7. [1999/7/11] (c) Show how to construct a circle using non-uniform rational B-splines (NURBS). [8 marks]
   Note: this question is challenging unless you remember and follow the worked example in these notes or **R&A** pages 371-375.

[2] If we had not set $\beta = 1$ above, then we would find that $\alpha = \beta/\sqrt{2}$.

(d) Show how the circle definition from the previous part can be used to define a NURBS torus. [4marks]
Note: you need explain only the general principle and the location of the torus' control points.

6 Subdivision

Subdivision schemes work by taking a coarse polygon mesh and introducing new vertices to create a finer mesh. Iterating this process several times creates a very fine mesh of polygons. In computer graphics, we are interested in drawing things only to a certain level of accuracy: there is no point in having polygons that are much smaller than pixels. This means that subdivision need only iterate until the polygons are about pixel-sized.

Subdivision schemes have been around for a long time. Subdivision methods for curves were first mathematically analysed in 1947. Their use in computer graphics dates from 1974 when Chaikin used them to derive a simple algorithm for generating curves quickly. In 1978 Doo & Sabin and Catmull & Clark generalised Chaikin's work from curves to surfaces.

Subdivision schemes are now the industry-standard modelling approach in computer animation and visual effects. NURBS remain the industry-standard approach in computer-aided design. The two approaches have the same foundations. The principal subdivision scheme used in practice is the *Catmull-Clark* scheme, invented in 1978, by Ed Catmull and Jim Clark, and commercialised in 1998 by Tony DeRose's team at *Pixar*. In the limit, as we subdivide infinitely finely, the *Catmull-Clark* scheme produces exactly the same surface as the uniform cubic B-spline, except around the so-called *extraordinary points* of the surface.

W&W and **P&R** both survey the field and the related mathematical tools. The course handout contains a slide presentation that presents the concepts from this section of the notes.

6.1 Subdivision curves

Take an arbitrary control polygon, comprised of a set of vertices connected in sequence. We use the positions of the current vertices to determine the location of the new vertices in a new, refined, more detailed, control polygon (see Figure 13). The standard approach is for each old vertex to give rise to two new vertices. For example, you could place new vertices one-quarter and three-quarters of the way between each adjacent pair of old vertices. Connecting all the new vertices together, in the appropriate order, produces a more refined control polygon. Repeat this process several times and you produce a good approximation to the curve that you would get if you were able to subdivide infinitely many times. For practical purposes, we need subdivide only a sufficient number of times that we cannot tell the refined polygon from a curve. For computer graphics, that is when the polygon segments have become a little smaller than a pixel.

Let the initial control polygon be defined by the sequence of control points:

$$\mathbf{P}^i = (\ldots, \mathbf{p}^i_{-1}, \mathbf{p}^i_0, \mathbf{p}^i_1, \mathbf{p}^i_2, \ldots)$$

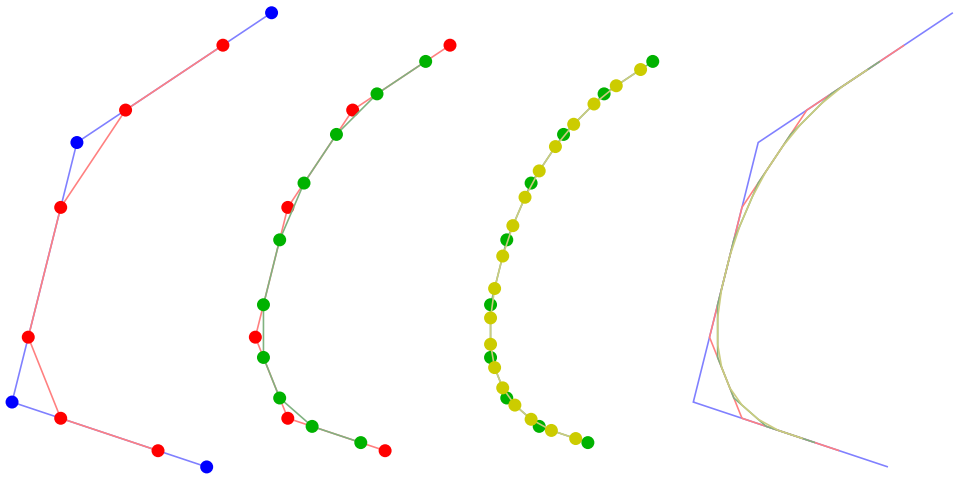


Figure 13: Chaikin's corner-cutting method. The first three diagrams show an original polygon with the subdivided version superimposed. The output polygon from the left hand diagram becomes the input to the second diagram, and its output becomes the input for the third. The right hand diagram shows all four polygons superimposed. The final two are very similar.

Subdivision maps this sequence of control points to a new sequence, $\mathbf{P}^{i+1}$ by applying subdivision rules. This process doubles[3] the number of points, and there is one rule for the odd numbered points and one for the even.

Chaikin's corner-cutting method introduces new points one-quarter and three-quarters of the way between each adjacent pair of old vertices. Informally, it "cuts the corners off" the original control polygon. Mathematically, it can be described thus:

$$\mathbf{p}_{2j}^{i+1} = \frac{3}{4}\mathbf{p}_j^i + \frac{1}{4}\mathbf{p}_{j+1}^i \tag{46}$$

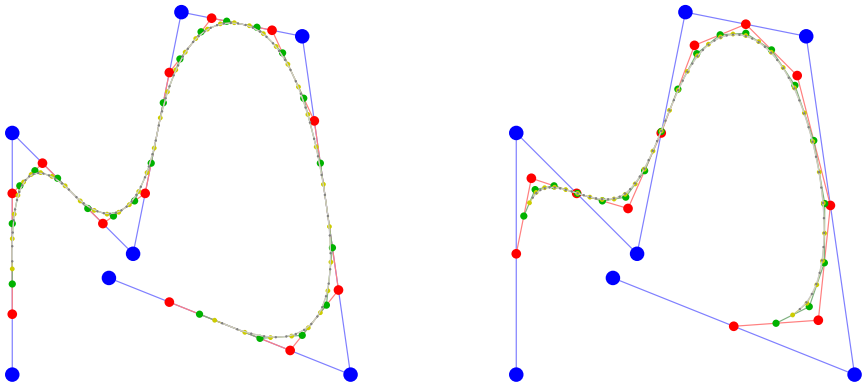
$$\mathbf{p}_{2j+1}^{i+1} = \frac{1}{4}\mathbf{p}_j^i + \frac{3}{4}\mathbf{p}_{j+1}^i \tag{47}$$

Figure 13 shows a polygon defined by four points subdivided three times using Chaikin's corner cutting. The difference in the polygons in the final two iterations is already small. Figure 14(a) shows a larger example with four levels of subdivision.

In the limit, after infinitely many steps of subdivision, Chaikin's corner-cutting method produces a curve identical to the quadratic uniform B-spline. As you will notice from Figure 14(a), it does not take many steps to produce a curve that is extremely close to the limit curve. In practice, therefore, we might need to take five or six steps of subdivision to produce a curve that looks smooth on a computer screen (say 100 dpi), and only three or four more steps to produce a curve that looks smooth on a printout (say, 1200 dpi).

Chaikin's corner-cutting method can be generalised to surfaces, producing the *Doo-Sabin* surface subdivision scheme (see Section 6.2).

[3]It doesn't quite double the number of points when the sequence is open and of finite length. If we are dealing with a finite-length vector $\mathbf{P}^i = (\mathbf{p}_1^i, \mathbf{p}_2^i, \ldots, \mathbf{p}_m^i)$, then the subdivided vector, using the rules in equations 46 and 47, would have $2m - 2$ points.



(a) (b)

Figure 14: (a) An example of Chaikin's corner-cutting method, whose limit curve is the quadratic uniform B-spline. (b) An example of the subdivision method whose limit curve is the cubic uniform B-spline. In both cases, the original polygon is blue and there are four levels of subdivision: red, green, yellow, grey.

The industry-standard surface subdivision method is *Catmull-Clark* subdivision. The curve subdivision rules on which the Catmull-Clark surface method is based are only slightly more complicated than the ones we have seen above. They are:

$$\mathbf{p}_{2j}^{i+1} = \frac{1}{8}\mathbf{p}_{j-1}^{i} + \frac{6}{8}\mathbf{p}_{j}^{i} + \frac{1}{8}\mathbf{p}_{j+1}^{i} \tag{48}$$

$$\mathbf{p}_{2j+1}^{i+1} = \frac{4}{8}\mathbf{p}_{j}^{i} + \frac{4}{8}\mathbf{p}_{j+1}^{i} \tag{49}$$

An example of a polygon subdivided by these rules can be seen in Figure 14(b). The limit curve generated by these rules is the cubic uniform B-spline.

As is the way with much mathematics, we can write it in a more compact, more general, but less obvious, form as:

$$\mathbf{p}_{j}^{i+1} = \sum_{k=-\infty}^{\infty} \alpha_{2k-j}\mathbf{p}_{k}^{i} \tag{50}$$

where the α_j are coefficients depending on the subdivision rules. Note that the index $2k - j$ alternately selects the even indexed α_j and the odd indexed α_j. So, the two schemes given above, can be compactly described as:

$$\alpha = \frac{1}{4}(\ldots, 0, 0, 1, 3, 3, 1, 0, 0, \ldots) \tag{51}$$

and

$$\alpha = \frac{1}{8}(\ldots, 0, 0, 1, 4, 6, 4, 1, 0, 0, \ldots) \tag{52}$$

respectively. You will recognise the sequences in parentheses as being two rows from Pascal's triangle.

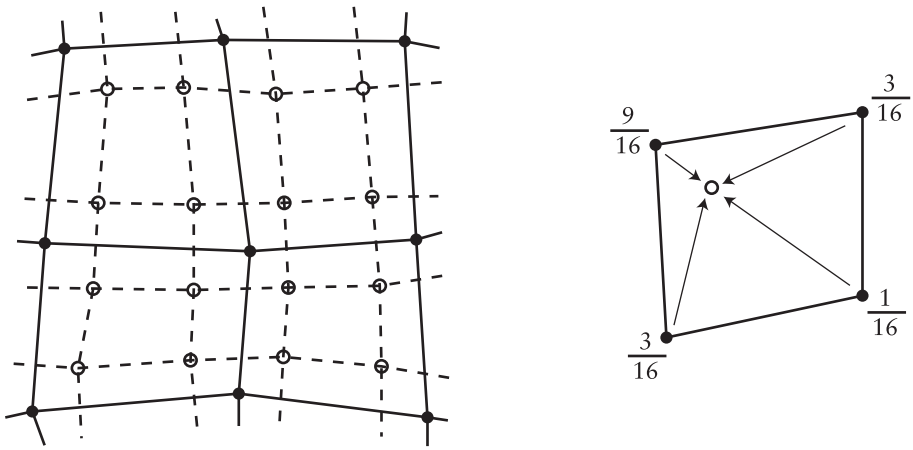


Figure 15: Doo-Sabin subdivision. On left a mesh (solid dots and solid lines) that has been refined (open dots and dashed lines). At right the weights used to generated one of the refined vertices.

6.2 Subdivision surfaces

The above subdivision methods can be easily extended from a control polygon to a quadrilateral mesh. This is a mesh where every polygon is a quadrilateral and every vertex is connected to four other vertices.

The Doo-Sabin subdivision method for surfaces is a generalisation of Chaikin's corner cutting method for curves. It introduces four new vertices in each quadrilateral, and connects up vertices accordingly. The new vertices are blended mixtures of the old vertices in the proportions $9:3:3:1$ (derived from the tensor product of the univariate case: $3\times 3:3\times 1:1\times 3:1\times 1$). This is illustrated in Figure 15.

This all works beautifully for quadrilateral meshes. In this case it produces a limit surface that is identical to the uniform quadratic B-spline surface, defined with the same control points. The surface has C^1-continuity everywhere, as this is a property of the uniform quadratic B-spline surface.

Now, suppose we have a quadrilateral mesh that contains *extraordinary polygons*, that is, polygons that are not quadrilateral, such as the pentagon in Figure 16. The Doo-Sabin scheme copes with meshes that have such polygons. For a k-sided polygon, the weights, α_k on the k vertices need to be chosen. Doo and Sabin demonstrated that good results are achieved if we let:

$$\alpha_0 = \frac{1}{4}+\frac{5}{4k} \tag{53}$$

$$\alpha_i = \frac{1}{4k}\left(3+2\cos\frac{2i\pi}{k}\right) \tag{54}$$

The resulting limit surface can be shown to be C^1 everywhere.

Now consider *extraordinary vertices*, that is, vertices with other than four immediate neighbours. You will see in Figure 16 that such vertices become extraordinary polygons in the first step of subdivision. It is impossible for the Doo-Sabin to introduce new extraordinary vertices: every vertex in the subdivided mesh has four neighbours. It is

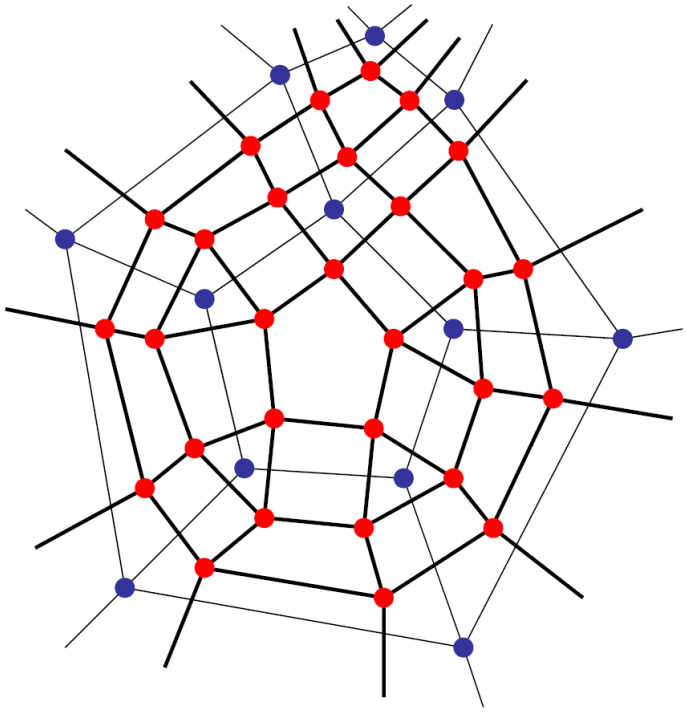


Figure 16: A Doo-Sabin mesh before (blue points, thin lines) and after (red points, thick lines) one level of subdivision. Notice that every polygon produces a smaller polygon of the same type (quadrilaterals produce quadrilaterals, pentagons produce pentagons), every edge generates a new quadrilateral, every vertex generates a polygon of the same valency, and that every new vertex is of valency four.

also impossible for the Doo-Sabin method to introduce new extraordinary polygons after the first step. At each step, every extraordinary polygon shrinks to a smaller polygon of the same type. All the new polygons are quadrilaterals. As we subdivide further, every extraordinary polygon becomes surrounded by a "sea" of quadrilaterals. Thus you can see that the limit surface is the uniform quadratic B-spline defined by all of these quadrilaterals, except at the centres of the extraordinary polygons, where additional mathematical analysis is needed to demonstrate that the limit surface is also C^1 at these points.

Catmull-Clark subdivision is a surface method that generalises the $\frac{1}{8}(1,4,6,4,1)$ subdivision curve method. It is similar in spirit to Doo-Sabin, with some important differences. The first difference is that is there are three types of new vertex in the normal regions of the mesh: a vertex is introduced in the centre of each quadrilateral (a *face vertex*), in the centre of each edge (an *edge vertex*), and near to each old vertex (a *vertex vertex*). Each of these three types of vertex has a different set of weights as illustrated in Figure 17.

Catmull-Clark subdivision needs special rules for *extraordinary vertices*. The edge vertex and face vertex rules remain unchanged, but we need a new rule for the vertex vertex generated by the extraordinary vertex. Catmull and Clark solved this by creating

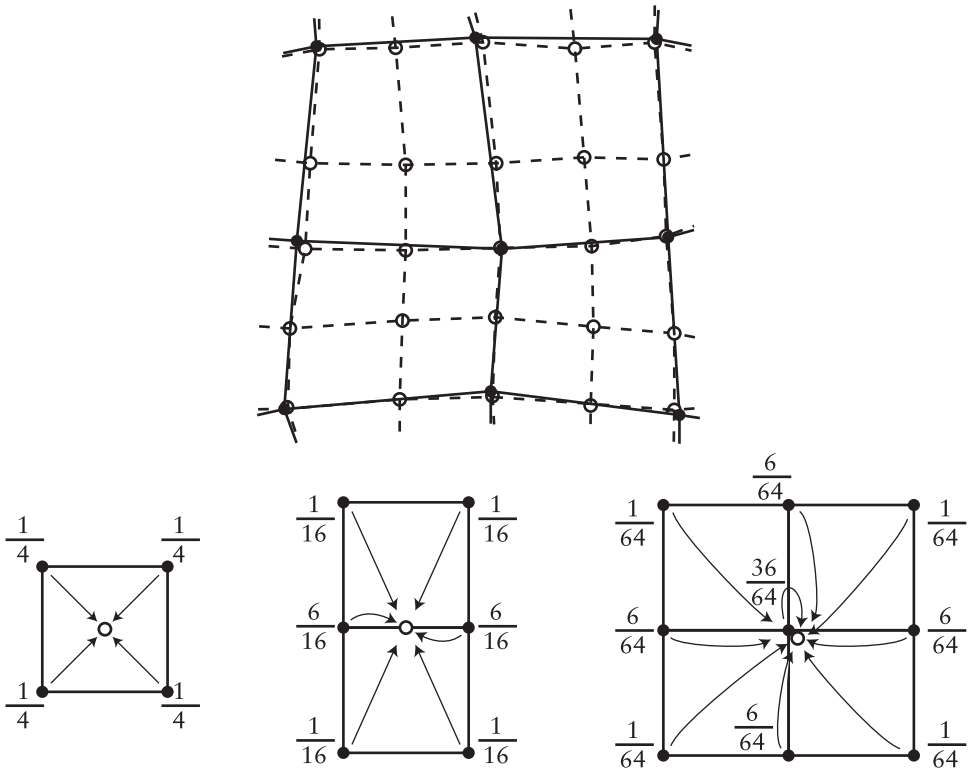


Figure 17: Catmull-Clark subdivision. Above: a mesh (solid dots and solid lines) that has been refined (open dots and dashed lines). Below: the weights used to generated each type of refined vertex: centre, edge, and modified old vertex.

a new set of weights, one set of weights for each vertex valence (the valence of vertex is a number of other vertices to which it is connected). There has since been considerable work on finding the optimal weights to give the best result. For example, instead of weights of $1/64$, $6/64$, and $36/64$ you can use Denis Zorin's weights of $1/4n^2$, $3/2n^2$, and $1 - 7/4n$, where n is the valence of the vertex.

The limit surface of the Catmull-Clark scheme is C^2 *almost* everywhere. It is, however, only C^1 at the limit positions of the extraordinary vertices. It has been proven that the only way to make a Catmull-Clark limit surface C^2 at the extraordinary vertices is for its curvature to be zero at those points. This produces unsightly "flat spots" that we would rather avoid. This means that we either have to accept "flat spots" or have to accept that the surface is not everywhere C^2.

There are other subdivision schemes. The most notable are the *Butterfly* scheme, which interpolates the control points and the *Loop* scheme (named after Charles Loop), which works on triangular meshes.

6.3 Exercises

1. Draw an arbitrary control polygon and perform a couple of subdivision steps using the first of the two subdivision schemes above. Once you feel happy that you

understand what is going on, you may like to try the second scheme.

2. Draw an arbitrary control polygon and consider what happens if you try to use the previous row from Pascal's triangle $\frac{1}{2}(1,2,1)$. You will find that $\frac{1}{2}(1,2,1)$ has a minimal effect on the *shape* of the control polygon.

3. What happens if you try to use the next row of Pascal's triangle, $\frac{1}{16}(1,5,10,10,5,1)$? Which uniform B-spline do you think this produces in the limit?

4. Explain how Doo-Sabin subdivision works for an arbitrary polygon mesh.

5. Explain how the Catmull-Clark scheme handles extraordinary polygons.