Chapter 1

Bézier curves and Splines

This a very incomplete and rough draft, so please read it with caution. Many topics are just skipped, and there are no pictures at all. My suggestion is that you skim it quickly and read any sections that appear both relevant and accessible. To study the midterm, you are better off concentrating on notes from lectures.

A spline curve, or spline surface, is a smooth curve or surface which is specified compactly in terms of a few points. Both aspects of splines of curves are important. First, the ability to specify a curve with only a few points reduces storage requirements and greatly facilitates computer-aided design of curves and surfaces. Second, the commonly used splines gives curves with good smoothness properties and without undesired oscillations; and, conversely, these splines also allow for isolated points where the curve is not smooth, such as points where the spline has a 'corner'. Finally, splines provide us with simple algorithms for finding points on the spline curve or surface, and simple criteria for deciding how finely a spline should be approximated by flat patches to get a sufficiently faithful representation of the spline.

Historically, splines were flexible strips of wood or medal, that were tied into position to record a desired curve. Nowadays, a mathematical description is much more useful and more permanent, not to mention more amenable to computerization. Nonetheless, some of the terminology of splines persists, such as the use of 'knots' in B-spline curves.

This chapter discusses first Bézier curves and then uniform and non-uniform B-splines, including NURBS.

1.1 Beźier curves

Bézier curves were first developed by automobile designers for the purpose of describing the shape of exterior car panels. The original developer of Bézier curves was P. Bézier of Renault in the 1960's [?]. Slightly earlier, P. de Casteljau at Citroën had independently developed mathematically equivalent methods of defining spline curves [?].

XX TODO

The most common type of Bézier curves is the *degree 3* polynomial curves which are specified by four points, called *control points.* This is illustrated in Figure 1.1, where a parametric curve $\mathbf{Q} = \mathbf{Q}(u)$ is defined by four control points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$. The curve starts from $\mathbf{p}_0$ initially in the direction of $\mathbf{p}_1$, then curves generally towards $\mathbf{p}_2$, and ends up at $\mathbf{p}_3$ coming from the direction of $\mathbf{p}_2$. Only the first and last points, $\mathbf{p}_0$ and $\mathbf{p}_3$, are lie on $\mathbf{Q}$. The other two control points, $\mathbf{p}_1$ and $\mathbf{p}_2$, influence the curve: the intuition is that these two middle control points "pull" on the curve. One can think of $\mathbf{Q}$ as being a flexible, strechable curve that is contrained to start at $\mathbf{p}_0$, end at $\mathbf{p}_2$, and in the middle is pulled by the two middle control points.

XX LOTS MORE TO BE WRITTEN HERE XX

1.2 Bézier surfaces

XX TO BE WRITTEN XX

1.3 Bézier curves and surfaces in OpenGL

1.3.1 Bézier curves

OpenGL has several routines for automatic generation of Bézier curves of any degree. Unfortunately, OpenGL does not have generic Bézier curve support; instead OpenGL has Bézier curve linked directly to drawing routines, so that the the OpenGL Bézier curve routines can be used only for drawing. So if you wish to use Bézier curves for other applications such as animation, you cannot use the built-in OpenGL routines.

Rather than having a single command for generating Bézier curves, OpenGL has instead separate commands for specifying a Bézier curve from its control points and for displaying part of all of the Bézier curve.

Defining Bézier curves. To define and enable (i.e., activate) a Bézier curve, the following two OpenGL commands are used:

```
glMap1f(GL_MAP1_VERTEX_3, float u_min , float u_max ,
          int stride, int order, float* controlpoints )
glEnable(GL_MAP1_VERTEX_3)
```

The values of u_{min} and u_{max} give the range of u values over which the curve is defined. There are typically set to 0 and 1.

The last parameter points to an array of floats which contain the control points. A typical usage would define `controlpoints` as an array of x, y, z values:

```
float controlpoint[N][3];
```

The `stride` value is the distance (in floats) from one control point to the next; i.e., the i-th control point is pointed to by `controlpoints+`i`*stride`. For the above definition of `controlpoints`, *stride* equals 3.

The value of `order` is equal to one plus the degree of the Bézier curve: thus it also equals the number of control points. Thus, for the usual degree 3 Bézier curves, `order` equals 4.

As we mentioned above, Bézier curves can be used only for drawing purposes. In fact, several Bézier curves can be active at one time to affect different aspects of the drawn curve, such as its location and color, etc. The first parameter to `glMap1f()` describes how the Bézier curve is used when the curve is drawn. `GL_MAP1_VERTEX_VERTEX_3` means that the Beziér curve is defining the x, y, z values of point in 3-space, as a function of u. There are several other useful constants that can be used for the first parameter. These include `GL_MAP1_VERTEX_4`, which means that we are x, y, z, w values of a curve, i.e., a rational Bézier curve. Also, one can use `GL_MAP1_COLOR_4` as the first parameter: this means that as the Bézier curve is being drawn (by the commands described below), that color values will be specified as a Bézier function of u. The option `GL_MAP1_TEXTURE_COORD_1` works similarly to specify one-dimensional texture coordinates. See the OpenGL documentation for other permitted values for this first parameter.

Drawing with Bézier curves. Once the Bézier curve has been specified with `glMap1f()`, the curve can be drawn with the following commands. The most basic way to specify a point on the curve is with the command:

```
glEvalCoord1f( float u )
```

which must be given between a `glBegin()` and `glEnd()`. The effect of the this command is similar to specifying a point with `glVertex*` and, if the appropriate curves are enabled, with `glNormal*` and `glTexCoord*` commands. However, the currently active normal and texture coordinates are not changed by a call to `glEvalCoord1()`.

If you use `glEvalCoord1f()`, then you explicitly draw all the points on the curve. However, frequently you want to draw points at equally spaced intervals along the curve and OpenGL has also several commands for drawing an entire curve or a portion of curve at once. First, you must tell OpenGL the "grid" or "mesh" of points on the curve to be drawn. This is done with the following command:

```
glMapGrid1f(int N, float u_start, float u_end);
```

which tells OpenGL that you want to draw $N+1$ points on the curve equally spaced starting with the value $u = u_{start}$ and ending with $u = u_{end}$. It is required that $u_{min} \leq u_{start} \leq u_{end} \leq u_{max}$.

A call to glMapGrid1f() only sets a grid of u values. In order to actually draw the curve, you should call

```
glEvalMesh1(GL_LINE, int p_start, int p_end)
```

This causes OpenGL to draw the curve at grid values, letting p range from p_{start} to p_{end}, and drawing the points on the Bézier curve with coordinates

$$u \;=\; \left((N-p)u_{start} + p \cdot u_{end}\right)/N.$$

The first parameter, `GL_LINE` tells OpenGL to draw the curve as sequence of straight lines, this the same functionality drawing points after a call to `glBegin(GL_LINE_STRIP)`. To draw only the points on the curve without the connecting lines, use `GL_POINT` instead (similar in functionality to using `glBegin(GL_POINTS)`). The values of p_{start} and p_{end} should satisfy $0 \le p_{start} \le p_{end} \le N$.

You can also use `glEvalPoint1( int` p `)` to draw a single point from the grid. `glEvalPoint1` and `glEvalMesh1` are **not** called from inside `glBegin()` and `glEnd()`.

1.3.2 Bézier patches

Bézier patches, or Bézier surfaces, can be drawn using OpenGL commands analogous to the commands describe above for Bézier curves. Since the commands are very similar, only very brief descriptions are given of the OpenGL routines for Bézier patches.

To specify a Bézier patch, one uses the `glMap2f()` routine:

```
glMap2f(GL_MAP2_VERTEX_3,
        float u_min, float u_max, int ustride, int uorder,
        float v_min, float v_max, int vstride, int vorder,
        float* controlpoints ) glEnable(GL_MAP2_VERTEX_3);
```

The `controlpoints` array is no a `(uorder)`$\times$`(vorder)` array and would usually be specified by:

```
float controlpointsarray[N_u][N_v][3];
```

In this case, the value `vstride` would equal 3, and `ustride` should equal $3N_v$. Note that the order's (one plus the degree's) of the Bézier curves are allowed to be different for the u and v directions.

Other useful values for the first parameter to `glMap2f()` include `GL_MAP2_VERTEX_4` for rational Bézier patches, `GL_MAP2_COLOR_4` to specify colors, and `GL_MAP2_TEXTURE_COORD_2` to specify texture coordinates. Generally, one wants the texture coordinates s, t to just be equal to u and v. This is done by specifying a degree one (`order`$=2$) Bézier curve, for instance,

```
float texpts[8]={0,0, 0,1, 1,0, 1,1};
glMap2f(GL_MAP2_TEXTURE_COORD_2,0,1,4,2,0,1,2,2,texpts);
glEnable(GL_MAP2_TEXTURE_COORD_2);
```

The normals to the patch may be specified by a Bézier formula using `GL_MAP2_NORMAL` as the first parameter to `glMap2f()`; however, this is rarely useful since usually one wants the true normals to the Bézier surface. OpenGL will calculate these true normals for you (according to the formula XX), if you give the command `glEnable(GL_AUTO_NORMAL)`.

To display the Bézier patch, or a portion of the Bézier surface, the following OpenGL commands are available:

```
glEvalCoord2f(float u, float v);
glMapGrid2f(int N_u, int u_start, int u_end,
        int N_v, int v_start, int v_end);
glEvalMesh2(GL_FILL, int p_start, p_end, q_start, q_end);
glEvalPoint2(int p, int q);
```

The first parameter to `glEvalMesh2()` may be also `GL_LINE` or `GL_POINTS`.

1.4 B-splines

B-splines provide a method of specifying a single curve with many control points. They has many advantages over Bézier curves, especially in allowing a great deal of flexibility in specifying characteristics of the curve. In addition, it is possible to translate Bézier curves into B-splines and vice-versa.

We shall first treat the case of uniform B-splines, and then the more general case of non-uniform B-splines and NURBS.

1.4.1 Uniform B-splines

Before introducing the general definition of B-splines in the next section 1.4.2, this section will introduce one of the simplest and most useful cases of B-splines, namely the *uniform B-splines of degree 3.* Such a B-spline is defined with a sequence $\mathbf{p}_0, \mathbf{p}_1, p_2, \ldots, \mathbf{p}_n$ of *control points.* Together with a set of blending (or basis) blending functions $B_1(u), \ldots, B_n(u)$, this parametrically defines a curve $\mathbf{Q}(u)$:

$$\mathbf{Q}(u) \;=\; \sum_{i=0}^{n} B_i(u) \cdot \mathbf{p}_i \qquad\qquad 3 \le u \le n+1\,.$$

We define these blending functions below: it is important to note that they are *not* the same as the Bernstein polynomials used in the definition of Bézier curves.

An important property of the blending functions B_i is that $B_i(u)$ will equal zero if either $u \le i$ or $i+4 \le u$. That is to say, the support of $B_i(u)$ is

XX TODO

Figure 1.1: A uniform B-spline curve with seven control points.

XX TODO

Figure 1.2: The blending functions for a uniform, degree 3 B-spline

the open interval $(i, i+4)$. In particular, this means that we can rewrite the formula for $\mathbf{Q}(u)$ as

$$\mathbf{Q}(u) \;=\; \sum_{i=j-3}^{j} B_i(u)\cdot \mathbf{p}_i \qquad \text{provided } u\in[j,j+1],\ 3\le j\le n \tag{1.1}$$

since the terms omitted from the summation are all zero. This means that the B-spline has *local control*; namely, if a single control point $\mathbf{p}_i$ is moved then, then only the segments $\mathbf{Q}_i, \mathbf{Q}_{i+1}, \mathbf{Q}_{i+2}, \mathbf{Q}_{i+3}$ of the B-spline are changed. The rest of the B-spline remains fixed.

Figure 1.4.1 shows an example of a B-spline curve $\mathbf{Q}(u)$ defined with seven control points and defined for $3\le u\le 7$. The curve $\mathbf{Q}$ is split into four subcurves $\mathbf{Q}_3,\ldots\mathbf{Q}_6$ where $\mathbf{Q}_3$ is the portion of $\mathbf{Q}(u)$ corresponding to $3\le u\le 4$, $\mathbf{Q}_4$ is the portion with $4\le u\le 5$, etc. More generally, $\mathbf{Q}_i(u)=\mathbf{Q}(u)$ for $i\le u\le i+1$.

The intuition of how the curve $\mathbf{Q}(u)$ behaves is as follows. At the beginning of Q_3, the point on the curve $\mathbf{Q}$ where $u=3$ is being pulled strongly towards the point $\mathbf{p}_1$ and less strongly towards the points $\mathbf{p}_0$ and $\mathbf{p}_2$. The other points on $\mathbf{Q}_3$ are calculated as a weighted average of $\mathbf{p}_0,\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3$. The other segments are similar: namely, the beginning of Q_i is being pulled strongly towards $\mathbf{p}_{i-2}$, the end of Q_i is being pulled strongly towards $\mathbf{p}_{i-1}$, and the points interior to Q_i are computed as a weighted average of the four control points $\mathbf{p}_{i-3},\mathbf{p}_{i-2},\mathbf{p}_{i-1},\mathbf{p}_i$. Finally, the segments Q_i are degree 2 polynomials; thus, $\mathbf{Q}$ is piecewise a degree 3 polynomial. Furthermore, $\mathbf{Q}$ has continuous second derivatives everywhere is it defined.

These properties of the curve $\mathbf{Q}$ all depend on properties of the blending functions B_i. Figure 1.4.1 shows the graphs of the functions $B_i(u)$ — the functions are defined to equal zero outside their domain shown with their graphs. It is immediate from the graphs of the blending functions that equation (1.1) is correct. Also, it is now clear that a control point $\mathbf{p}_i$ affects only the four segments $\mathbf{Q}_i, \mathbf{Q}_{i+1}, \mathbf{Q}_{i+2}, \mathbf{Q}_{i+3}$.

The blending functions should have the following properties:

(a) The blending functions are translates of each other, i.e.,

$$B_i(u) \;=\; B_0(u-i).$$

(b) The functions $B_i(u)$ are piecewise degree 3 polynomials. The breaks between the pieces occur only at integer values of i.

(c) The functions $B_i(u)$ have continuous second-derivatives, i.e., are C^2-continuous.

(d) The blending functions are a *partition of unity*, i.e.,

$$\sum_i B_i(u) \;=\; 1.$$

for $3 \le u \le 7$. (Or for $3 \le u \le n$ when there are n control points.) This property is necessary for points on the B-spline curve to be defined as weighted averages of the control points.

(e) $B_i(u) \ge 0$ for all i and u. Therefore, $B_i(u) \le 1$ for all u.

(f) $B_i(u) = 0$ for $u \le i$ and for $i+4 \le u$.

Because of conditions (a) and (f), the blending functions will be fully specified once we specify the function $B_0(u)$ on the domain $[0,4]$. For this purpose, we will define four functions R_0, R_1, R_2, R_3 for $0 \le u \le 1$ by

$$\begin{aligned} R_0(u) &= B_0(u) & \qquad R_2(u) &= B_0(u+2) \\ R_1(u) &= B_0(u+1) & \qquad R_3(u) &= B_0(u+3) \end{aligned}$$

Thus the function R_i are the translates of the four segments of B_0 to the interval $[0,1]$. These functions are degree 3 polynomials by condition (b). In fact, we claim that the following choices for the R_i functions work (and this is the unique way to define these functions):

$$\begin{aligned} R_0(u) &= \tfrac{1}{6}u^3 \\ R_1(u) &= \tfrac{1}{6}(-3u^2 + 3u^2 + 3u + 1) \\ R_2(u) &= \tfrac{1}{6}(3u^3 - 6u^2 + 4) \\ R_3(u) &= \tfrac{1}{6}(1-u)^3 \end{aligned}$$

We claim that if the functions R_0, R_1, R_2, R_3 are used to define the blending functions, then the blending functions have continuous second derivatives and condition (c) holds. To prove this, we need to know that the curves also are continuous and have continuous first-derivaties, so that the second-derivatives of the curves are defined. All these facts are proved by noticing that when the R_i functions are pieced together, their values and their first- and second-derivatives match up. Namely,

$$\begin{array}{lll} R_0(0) = 0 & R_0'(0) = 0 & R_0''(0) = 0 \\ R_0(1) = R_1(0) & R_0'(1) = R_1'(0) & R_0''(1) = R_1''(0) \\ R_1(1) = R_2(0) & R_1'(1) = R_2'(0) & R_1''(1) = R_2''(0) \\ R_2(1) = R_3(0) & R_2'(1) = R_3'(0) & R_2''(1) = R_3''(0) \\ R_3(1) = 0 & R_3'(1) = 0 & R_3''(1) = 0 \end{array}$$

Exercise: Graph the four functions R_i on the interval $[0,1]$.

Exercise: Give formulas for the first and second derivatives of the R_i functions. Verify the 15 conditions needed for the C^2-continuity of the blending function B_0.

Exercise: Verify that $\sum_i R_i(u) = 1$. Prove that $R_i(u) > 0$ for $i = 0, 1, 2, 3$ and for all $u \in (0, 1)$.

Exercise: Verify that $R_0(u) = R_3(1-u)$ and that $R_1(u) = R_2(1-u)$. Show that this means that uniform B-splines have left-right symmetry, in that if the order of the control points is reversed, then the curve $\mathbf{Q}$ is unchanged except for being tranversed in the opposite direction.

Exercise: Describe the effect of repeating control points in degree 3 uniform B-splines. Qualitatively describe the curve obtained if a control point is repeated, for instance, if $\mathbf{p}_3 = \mathbf{p}_4$.

Secondly, suppose $\mathbf{p}_3 = \mathbf{p}_4 = \mathbf{p}_5$. Show that the curve $\mathbf{Q}$ interpolated the point $\mathbf{p}_3$ with $\mathbf{Q}(6)$. Further show that the segments $\mathbf{Q}_5$ and $\mathbf{Q}_6$ are straight lines.

1.4.2 Non-uniform B-splines

The uniform B-splines of the previous section were defined so that curve $\mathbf{Q}$ was affected by the control points so that $\mathbf{Q}(i)$ is close to (or at least, strongly affected by) the control point P_{i-2}. These splines are called "uniform" since the values $u_i = i$, where the curve point $\mathbf{Q}(i)$ are most strongly affected by a control point, are evenly spaced at integer values. A *non-uniform* spline is one where the are values u_i where $\mathbf{Q}(u)$ is most strongly affected by some control point are not uniformly spaced. Of course, the uniform B-splines are just the special case of non-uniform B-splines where $u_i = i$.

We define a *knot vector* to be a sequence $u_0 s \le u_1 \le u_2 \le \cdots \le u_{m+k-1} \le u_{m+k}$ of real numbers, called *knots*. A knot vector is used with a sequence of $m+1$ control points, $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_m$ to define a non-uniform B-spline curve. The best way to get a qualitative understanding of how knots work is to consider a few examples. You should think of the spline curve as being a flexible and streachable curve: the flexibility of the curve is limited and thus the curve resists being sharply bent. he curve is parameterized by u values and we think of u as measuring the time spent traversing the length of the curve. The control points are tied to the curve with a knots being placed at the positions $u = u_i$ along the curve; you should mentally think of the knot being a strechy string attached to the curve at u_i and tied also to the control point $\mathbf{p}_i$. These knots pull on the spline and the spline settles down into a smooth curve.

It is possible for knots to be repeated as $u_i = u_{i+1}$. If a knot position occurs twice, i.e., if $u_{i-1} < u_i = u_{i+1} < u + i + 2$, then the curve will be affected extra strongly by the corresponding control point, $\mathbf{p}_{i+1}$ (and the control point $\mathbf{p}_i$ will just be ignored). If a knot position occurs three times, with $u_{i-1} < u_i = u_{i+1} = u_{i+2} < u_{i+3}$, then, for degree 3 B-splines, the curve will interpolate the point $\mathbf{p}_{i+2}$ and will generally have a "corner" at $\mathbf{p}_{i+2}$ and thus not be G_1-continuous. If a knot position occurs four times (in a degree 3 curve), then the curve will actually become discontinuous! Thus knots which repeat four times are usually used only at the beginning or end of the knot vector.

All these behaviors are exhibited in the examples in Figure 1.4.2.

Next, we give the Cox-de Boor mathematical definition of non-uniform B-spline blending functions. So far, all of examples have been for degree 3 splines, but it now convenient to generalize to degree $k-1$ splines, which are also called *order* k splines. Assume the knot vector $u_0 \le u_1 \le \cdots \le u_n$ has been fixed. The blending functions $B_{i,k}$ for order k splines are defined by induction on $k \ge 1$ as follows. First, let

$$B_{i,1}(u) \;=\; \begin{cases} 1 & \text{if } u_i \le u < u_{i+1} \\ 0 & \text{otherwise.} \end{cases}$$

There is one minor exception to the above definition to include the very last point $u = u_n$ in the domain of the functions: namely, we let $B_{n-1,1}(u) = 1$ for $u_{n-1} \le u \le u_n$. In this way, the theorems stated below hold also for $u = u_n$. Second, for $k \ge 1$,

$$B_{i,k+1}(u) \;=\; \frac{u-u_i}{u_{i+k}-u_i} B_{i,k}(u) + \frac{u_{i+k+1}-u}{u_{i+k+1}-u_{i+1}} B_{i+1,k}(u).$$

When multiple knots occur, some of the denominators above may be zero: we adopt the convention that $0/0 = 0$ and $(a/0)0 = 0$. Since $B_{i,k}(u)$ will be identically zero when $u_{i+k} = u_k$ (see the next paragraph), this means that any term with denominator equal to zero may be ignored.

The form of the Cox-de Boor recursive formulas for the blending functions immediately implies that the functions $B_{i,k}$ are piecewise degree $k-1$ polynomials and that the breaks between pieces occur at the knots u_i. Secondly, it is easy to prove, by induction on $k \ge 2$, that the function $B_{i,k}$ have support in $[u_i, u_{i+k}]$, i.e., $B_{i,k}(u) = 0$ for $u < u_i$ and for From these considerations, it follows that the definition of the blending function $B_{i,k}$ depends only on the knots $u_i, u_{i+1}, \ldots, u_{i+k}$.

TO DO: AN EXAMPLE OR TWO

Theorem 1 *Let $u_0 \le u_1 \le \cdots \le u_n$ be a knot vector. Then the blending functions $B_{i,k}$, for $0 \le i \le n-k$, satisfy the following properties.*

(1) *$B_{i,k}$ has support in $[u_i, u_{i+k}]$ for all $k \ge 2$.*

(2) *$B_{i,k}(u) \ge 0$ for all u.*

(3) *$\sum_{i=0}^{n-k} B_{i,k}(u) = 1$, for all u such that $u_{k-1} \le u \le u_{n-k+1}$.*

Proof As discussed above, conditions (1) and (2) are readily proved by induction on k. Condition (3) is also proved by induction on k, by the following argument. The base case, with $k = 1$ is obviously true. For the induction step, assume $u_k \le u \le u_{n-k}$:

$$\begin{aligned}
\sum_{i=0}^{n-k-1} B_{i,k+1}(u) \; &= \\
&= \sum_{i=0}^{n-k-1} \left(\frac{u-u_i}{u_{i+k}-u_i} B_{i,k}(u) + \frac{u_{i+k+1}-u}{u_{i+k+1}-u_{i+1}} B_{i+1,k}(u) \right) \\
&= \frac{u-u_0}{u_k-u_0} B_{0,k}(u) \; + \sum_{i=1}^{n-k-2} \frac{(u-u_i)+(u_{i+k}-u)}{u_{i+k}-u_i} B_{i,k}(u) \\
&\qquad + \frac{u_n-u}{u_n-u_n} B_{n-k-1,k}(u) \\
&= 0 \; + \sum_{i=1}^{n-k-1} 1 \cdot B_{i,k}(u) \; + \; 0 \\
&= 0.
\end{aligned}$$

The first zero in the next to last line above is justfied by the fact that $u \ge u_k$ and by (1). □

The importance of conditions (2) and (3) is that they allow the blending functions to be used as coefficients of control points to give a weighted average of control points. To define an order k (degree $k-1$) B-spline curve, one needs $n = m + k$ knot positions and m control points $\mathbf{p}_0, \ldots, \mathbf{p}_m$. Then the B-spline curve equals

$$\mathbf{Q}(u) \; = \; \sum_{i=0}^{m} B_{i,k}(u)\mathbf{p}_i$$

for $u_{k-1} \le u \le u_{n-k+1}$. The bounded interval of support given in condition (1) means that

$$\mathbf{Q}(u) \; = \; \sum_{i=j-k+1}^{j} B_{i,k}(u)\mathbf{p}_i$$

provided $u_j \le u \le u_{j+1}$ and $u_j < u_{j+1}$. Thus the control points provide *local control* over the B-spline curve, as changing a control point only affects k segments of the B-spline curve.

We next discuss the smoothness properties of a B-spline curve. Since a B-spline consists of pieces which are degree $k-1$ polynomials, it is certainly continuous and C^∞-continuous everywhere, except possible at knot positions. If there are no repeated knots and if $k > 1$, then the curve is in fact continuous everywhere in its domain and, even more, the curve is C^{k-2}-continuous everywhere in its domain. For instance, a degree 3 B-spline has its second derivatives defined and continuous everywhere in its domain.

Now we consider the case of repeated knots. We say that a knot has *multiplicity* p if it occurs p times in the knot vector. Since the knots are ordered, these p occurences must be be consecutive values in the knot vector, i.e., we have

$$u_{i-1} < u_i = u_{i+1} = \cdots = u_{i+p-1} < u_p.$$

In this case, the curve will have its $k-p-1$-th derivative defined and continuous at $u = u_i$. For instance, a degree 3 B-spline will have continuous first derivative at a twice repeated knot position, but will be only continuous at a knot position of multiplicity 3. In the latter case, the curve will generally have a "corner" or "bend" at that knot position. A B-spline curve of degree 3 can ne discontinuous at a knot position of multiplicity four.

The ability to repeat knots and make the curve have fewer continuous derivatives is important for the usefulness of B-splines since it allows a single curve to include both smooth portions and sharply bending portions.

We combine the above assertions about the smoothness of B-splines into the next theorem.

Theorem 2 *Let the knot u_i have multiplicity p. Then the curve $\mathbf{Q}(u)$ has continuous $(k-p-1)$-th derivative at $u = u_i$.*

We shall prove this theorem from the next two lemmas. The first lemma, which is interesting in its own right, gives formulas for the first derivatives of the blending functions.

Lemma 3 *Fix a knot vector and let $l \geq 1$. If $B_{i,k+1}$ has a first derivative at u, then*

$$B'_{i,k+1}(u) \;=\; \frac{k}{u_{i+k} - u_i} B_{i,k}(u) - \frac{k}{u_{i+k+1} - u_{i+1}} B_{i+1,k}(u).$$

Proof (of Lemma 3). It will suffice to prove the lemma for the case where u is not one of the knot positions. This is because the functions $B_{i,k}$ and $B_{i+1,k}$ have well-behaved limits at knot positions. In particular, both $\lim_u \rightarrow u_j^+ B_{i,k}(u)$ and $\lim_u \rightarrow u_j^- B_{i,k}(u)$ both exist, and similarly for $B_{i+1,k}$. This is enough to imply that if the lemma holds for all values of u which are not knot positions and if $B_{i,k+1}$ has a first derivative at $u = u_j$, then the derivative of $B_{i,k+1}$ is continuous at $u = u_j$.*

So suppose u is not a knot position. We prove the lemma by induction on k. The base case, $k = 1$, is immediate from the definition of $B_{i,2}$, since $B_{i,k}(u)$ and $B_{i+1,k}$ are either zero or the constant one in a neighborhood of u. It remains to prove the induction step where $k > 1$. By taking the first derivative of the Cos-de Boor definition of the B-spline basis functions, we have

$$B'_{i,k+1}(u) \quad = \quad \frac{B_{i,k}(u)}{u_{i+k} - u_i} + u - u_i u_{i+k} - u_i B'_{i,k}(u)$$

*This uses the following fact from real analysis: If a function f has a derivative at a point u, and if the limit $L = \lim_{v \rightarrow u^+} f'(v)$ exists, then $L = f'(u)$.

$$-\frac{B_{i+1,k}(u)}{u_{i+k+1}-u_{i+1}}+\frac{u_{i+k+1}-u}{u_{i+k+1}-u_{i+1}}B'_{i+1,k}(u).$$

Expanding $B'_{i,k}$ and $B'_{i+1,k}(u)$ with the induction hypothesis yields that $B'_{i,k+1}(u)$ equals

$$\frac{B_{i,k}(u)}{u_{i+k}-u_i}+\frac{u-u_i}{u_{i+k}-u_i}\left(\frac{k-1}{u_{i+k-1}-u_i}B_{i,k-1}(u)-\frac{k-1}{u_{i+k}-u_{i+1}}B_{i+1,k-1}(u)\right)$$

$$-\frac{B_{i+1,k}(u)}{u_{i+k+1}-u_{i+1}}+\frac{u_{i+k+1}-u}{u_{i+k+1}-u_{i+1}}\left(\frac{k-1}{u_{i+k}-u_{i+1}}B_{i,k-1}(u)-\frac{k-1}{u_{i+k+1}-u_{i+2}}B_{i+2,k-1}(u)\right).$$

Using the identity

$$\frac{u_{i+k+1}-u}{u_{i+k+1}-u_{i+1}}-\frac{u-u_i}{u_{i+k}-u_i}=-\frac{u-u_{i+1}}{u_{i+k+1}-u_{i+1}}+\frac{u_{i,k}-u}{u_{i+k}-u_i},$$

which is easily proved noting that the two fractions on the right are each obtained by subtracting the corresponding terms on the left from one, and some algebra, we get that $B'_{i,k+1}(u)$ equals

$$\frac{B_{i,k}(u)}{u_{i+k}-u_i}+\frac{k-1}{u_{i+k}-u_i}\left[\frac{u-u_i}{u_{i+k-1}-u_i}B_{i,k-1}(u)+\frac{u_{i+k}-u}{u_{i+k}-u_{i+1}}B_{i+1,k-1}(u)\right]$$

$$-\frac{B_{i+1,k}(u)}{u_{i+k+1}-u_{i+1}}+\frac{k-1}{u_{i+k+1}-u_{i+1}}\left[\frac{u-u_{i+1}}{u_{i+k}-u_{i+1}}B_{i,k-1}(u)-\frac{u_{i+k+1}-u}{u_{i+k+1}+u_{i+2}}B_{i+2,k-1}(u)\right].$$

Now the Cox-de Boor formulas defining $B_{i,k}$ and $B_{i+1,k}$ are identical to the expressions in the square brackets, and from this, we get

$$B'_{i,k+1}(u) \;=\; \frac{k}{u_{i+k}-u_i}B_{i,k}(u)-\frac{k}{u_{i+k+1}-u_{i+1}}B_{i+1,k}(u).$$

which proves Lemma 3. □

We wish to use Lemma 3 to prove Theorem 2 by induction on k. The next lemma contains what is needed for the base case of the induction proof.

Lemma 4 *Let $k \geq 1$. Suppose that $u_i = u_{i+k-1} < u_{i+k}$. Then*

$$\lim_{u\to u_i^+} B_{i,k}(u) \;=\; 1.$$

Dually, suppose $u_i < u_{i+1} = u_{i+k}$. Then

$$\lim_{u\to u_{i+1}} B_{i,k}(u) \;=\; B_{i,k} \;=\; 1.$$

Proof The lemma is proved by induction on k. It is immediate from the Cox-de Boor definition in the base case $k=1$. For the induction step, the recursive case of the Cox-de Boor definition of $B_{i,k}$ is

$$B_{i,k}(u) \;=\; \frac{u-u_i}{u_{i+k-1}-u_i}B_{i,k-1}(u)+\frac{u_{i+k}-u}{u_{i+k}-u_{i+1}}B_{i+1,k-1}(u).$$

From $u_{i+k-1}-u_i = 0$ and the conventions about $0/0$, the lemma follows immediately from the induction hypothesis applied to $B_{i+1,k-1}(u)$.

The argument for the dual case is completely similar. □

We are now ready to prove Theorem 2. It is certainly enough to prove the following statement: If u_i has multiplicity p, then $B_{j,p+s+1}$ has continuous s-th derivative at $u = u_i$ for all $j \geq 0$ and $s \geq 0$. This statement is proven by holding u_i and its multiplicity fixed and using induction on s. The base case, $s = 0$, is a direct consquence of Lemma 4. The induction step is a simple application of Lemma 3. Q.E.D. Theorem 2

1.4.3 Knot insertion

An important tool for the practical interactive use of Bézier curves is the technique of knot insertion, which allows one to add a new knot to a B-spline curve without changing the curve, or its degree. For instance, when editting a B-spline curve with a CAD program, one may wish to insert additional knots in order to be able to make further adjustments to a curve. Adding the additional knots in the area of the curve that needs adjustment allows more flexibility in editting the curve. Knot insertion also allows the multiplicity of a knot to be increased, which affects the smoothness of the curve at that point. A second use of knot insertion is to convert a B-spline curve into a series of Bézier curves, as we shall see in the next section.

There are commonly used two methods for knot insertion. The Böhm method allows a single knot a time to be inserted into a curve [?, ?], and the Oslo method allows multiple knots to be inserted at once [?, ?]. We discuss only the Böhm method here: of course, multiple knots may be inserted by iterating the Böhm method.

Suppose $Q(u)$ is an degree k B-spline curve, defined with knot vector $[u_0, \ldots, u_m + k + 1]$ and with control points $P_0, \ldots, P_m$. We wish to insert a new knot position u' where $u_s \leq u' \leq u_{s+1}$, and then choose new control points so that the curve $Q(u)$ remains unchanged.

The new knot vector is denoted $[u'_0, \ldots, u'_{m+k+2}]$ where, of course,

$$u'_i = \begin{cases} u_i & \text{if } i \leq s \\ u' & \text{if } i = s+1 \\ u'_{i-1} & \text{if } i > s+1. \end{cases}$$

The method of choose the new control points is less obvious, since we must be sure not to change the curve. The Böhm algorithm gives the following definition of the control points:

$$P'_i = \begin{cases} P_i & \text{if } i \leq s-k+1 \\ \dfrac{u_{i+k} - u'}{u_{i+k} - u_i} P_i + \dfrac{u' - u_i}{u_{i+k} - u_i} P_{i+1} & \text{if } s-k+1 < i \leq s \\ P_{i-1} & \text{if } i > s \end{cases}$$

Theorem 5 *Suppose $k \geq 1$ and let $Q'(u)$ be the degree k B-spline curve defined with the knot vector $[u'_0, \ldots, u'_{m+k+1}]$ and control points $P'_0, \ldots P'_m + 1$. Then $Q(u) = Q'(u)$ for all u.*

Proof XXXX PROOF TO BE WRITTEN

1.4.4 Bézier and B-spline curves

We now discuss methods for translating Bézier curves between B-spline curves. These methods are degree perserving in that they will preserve the a degree k Bézier curve into a degree k B-spline, and vice-versa. Of course, there is a bit of a mismatch: a Bézier curve consists of a single degree k curve, specified by $k+1$ control points. A B-spline curve consists of series of pieces, each piece a degree k polynomial. Accordingly, the translation between B-spline curves and Bézier curves will transform a series of degree k pieces that join together to make a single curve. Such a series of curve pieces can be viewed as either a single B-spline curve or as a collection of Bézier curves.

First, we consider the problem of converting a single Bézier curve into a B-spline curve. Suppose we have a degree 3 Bézier curve $\mathbf{Q}(u)$ defined with control points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$, defined over the range $0 \le u \le 1$. In order to construct a definition of this curve as a B-spline curve with the same control points we let $[0,0,0,0,1,1,1,1]$ be the knot vector and keep control points as $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$. It can be verified by direct computation that the B-spline curve is in fact the same curve $Q(u)$ as the Bézier curve. In fact, we have the following general theorem.

Theorem 6 *Let $k \ge 1$ and $\mathbf{Q}(u)$ be a degree k Bézier curve defined by control points $\mathbf{p}_0, \ldots, \mathbf{p}_{k-1}$. Then $\mathbf{Q}(u)$ is identical to the degree k B-spline curve defined with the same control points over the knot vector consisting of the knot 0 with multiplicity $k+1$ followed by the knot 1 also with multiplicity $k+1$.*

In order to prove this theorem, let $B_{i,k+1}$ be the basis functions for the B-spline with the knot vector $[0, \cdots, 0, 1, \cdots, 1]$. Then we claim that

$$B_{i,k+1} = \binom{k}{i}(1-u)^i u^{k-1}. \tag{1.2}$$

The righthand side of this equation is just the same as the Bernstein polynomials used to define Bézier curves, so the theorem follows immediately from equation (1.2). Equation (1.2) is easy to prove by induction on k, and we leave the proof to the reader. □

The most useful cases of the previous theorem are when $k=2$ and $k=3$. The $k=2$ case is frequently used for defining conic sections, including circles via Bézier curves or B-splines. In the $k=2$ case, a degree 2 Bézier curve with the three control $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ is equivalent to the degree 2 B-spline curve with knot vector $[0,0,0,1,1,1]$ and with the same three control points.

Often one wants to combine two or more Bézier curves into a single B-spline curve. For instance, suppose one has a degree 2 Bézier curves $Q_0(u)$ and $Q_1(u)$ defined with control points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{p}'_0, \mathbf{p}'_1, \mathbf{p}'_2$. By Theorem ??, these two curves are equivalent to the degree 2 B-spline curve with knot vector $[0,0,0,1,1,1,2,2,2]$ and with the eight control points $\mathbf{p}_0, \ldots, \mathbf{p}'_3$. However, usually the two Bézier curves form a single continuous curve, i.e., $\mathbf{p}_3 = \mathbf{p}'_0$. In this case, the two Bézier curves are the same as the B-spline curve with knot vector $[0,0,0,1,1,2,2,2]$ and with control points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}'_1, \mathbf{p}'_2$. Note t XX TO DO

Figure 1.3: Two ways to define circular arcs with rational Bézier curves without control points at infinity.

that one knot position and the duplicate control point have been omitted. The fact that this construction works is proved by the calculation in the two next exercises.

Exercise: Calculate the degree 2 blending functions for the knot vector $[0,0,0,1,1,2,2,2]$. Show that the results are the degree 2 Bernstein polynomials and the degree 2 Bernstein polynomials translated to the interval $[1,2]$.

Exercise: Let $Q(u)$, $0 \le u \le 2$, be the B-spline curve defined with knot vector $[0,0,0,1,1,2,2,2]$ and control points $P_0,\ldots,P_4$. Let $Q_1(u)$, $0 \le u \le 1$, be defined with knot vector $[0,0,0,1,1,1]$ and control points P_0, P_1, P_2. Let $Q_2(u)$, $1 \le u \le 2$, be defined with knot vector $[1,1,1,2,2,2]$ and control points P_2, P_3, P_4. Prove that the curve $Q(u)$ is the union of the curves $Q_1(u)$ and $Q_2(u)$.

The construction in these exercises can be generalized in several ways. First, if one has three degree 2 Bézier curves which form a single continuous curve, then they are equivalent to a degree 2 B-spline curve with knot vector $[0,0,0,1,1,2,2,3,3,3]$. This generalizes to allow expressing any continuous curve which consists of any number of Bézier as a single B-spline curve. Second, the construction generalizes to other degrees: for instance, a continuous curve which consists of two degree 3 Bézier curves is the same as the degree 3 B-spline curve that has knot vector $[0,0,0,0,1,1,1,2,2,2,2]$ and has the same seven points as its control points. We will leave the proofs of these statements to the reader.

Any B-spline curve can be split into a sequence of Bézier curves. XXX

(a) General principles. XXX

(b) Knot insertion. XXX

1.4.5 Rational splines and NURBS

XXX

Preserved under perspective transformations too.

Rational curves - equivalent to weighting for points not at infinity.

Conic sections, especially circles.

Exercise: Prove that the following rational degree 2 Bézier curve defines the $90°$ arc of the unit circle in the plane from the point $(1,0)$ to $(0,1)$. The knot vector is $[0,0,0,1,1,1]$ and the control points are $P_0 = (1,0,1)$, $P_1 = (1/\sqrt{2}, 1/\sqrt{2}, 1/\sqrt{2})$ and $P_2 = (0,1,1)$. (See figure 1.4.5 for this and the next exercise.)

XX TODO

Figure 1.4: Defining the Catmull-Rom spline segment from $\mathbf{p}_i$ to $\mathbf{p}_{i+1}$.

Exercise: Prove that the following rational degree 2 Bézier curve defines the $120°$ arc of the unit circle in the plane from $(\sqrt{3}/2, 1/2)$ to $(-\sqrt{3}/2, 1/2)$. The knot vector is $[0,0,0,1,1,1]$ and the control points are $P_0 = (\sqrt{3}/2, 1/2, 1)$, $P_1 = (0, 2, 1/2)$ and $P_2 = (-\sqrt{3}/2, 1/2, 1)$. (See figure 1.4.5.)

Exercise: Prove that there is no non-rational degree 3 Bézier or B-spline curve which traces out a non-trivial part of a circle.

Exercise: Prove that there is no non-rational Bézier or B-spline curve of any degree which traces out a non-trivial part of a circle.

1.5 Interpolating with spline curves

Frequently one wishes to define a smooth curve that interpolate (i.e., 'pass though') a given set of points. However, the B-spline curves that have been defined so far are not particularly convenient for this purpose: they have been defined from control points, and the control points merely influence the curve but are usually not interpolated. When control points are interpolated it is generally because of repeated knot values and then curve loses its good smoothness properties and may have discontinuous first derivatives.

However, there are a number of good methods to define interpolating spline curves. Suppose that we have a set of interpolation points $\mathbf{p}_0, \ldots, \mathbf{p}_m$ and a set of knot values $u_0, \ldots, u_m$. The problem is to define a piecewise (degree 3) polynomial curve so that $\mathbf{Q}(u_i) = \mathbf{p}_i$ for all i.

We describe two general solutions to this problem. The first is Catmull-Rom splines which use a Bézier curve construction to define degree 3 splines. The second is based on solving a system of linear equations to find control points that define the desired curve. The linear equations are 'upper diagonal' and thus are easily solved. Both approaches benefit from the use of chord-length parameterization techniques.

Catmull-Rom Splines Catmull-Rom splines are specified by a list of control points $\mathbf{p}_0, \ldots \mathbf{p}_m$ and are piece-wise degree 3 polynomial curves that interpolate all the points except the endpoints $\mathbf{p}_0$ and $\mathbf{p}_m$. They are defined by making an estimate for the slope of the curve passing through $\mathbf{p}_i$, using this to define additional control points for Bézier curves so that the i-th Bézier curve has beginning point $\mathbf{p}_i$ and ending point $\mathbf{p}_{i+1}$.

Figure 1.5 illustrates the definition of a Catmull-Rom spline segment. Let $\mathbf{v}_i = \frac{1}{2}(\mathbf{p}_{i+1} - \mathbf{p}_{i-1})$ and define $\mathbf{p}_i^+ = \mathbf{p}_i + \frac{1}{3}\mathbf{v}_i$ and $\mathbf{p}_i^- = \mathbf{p}_i - \frac{1}{3}\mathbf{v}_i$. Then let $\mathbf{Q}_i(u)$ be the Bézier curve, translated to have domain $i \le u \le i+1$, which is

XX TODO

Figure 1.5: Examples of uniform and chord-length Catmull-Rom splines

defined with control points $\mathbf{p}_i, \mathbf{p}_i^+, \mathbf{p}_{i+1}^-, \mathbf{p}_{i+1}$. Define $\mathbf{Q}(u)$ by piecing together these curves, namely, $\mathbf{Q}(u) = \mathbf{Q}_i(u)$ if $i \le u \le i+1$.

Since Bézier curves interpolate their end points, the curve $\mathbf{Q}$ is continuous and $Q(i) = \mathbf{p}_i$ for all integer i such that $1 \le i \le m-1$. In addition, $\mathbf{Q}$ has continuous second derivatives, with $\mathbf{Q}'(i) = (\mathbf{p}_{i+1} - \mathbf{p}_{i-1})/2$.

Figure 1.5, parts (a) and (b), shows two examples of Catmull-Rom splines.

The Figure 1.5(b) shows that bad effects can occur when the interpolated points are not more-or-less equally spaced. One way to fix this problem is to use chord-length parameterization: what this means is that we assign the knot positions so that $u_{i+1} - u_i$ is equal to $||\mathbf{p}_{i+1} - \mathbf{p}_i||$. The idea is that the arclength of the curve between $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ will be approximately proportional to the distance from $\mathbf{p}_i$ to $\mathbf{p}_{i+1}$ and therefore approximately proportional to $u_{i+1} - u_i$. Therefore, if one views the u-parameter as time, then, as u varies, the curve will traversed at approximately a constant rate of speed.

Of course, in order to use chord-length parameterization, we need to modify the formalization of Catmull-Rom splines to allow for non-uniform knot positions. For this purpose, we assume that all knot positions are distinct and define

$$\mathbf{v}_{i+\frac{1}{2}} = \frac{\mathbf{p}_{i+1} - \mathbf{p}_i}{u_{i+1} - u_i}.$$

The idea is that $\mathbf{v}_{i+\frac{1}{2}}$ is the mean velocity at which the interpolating spline is traversed from $\mathbf{p}_i$ to $\mathbf{p}_{i+1}$. Of course, if we have defined the knot positions using a chord-length interpolation, then the velocities $\mathbf{v}_{i+\frac{1}{2}}$ will be unit vectors. Then we define a further velocity

$$\mathbf{v}_i = \frac{(u_{i+1} - u_i)\mathbf{v}_{i-\frac{1}{2}} + (u_i - u_{i-1})\mathbf{v}_{i+\frac{1}{2}}}{u_{i+1} - u_{i-1}},$$

which is a weighted average of the two velocities of the curve segments just before and and just after the interpolated point $\mathbf{p}_i$. Finally define

$$\mathbf{p}_i^- = \mathbf{p}_i - \frac{1}{3}\mathbf{v}_i \qquad \text{and} \qquad \mathbf{p}_i^+ = \mathbf{p}_i + \frac{1}{3}\mathbf{v}_i.$$

These points are then used to define Bézier curves exactly as in the manner used for the uniform Catmull-Rom curves.

An example of chord-length parameterization is given in Figure 1.5(c): clearly this represents a qualitative improvement over the uniform Catmull-Rom interpolating curve shown in Figure ??(c).

Exercise: Investigate the chord-length parameterization Catmull-Rom curve from $\mathbf{p}_1$ to $\mathbf{p}_2$ when $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ are colinear. What is the velocity at $\mathbf{p}_1$? Consider separately the cases where $\mathbf{p}_1$ is and is not between $\mathbf{p}_0$ and $\mathbf{p}_2$.

Interpolating with B-splines XXX

1.5.1 B-splines and NURBS in OpenGL

OpenGL provides routines for drawing (non-uniform) B-spline surfaces in the `glu` library. By specifying the control points in homogeneous coordinates, this includes the ability to render NURBS surfaces. The B-spline routines include `gluNewNurbsRenderer` and `gluDeleteNurbsRenderer` to allocate and deallocate a B-spline renderer: these routines are misnamed as they can also be used to render non-rational B-splines. The routines `gluBeginSurface()` and `gluEndSurface()` are used to bracket one or more calls to `gluNurbsSurface()`. The latter routine allows specification of an array of knots and control points. Since it renders a surface, it uses two knot arrays and a two dimensional array of control points. A routine `gluNurbsProperty()` allows you to control the level of detail at which the B-spline surface is rendered.

These B-spline routines work similarly to, but somewhat less complicatedly than, the OpenGL routines for Bézier curves discussed in section 1.3. The interested reader should refer to the OpenGL documentation for more details on how to use the B-spline rendering routines.