A Curve Tutorial for Introductory Computer Graphics

Michael Gleicher
Department of Computer Sciences
University of Wisconsin, Madison

October 7, 2003

Note to 559 Students: These notes were put together rather hastily. Some sections are still missing. However, the ones that are important for this class are here (or in *Fundamentals of Computer Graphics*). Sometimes I refer to the old 559 textbook *Computer Graphics* by Hearn and Baker. Be sure to check for errata - I hope I didn't make many mistakes, but realistically, ...

1 Curves

Mathematically, a curve is:

> A continuous map from a one-dimensional space to an n-dimensional space.

Intuitively, think of a curve as something you can draw with a (thin) pen on a piece of paper. You cannot create filled regions, but you can create the outlines of things.

A curve is an infinitely large set of points. The points in a curve have a property that any point has 2 neighbors, except for a small number of points that have one neighbor (these are the endpoints). Some curves have no endpoints, either because they are infinite (like a line) or they are *closed* (loop around and connect to themselves).

The problem that we need to address is how to describe a curve - to give "names" or representations to all curves so that we can represent them on a computer. For some curves, the problem of naming them is easy since they have known shapes: line segments, circles, elliptical arcs, etc. A general curve that doesn't not have a "named" shape is sometimes called a *free-form* curve.

Because a free-form curve can take on just about any shape, they are much harder to describe.

There are three main ways to describe curves mathematically:

Implicit curve representations define the set of points on a curve by giving a procedure that can test to see if a point in on the curve. Usually, an implicit curve is defined by an *implicit function* of the form:

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

so that the curve is the set of points for which this equation is true. Note that the implicit function is a scalar function (it returns a single real number).

Explicit or Parametric curve descriptions provide a mapping from a *free parameter* to the set of points on the curve. That is, this free parameter (a single number) provides an index to the points on the curve. The parametric form of a curve defines a function that assigns positions to values of the free parameter. Intuitively, if you think of a curve as something you can draw with a pen on a piece of paper, the free parameter is time, ranging from the time that we began drawing the curve to the time that we finish. The *parametric function* of this curve would tells us where the pen was at any instant in time:

$$x, y = \mathbf{f}(t).$$

Note that the parametric function is a vector valued function that returns a vector (a point position). Since we are working in 2D, these will be 2-vectors, but in 3D they would be 3 vectors.

Generative or Procedural curve descriptions provide procedures that can generate the points on the curve that do not fall into the first two categories. Examples of generative curve descriptions include subdivision schemes and fractals.

Some curves can be easily represented in both explicit and implicit forms. For example, a circle with its center at the origin and radius of 1 can be written in implicit form as:

$$f(x, y) = x^2 + y^2 = 0,$$

or in parametric form as:

$$x, y = \mathbf{f}(u) = \cos u, \sin u.$$

Different representations of curves have advantages and disadvantages. For example, parametric curves are much easier to draw because we can sample the free parameter. Generally, parametric forms are the most commonly used in computer graphics since they are easier to work with. Our focus will be on parametric curves.

1.1 Natural and Arc-Length Parameterizations

A parametric curve is a mapping from the values of the free parameter to positions. To be more precise, a parametric curve is a mapping from a *range* or interval of the parameters. Without loss of generality, we can have the range of the parameter be from 0 to 1. It is often useful to have the parameter in this range. When the free parameter ranges over the unit interval, we often denote it as u.

If we consider the parametric curve to be a line drawn with a pen, we can define the beginning of time (when $u = 0$) to be when the pen is first set down on the paper, and the unit of time to be the amount of time it takes to draw the curve ($u = 1$ is the end of the curve). To use a different unit for time (for example one second), we scale or shift time into the unit interval. The parametric curve is defined by a function that maps time (in these unit coordinates) to positions. Basically, the definition of the curve is a function that can answer the question "where is the pen at time u?"

The existence of the free parameter (or the element of time) adds an invisible, potentially unknown element to our curves. When we look at the curve after it is drawn, we don't necessarily know the timing. The pen might have moved at a constant speed over the entire time interval, or it might

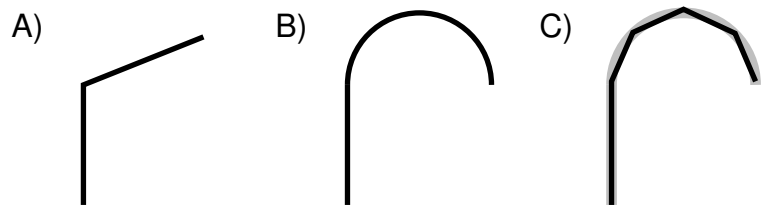

Figure 1: A: a curve that can be easily represented as two lines; B: a curve that can be easily represented as a line and a circular arc; C: approximating curve B with 5 line segments

have started slowly and sped up. While simply using the free parameter (or time) is a simple and natural way to index the points along the curve (and is, therefore, sometimes referred to as the *natural parameterization*), the fact that we don't know about how the speed changes along the curve can make working with the curve difficult. For example, while $u = .5$ is halfway through the parameter space, it may not be half-way along the curve if the motion of the pen starts slowly and speeds up at the end.

If we know that the pen moves at a constant velocity, then the values of the free parameters have more meaning. Halfway through parameter space is half-way along the curve. Rather than measuring time, the parameter can be thought to measure length along the curve. Such parameterizations are called *arc length* parameterizations because they define curves by functions that map from the distance along the curve (known as the arc length) to positions.

Computing the length along a curve can be tricky. In general, it is defined by the integral of the magnitude of the derivative (intuitively, the magnitude of the derivative is the velocity of the pen as it moves along the curve). So, given u, you can compute s (the arc length distance along the curve from the point $\mathbf{f}(0)$ to the point $\mathbf{f}(u)$) as:

$$s = \int_0^u \left|\mathbf{f}'(v)\right|^2 dv \tag{1}$$

where $\mathbf{f}(v)$ is the curve function with a natural parameterization.

Using the arc-length parameterization requires being able to solve Equation 1 for u given s. For many of the kinds of curves we examine, it cannot be done in a closed-form (simple) manner, and must be done numerically. In this document, we work exclusively with natural parameterizations.

Generally, we use the variable u to denote natural free parameters that range over the unit interval, s to denote arc-length free parameters, and t to represent parameters that aren't one of the other two.

1.2 Piecewise Parametric Representations

For some curves, defining a parametric function that represents their shape is easy. For example, lines, circles, and ellipses all have simple functions that define the points they contain in terms of a parameter. For many curves, finding a function that describes their shape can be hard. The main strategy that we use to create complex curves is divide-and-conquer: we break the curve into a number of simpler smaller pieces, each of which has a simple description.

For example, consider the curve in Figures 1. The first two curves are easily described in terms of two pieces. In the case of curve B, we need two different kinds of pieces: a line segment and a circle.

To create a parametric representation of a compound curve (like B), we need to have our parametric function switch between the functions that represent the pieces. If we always define our parametric functions over the range $0 \leq u \leq 1$, then curve A or B might be defined as:

$$\mathbf{f}(u) = \begin{array}{ll} \mathbf{f_1}(2 * u) & \text{if } u \leq .5 \\ \mathbf{f_2}(2 * u - 1) & \text{if } u > .5, \end{array} \tag{2}$$

where $\mathbf{f_1}$ is the parametric function of the first piece, and $\mathbf{f_2}$ is the parametric function of the second piece, and both of these functions are defined over the unit interval.

We need to be careful in defining the functions $\mathbf{f_1}$ and $\mathbf{f_2}$ to make sure that the pieces of the curve fit together. If $\mathbf{f_1}(1) \neq \mathbf{f_2}(0)$, then our curve pieces will not connect, and will not form a single continuous curve.

To represent curve B in Figure 1 well, we needed to use two different types of pieces: a line segment and a circular arc. For simplicity's sake, we may prefer to use a single type of pieces. If we try to recreate curve B with only one type of piece (line segments), we cannot exactly recreate the curve (unless we use an infinite number of pieces). While the new curve made of line segments (as in Figure 1 C) may not be exactly the same shape as B, it might be close enough for our uses. In such a case, we might prefer the simplicity of using the simpler line segment pieces to having a curve that more accurately represents the shape.

Also, notice that as we use an increasing number of pieces, we can get a better approximation. In the limit (using an infinite number of pieces) we can exactly represent the original shape.

One advantage to using a piecewise representation is that it allows us to make a tradeoff between:

1. how well our represented curve approximates the real shape we are trying to represent;
2. how complicated the pieces that we use are;
3. how many pieces we use.

So if we're trying to represent a complicated shape, we might decide that a crude approximation is acceptable, and use a small number of simple pieces. To improve the approximation we can choose between using more pieces and using more complicated pieces.

In computer graphics practice, we tend to prefer using relatively simple curve pieces (either line segments or cubic polynomial segments).

2 Curve Properties

To describe a curve, we need to give some facts about its properties. For "named" curves, the properties are usually specific to the type of curve. For example, to describe a circle, we might provide its radius and the position of its center. For an ellipse, we might also provide the orientation of its major axis, and the ratio of the lengths of the axes. For free form-curves however, we need to have a more general set of properties to describe individual curves.

Some properties of curves describe only a single place on the curve, while other properties require knowledge of the whole curve. For an intuition of the difference, imagine that the curve is a train

track. If you are standing on the track on a foggy day you can tell that the track it is straight or curving and whether or not you at at an end point. These are *local* properties. You cannot tell whether or not the track is a closed curve, or crosses itself, or how long it is. We call the second kind a *global* property.

The study of local properties of geometric objects (curves and surfaces) is known as *differential geometry.* Technically, to be a differential property there are some mathematical restrictions about the properties (roughly speaking, in the train track analogy, you would not be able to have a GPS or a compass). Rather than worry about this distinction, I will use the term *local* property rather than differential property.

Local properties are important tools for describing curves because they do not require knowledge about the whole curve. Local properties include:

- continuity
- position at a specific place on the curve
- direction at a specific place on the curve
- curvature (and other derivatives).

Some examples of useful global properties include:

- whether the curve is open or closed
- whether the curve ever passes through a particular point, or goes through a particular region
- whether the curve ever points in a particular direction

2.1 Continuity

It will be very important to understand the local properties of a curve where two parametric pieces come together. If a curve is defined using an equation like Equation 2, then we need to be careful about how the pieces are defined. If $\mathbf{f_1}(1) \neq \mathbf{f_2}(0)$, then the curve will be "broken" - we would not be able to draw the curve in a continuous stroke of a pen. We call the condition that the curve pieces fit together *continuity* conditions because if they hold, the curve can be drawn as a continuous piece.

In addition to the positions, we can also check that the derivatives of the pieces match correctly. If $\mathbf{f_1}'(1) \neq \mathbf{f_2}'(0)$, then the combined curve will have an abrupt change in its first derivative at the switching point. The first derivative will not be continuous. In general, we say that a curve is $C(n)$ continuous if all of its derivatives up to n match across pieces. We denote the position itself as the 0^{th} derivative, so that the $C(0)$ continuity condition means that the positions of the curve are continuous, and $C(1)$ continuity means that positions and first derivatives are continuous.

An illustration of some continuity conditions is shown in Figure 2. A discontinuity in the first derivative (the curve is $C(0)$ but not $C(1)$) is usually noticeable because it leads to a sharp corner. A discontinuity in the second derivative is sometimes visually noticeable. Discontinuities in higher derivatives might matter, depending on the application. For example, if the curve is a motion, an

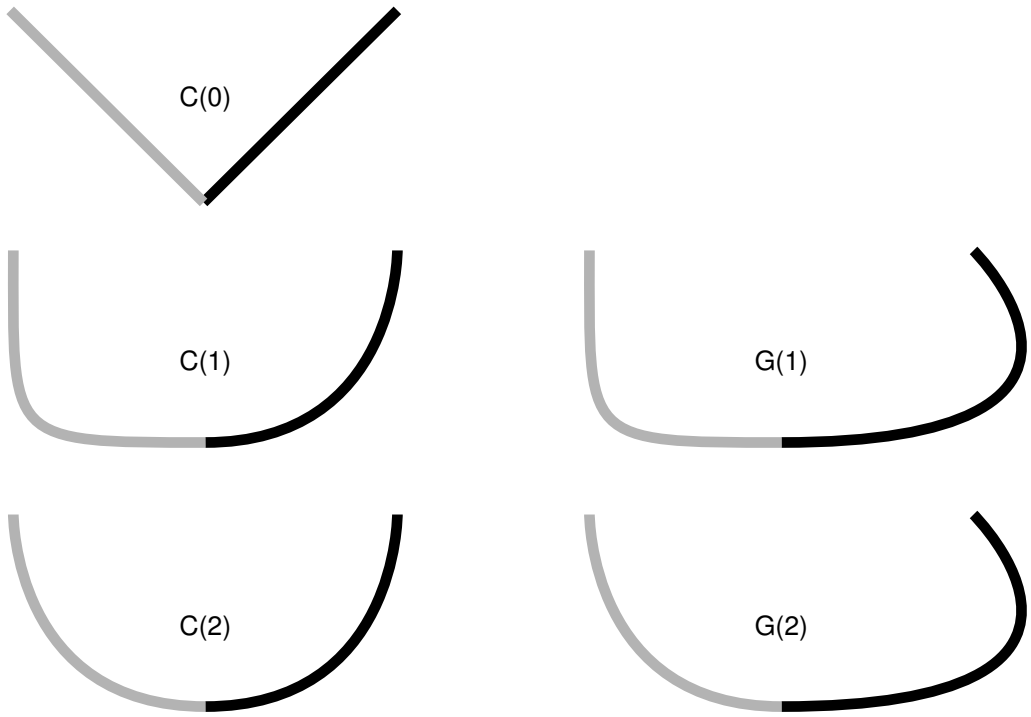

Figure 2: An illustration of various types of continuity between two curve segments

.

abrupt change in the 2nd derivative is noticeable, so 3rd derivative continuity is often useful. If the curve is going to have a fluid flowing over it (for example if it is the shape for an airplane wing or boat hull), a discontinuity in the 4th or 5th derivative might cause turbulence.

We have been considering derivative continuity with natural (rather than arc-length) parameterizations. Curves that are $C(1)$ continuous in their natural parameterizations may not be continuous in the derivatives of their arc length parameterizations. Creating continuity conditions for arc-length parameterizations can be very difficult.

Because the "speed" of the parameterization might be different, even if the derivatives match, we define a different type of continuity that ignores the speed. We define *geometric continuity* to be the condition where the derivative of the end of one segment differs only in magnitude from the beginning of the next. That is, where the $C(1)$ condition requires:

$$\mathbf{f_1}'(1) = \mathbf{f_2}'(0),$$

the $G(1)$ continuity condition requires:

$$\mathbf{f_1}'(1) = k\ \mathbf{f_2}'(0)$$

for some value k. Geometric continuity is less restrictive than parametric continuity (a $C(n)$ curve is necessarily $G(n)$, but not vice versa).

3 Parametric Polynomial Pieces

The most popular representations for curves in computer graphics are piecewise parametric functions where each piece is a polynomial of the free parameter. Piecewise linear representations (a

line segment is a polynomial of order 1) are a special case of this. In this section, we look over the mathematics of the individual polynomial pieces. In the next section, we discuss how to put pieces of polynomials together.

To introduce the concepts of polynomial parametric curves, we will discuss line segments. In practice, line segments are so simple that the mathematical derivations will seem extraneous. However, by understanding this simple case, things will be easier when we move on to more complicated polynomials.

3.1 A Line Segment

Let's return to a line segment that connects point $\mathbf{p_0}$ to $\mathbf{p_1}$. We could write the parametric function over the unit domain for this line segment as:

$$\mathbf{f}(u) = (1-u)\mathbf{p_0} + u\mathbf{p_1}. \tag{3}$$

Since we're treating each element of the point vector independently, we could have written this as separate equations in each dimension,

$$\begin{aligned} f_x(u) = &\quad (1-u)p_{0_x} + up_{1_x} \\ f_y(u) = &\quad (1-u)p_{0_y} + up_{1_y}, \end{aligned}$$

but from now on we'll stick to vector notation since its cleaner. We will call the vector of control parameters $\mathbf{p}$ the *control points*, and each element of $\mathbf{p}$ a *control point*.

While describing a line segment by the positions of its endpoints is obvious and usually convenient, there are other ways to describe a line segment. For example:

1. the position of the center of the line segment, the orientation, and the length;
2. the position of one endpoint and the position of the second point relative to the first;
3. the position of the middle of the line segment and one end.

It should be fairly obvious that given one kind of a description of a line segment, we can switch to another.

A different way to describe a line segment is:

$$\mathbf{f}(u) = \mathbf{a_0} + u\mathbf{a_1}. \tag{4}$$

Any line segment can be represented either by specifying $\mathbf{a_0}$ and $\mathbf{a_1}$, or the endpoints ($\mathbf{p_0}$ and $\mathbf{p_1}$). It is usually more convenient to specify the endpoints, because we can compute the other parameters from the endpoints.

The description of the line using $\mathbf{a_0}$ and $\mathbf{a_1}$ is convenient mathematically because it has a form that extends easily to more complicated curves. More generally, we might write Equations 4 as:

$$\mathbf{f}(u) = \sum_{i=0}^{n} u^i \mathbf{a_i}, \tag{5}$$

where n is the degree of the polynomial (1 for a line). To write this equation in a vector form, we define a vector $\mathbf{u}$ that is a vector of the powers of u:

$$\mathbf{u} = \begin{bmatrix} 1 & u & u^2 & u^3 & \dots & u^n \end{bmatrix}$$

so that Equation 5 can be written as:

$$\mathbf{f}(u) = \mathbf{u}\ \mathbf{a}. \tag{6}$$

This vector notation will make transforming between different forms of the curves easier.

Equations 5 or 6 describe a curve segment by the set of polynomial coefficients for the simple form of the polynomial. We call such a representation the *canonical* form. I will denote the parameters of the canonical form by $\mathbf{a}$.

While it is mathematically simple, canonical form is not always the most convenient way to specify curves. For example, we might prefer to specify a line segment by the positions of its endpoints. If we want to define $\mathbf{p_0}$ to be the beginning of the segment (where the segment is when $u = 0$) and $\mathbf{p_1}$ to be the end of the line segment (where the line segment is at $u = 1$), we can write:

$$\begin{array}{lll} \mathbf{p0} & = \mathbf{f}(0) & = [1\ 0] \quad [\mathbf{a_0}\ \mathbf{a_1}] \\ \mathbf{p1} & = \mathbf{f}(1) & = [1\ 1] \quad [\mathbf{a_0}\ \mathbf{a_1}]\,. \end{array} \tag{7}$$

We can solve these equations for $\mathbf{a_0}$ and $\mathbf{a_1}$:

$$\begin{aligned} \mathbf{a_0} &= \mathbf{p_0} \\ \mathbf{a_1} &= \mathbf{p_1} - \mathbf{p_0}. \end{aligned}$$

3.1.1 Matrix Form for Polynomials

While this first example was easy easy enough to solve, for more complicated examples it will be easier to write Equation 7 in the form:

$$\begin{bmatrix} \mathbf{p_0} \\ \mathbf{p_1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a_0} \\ \mathbf{a_1} \end{bmatrix}.$$

Or, if we call this *constraint matrix* $\mathbf{C}$, we can write

$$\mathbf{p} = \mathbf{C}\ \mathbf{a}, \tag{8}$$

although, this is being a little sloppy with notation[1]. If having vectors of points bothers you, you can consider each dimension independently (so that $\mathbf{p}$ is $[p_{0_x}\ p_{1_x}]$ or $[p_{0_y}\ p_{1_y}]$) and $\mathbf{a}$ is handled correspondingly).

We can solve Equation 8 for $\mathbf{a}$ by finding the inverse of $\mathbf{C}$. This matrix we will call the *basis* matrix, and we will denote it by $\mathbf{B}$. The basis matrix is very handy since it tells us how to convert between the convenient parameters $\mathbf{p}$ and the canonical form $\mathbf{a}$, and therefore gives us an easy way to evaluate the curve:

$$\mathbf{f}(u) = \mathbf{u}\ \mathbf{B}\ \mathbf{p}.$$

We can find a basis matrix for whatever form of the curves that we want, providing that there are no non-linearities in the definition of the parameters.

[1] I am being very sloppy about whether vectors are row vectors or column vectors. In general, the sense of a vector should be obvious from its context, and I'll skip all of the transpose symbols for vectors.

Another Example: Suppose we want to parameterize the line segment so that $\mathbf{p_0}$ is the halfway point ($u = .5$), and $\mathbf{p_1}$ is the ending point ($u = 1$). To derive the basis matrix for this parameterization:

$$\begin{aligned}(p_0) = \quad \mathbf{f}(.5) = \quad & 1\ \mathbf{a_0} + .5\ \mathbf{a_1} \\ (p_1) = \quad \mathbf{f}(1) = \quad & 1\ \mathbf{a_0} + 1\ \mathbf{a_1}\end{aligned}$$

So

$$\mathbf{C} = \begin{bmatrix} 1 & .5 \\ 1 & 1 \end{bmatrix}$$

and, therefore

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 2 & -1 \\ -2 & 2 \end{bmatrix}.$$

3.2 Beyond Line Segments

Line segments are simple enough that the effort of finding a basis matrix is silly. However, it was good practice for curves of higher degree. First, let's consider quadratics (curves of degree 2). The advantage of the canonical form (Equation 5) is that it works for these more complicated curves, just by letting n be a larger number.

A quadratic (a degree 2 polynomial) has 3 coefficients, $\mathbf{a_0}$, $\mathbf{a_1}$, and $\mathbf{a_2}$. These coefficients are not convenient for describing the shape of the curve. However, we can use the same basis matrix method to devise more convenient parameters. If we know the value of u, Equation 5 becomes a linear equation in the parameters,and the linear algebra from the last section still works.

Suppose that we wanted to describe our curves by the position of the beginning ($u = 0$), middle[2] ($u = .5$), and end ($u = 1$). Filling the appropriate values into Equation 5:

$$\begin{array}{lllllll} \mathbf{p_0} & = f(0) & = & \mathbf{a_0} + & 0^1 & \mathbf{a_1} & + 0^2 \quad \mathbf{a_2} \\ \mathbf{p_1} & = f(.5) & = & \mathbf{a_0} + & .5^1 & \mathbf{a_1} & + .5^2 \quad \mathbf{a_2} \\ \mathbf{p_2} & = f(1) & = & \mathbf{a_0} + & 1^1 & \mathbf{a_1} & + 1^2 \quad \mathbf{a_2}. \end{array}$$

So the constraint matrix is:

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & .5 & .25 \\ 1 & 1 & 1 \end{bmatrix},$$

and the basis matrix is:

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -3 & 4 & -1 \\ 2 & -4 & 2 \end{bmatrix}$$

There is one additional type of constraint (or parameter) that is sometimes convenient to specify: the derivative of the curve (with respect to its free parameter) at a particular value. Intuitively, the derivatives tell us how the curve is changing, so that the first derivative tells us what direction the curve is going, the second derivative tells us how quickly the curve is changing direction, etc. We will see examples of why it is useful to specify derivatives later.

[2]Notice that this is the middle of the parameter space, which might not be the middle of the curve itself.

For the quadratic,

$$\mathbf{f}(u) = \mathbf{a_0} + \mathbf{a_1}u + \mathbf{a_2}u^2,$$

the derivatives are simple:

$$\mathbf{f}'(u) = \frac{df}{du} = \mathbf{a_1} + 2\mathbf{a_2}u,$$

and

$$\mathbf{f}''(u) = \frac{d^2f}{du^2} = \frac{df'}{du} = 2\mathbf{a_2}.$$

Or, more generally,

$$\begin{aligned} \mathbf{f}'(u) = &\quad \textstyle\sum_{i=1}^{n} iu^{i-1}\mathbf{a_i}, \\ \mathbf{f}''(u) = &\quad \textstyle\sum_{i=2}^{n} i(i-1)u^{i-2}\mathbf{a_i}. \end{aligned}$$

For example, consider a case where we want to specify a quadratic curve segment by the position, first, and second derivative at its middle ($u = .5$).

$$\begin{array}{lllllllll} \mathbf{p_0} & = f(.5) & = & \mathbf{a_0}+ & .5^1 & \mathbf{a_1}+ & & .5^2 & \mathbf{a_2} \\ \mathbf{p_1} & = f'(.5) & = & & & \mathbf{a_1}+ & 2 & (.5) & \mathbf{a_2} \\ \mathbf{p_2} & = f''(.5) & = & & & & 2 & & \mathbf{a_2}. \end{array}$$

So the constraint matrix is:

$$\mathbf{C} = \begin{bmatrix} 1 & .5 & .25 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix},$$

and the basis matrix is:

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & -.5 & .125 \\ 0 & 1 & -.5 \\ 0 & 0 & .5 \end{bmatrix}$$

3.3 Basis Matrices for Cubics

Cubic polynomials are popular in graphics (See Section 5). One reason is that the basis matrix is a $4x4$ matrix, which is something graphics people work with a lot. The derivations for the various forms of cubics are just like the derivations we've seen already in this section. We will work through one more example, for practice.

A very useful form of a cubic polynomial is the *Hermite* form, where we specify the position and 1st derivative at the beginning and end. That is,

$$\begin{array}{lllllllllll} \mathbf{p_0} & = & f(0) & = \mathbf{a_0} + & 0^1 \mathbf{a_1} & + & 0^2 \mathbf{a_2}+ & & 0^3 \mathbf{a_3} \\ \mathbf{p_1} & = & f'(0) & = & \mathbf{a_1} & +2 & 0^1 \mathbf{a_2}+ & 3 & 0^2 \mathbf{a_3} \\ \mathbf{p_2} & = & f(1) & = \mathbf{a_0} + & 1^1 \mathbf{a_1} & + & 1^2 \mathbf{a_2}+ & & 1^3 \mathbf{a_3} \\ \mathbf{p_3} & = & f'(1) & = & \mathbf{a_1} & +2 & 1^1 \mathbf{a_2}+ & 3 & 1^2 \mathbf{a_3} \end{array}$$

So the constraint matrix is:

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix},$$

and the basis matrix is:

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix}$$

We will discuss Hermite cubic splines in Section 5.2.

3.4 Blending Functions

If we know the basis matrix ($\mathbf{B}$) we can multiply it by the parameter vector ($\mathbf{u}$) to get a vector of functions

$$\mathbf{b}(u) = \mathbf{u}\ \mathbf{B}.$$

Notice that we denote this vector by $\mathbf{b}(u)$ to emphasize the fact that its value depends on the free parameter u. We call the elements of $\mathbf{b}(u)$ the *blending functions* because they specify how to blend the values of the parameter vector together:

$$\mathbf{f}(u) = \sum_{i=0}^{n} \mathbf{b_i}(u)\mathbf{p_i}. \tag{9}$$

It is important to note that for a chosen value of u, Equation 9 is a *linear* equation, specifying a *linear blend* (or weighted average) of the control points. This is true no matter what degree polynomials are "hidden" inside of the $\mathbf{b_i}$ functions.

Blending functions provide a nice abstraction for describing curves. Any type of curve that can be represented as a weighted linear combination of its control points, where those weights are computed as some arbitrary functions of the free parameter.

For a single polynomial segment, writing out the blending functions is probably over-kill.

Another common term for blending function is *basis function.*

3.5 Exercises

For each of the following, find the constraint matrix, the basis matrix, and the blending functions. You should not invert matrices by hand - use a program such as MATLAB or OCTAVE (a free MATLAB-like system).

EX1: A Line Segment: Parameterized with $\mathbf{p_0}$ being 25% of the way along the segment ($u = .25$), and $\mathbf{p_1}$ being 75% of the way.

EX2: A Quadratic: Parameterized with $\mathbf{p_0}$ being the position of the beginning point ($u = 0$), $\mathbf{p_1}$ being the 1st derivative at the beginning point, and $\mathbf{p_2}$ being the 2nd derivative at the beginning point.

EX3: A Cubic: With its control points being positions that are equally spaced ($\mathbf{p_0}$ has $u = 0$, $\mathbf{p_1}$ has $u = 1/3$, $\mathbf{p_2}$ has $u = 2/3$, and $\mathbf{p_3}$ has $u = 1$).

EX4: A Quintic: (a degree 5 polynomial, so the matrices will be $6x6$) Where $\mathbf{p_0}$ is the beginning position, $\mathbf{p_1}$ is the beginning derivative, $\mathbf{p_2}$ is the middle ($u = .5$) position, $\mathbf{p_3}$ is the 1st derivative at the middle, $\mathbf{p_4}$ is the position at the end, and $\mathbf{p_5}$ is the 1st derivative at the end.

4 Putting Pieces Together

Now that we've seen how to make individual pieces of polynomial curves, we can consider how to put these curve pieces together.

4.1 Knots

When we put smaller pieces of parametric functions together to form a compound function, there are choices in how we divide up the parameter space. In the previous sections, we have always defined curves over the interval $0, 1$, so if we were to define a parametric function $\mathbf{f}$ having three pieces, we would choose:

$$\mathbf{f}(u) = \begin{array}{ll} \mathbf{f_1}(3 * u) & \text{if } 0 \leq u < \frac{1}{3} \\ \mathbf{f_2}(3 * u - 1) & \text{if } \frac{1}{3} \leq u < \frac{2}{3} \\ \mathbf{f_3}(3 * u - 2) & \text{if } \frac{2}{3} \leq u \leq 1. \end{array}$$

It is sometimes more convenient to have the curves defined over a different range. Most generally, if a curve has n segments, we could define $\mathbf{f}$ as:

$$\mathbf{f}(u) = \begin{array}{ll} \mathbf{f_1}(u) & \text{if } 0 \leq u < t_1 \\ \mathbf{f_2}(u - t_1) & \text{if } t_1 \leq u < t_1 + t_2 \\ \mathbf{f_3}(u - t_1 - t_2) & \text{if } t_1 + t_2 \leq u < t_1 + t_2 + t_3 \\ \cdots \mathbf{f_n}(u - \sum_{i=1}^{n-1} t_i) & \text{if } \sum_{i=1}^{n-1} t_i \leq u < \sum_{i=1}^{n} t_i \end{array},$$

where t_i is the length of the parameter space of piece i. This type of notation is nice when each piece might have a different parametric length. The parameter values where the curves change are known as the *knot values,* and the individual spacings, t_i in this example, are called *knot spacings.*

If you look closely, you will notice that for symmetry, each curve piece is an interval that is open at its end. In practice, when working with piecewise polynomials we will close the last interval.

Usually, basis functions are chosen so that they are non-zero only in a limited range of the parameter. At any given time, only a subset of the basis functions may be active, or potentially non-zero. A more formal definition of a *knot* is a place on the curve where the set of basis functions that are active changes.

For most of our discussion, we will consider only uniform knot spacing with the t_i. The more generalized notation will be handy when we introduce B-Splines in Section 6.2.

4.2 Putting Segments Together

If we want to make a single curve from two line segments, we need to make sure that the end of the first line segment is at the same location as the beginning of the next. There are three ways to connect the two segments(in order of simplicity):

1. Representat for the line segment as its two endpoints, and then use the same point for both. We call this a *shared-point* scheme.

2. Copy the value of the end of the first segment to the begining of the second every time that the parameters of the first segment change. We call this a *dependency* scheme.

3. Write an explicit equation for the connection, and enforce it through numerical methods as the other parameters are changed.

While the simpler schemes are preferable since they require less work, they also place more restrictions on the way the line segments are parameterized. For example, if we wanted to provide the center of the line segment as a parameter (so that the user could specify it directly), we might want to use the beginning of each line segment and the center of the line segment as their parameters. This would force us to use the dependency scheme.

Notice that if we use a shared point or dependency scheme, the total number of control points is less than $n * m$, where n is the number of segments and m is the number of control points for each segment; many of the control points of the independent pieces will be computed as functions of other pieces. Notice that if we use either the shared-point scheme for lines (each segment has its two endpoints as its parameters and shares interior points with its neighbors), or if we use the dependency scheme (such as the example one with the first endpoint and midpoint), we end up with $n + 1$ controls for an n segment curve.

Dependency schemes have a more serious problem. A change in one place in the curve can propagate through the entire curve. This is called a lack of *locality.* Locality means that if you move a point on a curve it will only effect a local region. The local region might be big, but it will be finite. If a curve's controls do not have locality, changing a control point may effect points infinitely far away.

To see locality, and the lack thereof, in action, consider two chains of line segments, as shown in Figure 3. One chain has its pieces parameterized by their endpoints and uses point-sharing to maintain continuity. The other has its pieces parameterized by an endpoint and midpoint, and uses dependency propagation to keep the segments together. The two segment chains can represent the same curves: they are both a set of n connected line segments. However, because of locality issues, the endpoint-shared is likely to be more convenient for the user. Consider changing the position of the first control point in each chain. For the endpoint-sharing version, only the first segment will change, while all of the segments will be affected in the midpoint version, as in Figure 3. In fact, for any point moved in the endpoint-shared version, at most two line segments will change. In the midpoint version all segments after the control point that is moved will change, even if the chain were infinitely long.

In this example, the dependency propagation scheme was the one that did not have local control. This is not always true. There are direct sharing schemes that are not local, and propagation schemes that are.

We emphasize that locality is a convenience of control issue. While it is inconvenient to have the entire curve change every time, the same changes can be made to the curve. It simply requires moving several points in unison.

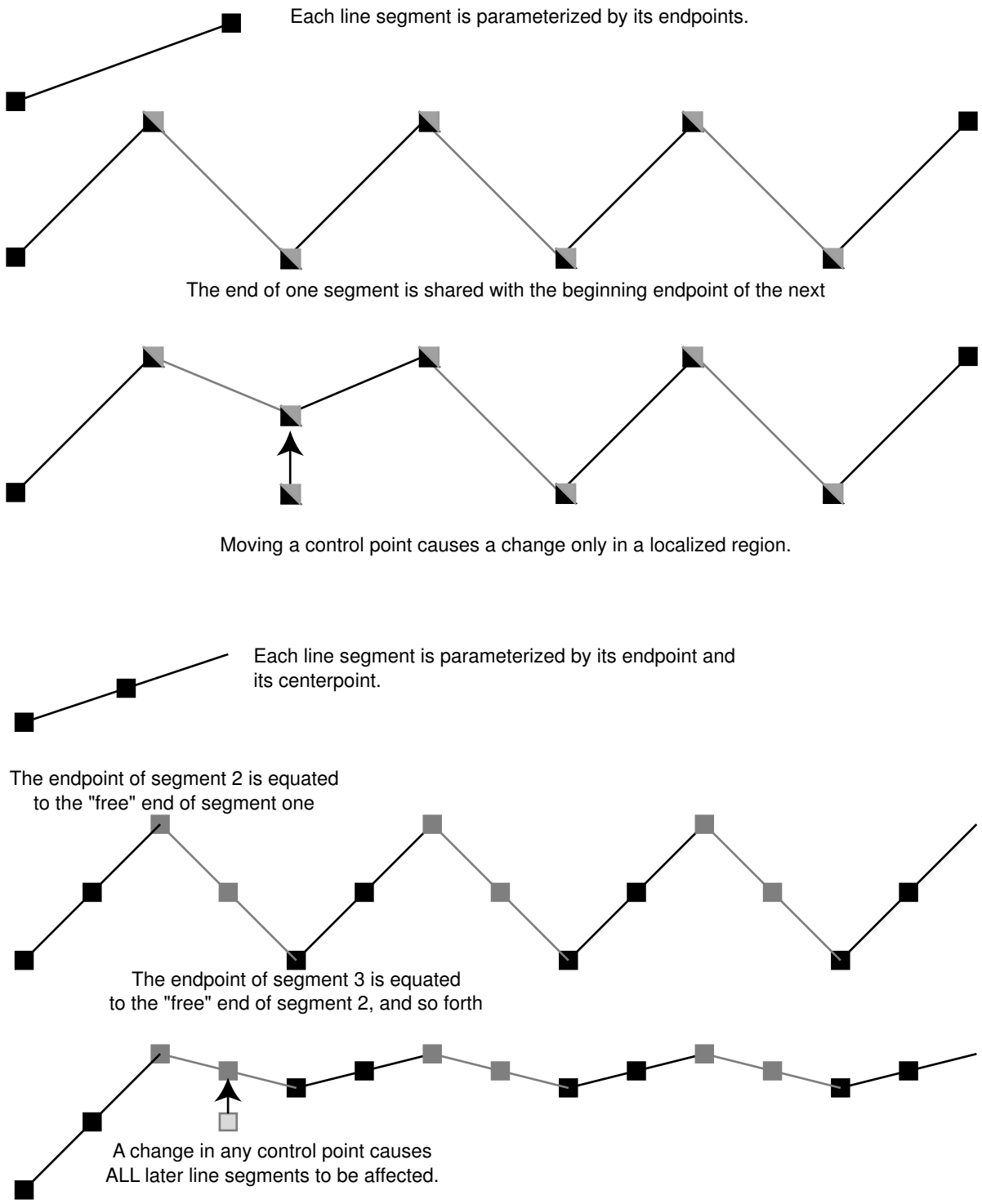

Figure 3: A chain of line segments with local control and one with non-local control.

5 Cubics

In graphics, when we make polynomial curves that are not lines, they are almost always cubics. There are a number of reasons why cubics are popular in computer graphics:

- Cubic polynomials allow for $C(2)$ continuity, which is generally sufficient for most visual tasks. The $C(1)$ smoothness that quadratics offer is often insufficient. The greater smoothness offered by higher order polynomials is rarely important.
- Cubic curves provide the minimum-curvature interpolants to a set of points. That is, if you have a set of $n + 3$ points and define the "smoothest" curve that passes through them (that is the curve that has the minimum curvature over its length), this curve can be represented as a piecewise cubic with n segments.
- Cubic polynomials have a nice symmetry where position and derivative can be specified at the beginning and end.
- Cubic polynomials have a nice tradeoff between the numerical issues in computation and the smoothness.
- The linear equations that cubics lead to (when using the matrix forms of Section 3.1.1) are 4×4, and computer graphicists tend to like 4×4 matrices.
- Everyone in computer graphics seems to use cubics. It is what the books talk about, its what the graphics libraries best support, ... The popularity of cubics is somewhat self-perpetuating!

Notice that we do not have to use cubics. They just tend to be the best tradeoff between amount of smoothness and complexity. Different applications may have different tradeoffs. For example, many font rendering systems use quadratic polynomials. We focus on cubics since they are what most computer graphics end up use.

A segment of a cubic is defined by:

$$\mathbf{f}(u) = a_0 + a_1\ u + a_2\ u^2 + a_3\ u^3.$$

As we discussed in Section 3, the polynomial coefficients are not usually a convenient way to describe a cubic segment. Instead, we will choose alternate representations that can be converted to this canonical form using the methods of Section 3.

Unfortunately, there is no single "best" representation for a piecewise cubic. It is not possible to have a curve representation that has all of the following desirable properties:

1. the curve is a cubic;
2. the curve interpolates its control points;
3. the curve has local control;
4. the curve has $C(2)$ continuity.

We can have any three of these properties, but not all four. The are representations that have any combination of three. In this document, we will discuss cubic B-Splines that do not interpolate their control points (but have local control and are $C(2)$), Cardinal and Catmull-Rom splines that interpolate their control points and offer local control but are not $C(2)$, and natural cubics that interpolate and are $C(2)$, but do not have local control.

The continuity properties of cubics refer to the continuity between the segments (at the knot points). The cubic pieces themselves have infinite continuity in their derivatives (the way we have been talking about continuity so far). Note that if you have a lot of control points (or knots), the curve can be wiggly, which might not seem "smooth."

5.1 Natural Cubics

With a piecewise cubic curve, it is possible to create a $C(2)$ curve. To do this, we need to specify the position, first and secondd derivative at the beginning of each segment (so that we can make sure that it is the same as the end of the previous segment). Notice, that each curve segment receives 3 out of its 4 parameters from the previous curve in the chain. These $C(2)$ continuous chains of cubics are sometimes referred to as *natural* cubics.

For one segment of the natural cubic, we need to parameterize the cubic by the positions of its endpoints, and the first and second derivative at the beginning point. The control points are therefore:

$$\begin{array}{lll} \mathbf{p_0} & = f(0) & = \mathbf{a_0} + 0^1\ \mathbf{a_1} + 0^2\ \mathbf{a_2} + 0^3\ \mathbf{a_3} \\ \mathbf{p_1} & = f'(0) & = \mathbf{a_1} + 2\ 0^1\ \mathbf{a_2} + 3\ 0^2\ \mathbf{a_3} \\ \mathbf{p_2} & = f''(0) & = 2\ \mathbf{a_2} + 6\ 0^1\ \mathbf{a_3} \\ \mathbf{p_3} & = f(1) & = \mathbf{a_0} + 1^1\ \mathbf{a_1} + \ 1^2\ \mathbf{a_2} + 1^3\ \mathbf{a_3}. \end{array}$$

Therefore, the constraint matrix is:

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

and the basis matrix is:

$$\mathbf{B} = \mathbf{C^{-1}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & .5 & 0 \\ -1 & -1 & -.5 & 1 \end{bmatrix}.$$

The disadvantage of natural cubic splines is that they are not local. Any change in any segment may require the entire curve to change (at least the part after the change was made). Another issue is that we only have control over the derivatives at the curve at its beginning. Segments after the beginning of the curve determine their derivatives from their beginning point.

5.2 Hermite Cubics

Hermite cubic polynomials were discussed in Section 3.3. A segment of a cubic Hermite Spline allows the positions and first derivatives of both of its end points to be specified. A chain of

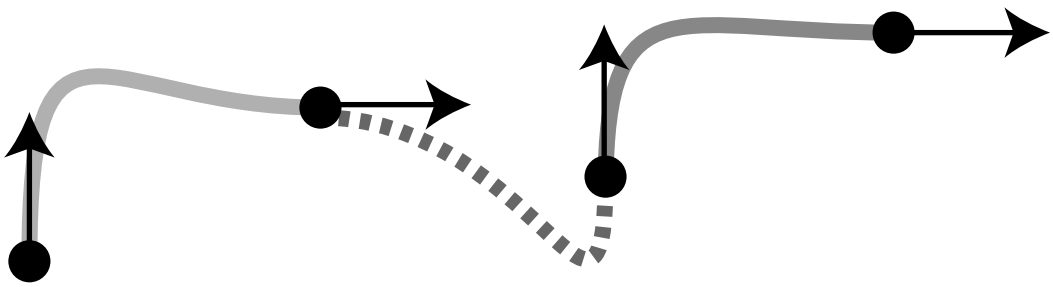

Figure 4: A Hermite Cubic Spline made of 3 segments.

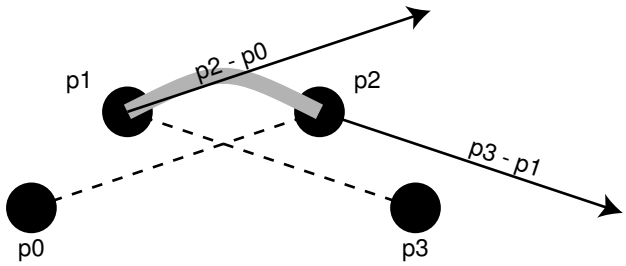

Figure 5: A segment of a cardinal cubic spline interpolates its second and third control points ($\mathbf{p_1}$ and $\mathbf{p_2}$), and uses its other points to determine the derivatives at the beginning and end.

segments can be linked into a $C(1)$ spline by using the same values for the position and derivative of the end of one segment and the beginning of the next.

A Hermite spline specifies a curve from a sequence of points. Each point has an associated vector (for the derivative at that point). The spline interpolates the points, as shown in Figure 4.

Hermite cubics are convenient because they provide local control over the shape, and provide $C(1)$ continuity. However, since the user must specify both positions and vectors, a special interface for the vectors must be provided. One possibility is to provide the user with points that represent where the derivative vectors would end if they were "placed" at the position point.

5.3 Cardinal Cubics

A *Cardinal Cubic Spline* is a type of $C(1)$ interpolating spline made up of cubic polynomial segments. A Cardinal Spline interpolates all of its controls except for the first and last (as opposed to a Hermite Spline which interpolates half of its controls, and uses the other half to specify derivative vectors). Cardinal Splines have a parameter called *tension* that controls how "tight" the curve is between the points it interpolates. For the important special case of $t = 0$, the splines are called *Catmull-Rom* splines.

A segment of a Catmull-Rom spline interpolates its second and third points. The derivative at the beginning of the curve is determined by the vector between the first and third control points, while the derivative at the end of the curve is given by the vector between the second and forth points, as shown in Figure 5.

The tension parameter adjusts how much the derivatives are scaled. Specifically, the derivatives

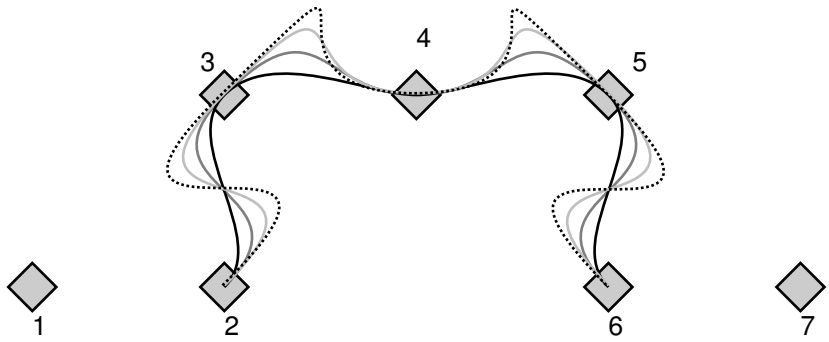

Figure 6: Cardinal splines through the 7 control points with varying values of tension parameter t.

are scaled by $1/2(1-t)$. The constraints on the cubic are therefore:

$$\begin{array}{rl} f(0) = & \mathbf{p_1} \\ f(1) = & \mathbf{p_2} \\ f'(0) = & \frac{1}{2}(1-t)(\mathbf{p_2} - \mathbf{p_0}) \\ f'(1) = & \frac{1}{2}(1-t)(\mathbf{p_3} - \mathbf{p_1}) \end{array} .$$

Solving these for the control points (defining $s = (1-t)/2$) gives:

$$\begin{array}{rll} \mathbf{p_0} = & f(1) - \frac{2}{1-t} f'(0) & = a_0 + (1 - \frac{1}{s})a_1 + a_2 + a3 \\ \mathbf{p_1} = & f(0) & = a_0 \\ \mathbf{p_2} = & f(1) & = a_0 + a_1 + a_2 + a3 \\ \mathbf{p_3} = & f(0) + \frac{1}{s} f'(1) & = a_0 + \frac{1}{s}a_1 + 2\frac{1}{s}a_2 + 3\frac{1}{s}a3 \end{array}$$

This gives the cardinal matrix

$$\mathbf{B} = \mathbf{C^{-1}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -s & 0 & s & 0 \\ 2s & s-3 & 3-2s & -s \\ -s & 2-s & s-2 & s \end{bmatrix} .$$

Cardinal splines are useful because they are the easiest way to interpolate a set of points with $C(1)$ continuity and local control. They are only $C(1)$, so they sometimes get "kinks" in them.

In Figure 6, a set of cardinal splines through a set of points are shown. The curves use the same control points, but use different values for the tension parameters. Note that the first and last control points are not interpolated.

5.4 Tension-Continuity-Bias Controls

Omitted in this version. See the section in Hearn and Baker for details.

6 Approximating Curves

It might seem like the easiest way to control a curve is to specify a set of points for it to interpolate. In practice, however, interpolation schemes often have undesirable properties because they have

less continuity and offer no control of what happens between the points. Curve schemes that only approximate the points are often preferred. With an approximating scheme, the control points influence the shape of the curve, but do not specify it exactly. Although we give up the ability to directly specify points for the curve to pass through, we gain better behavior of the curve and local control. The two most important types of approximating curves in computer graphics are Bézier Curves and B-Splines.

6.1 Bézier Curves

Mathematician Pierre Bézier was a pioneer in the mathematical representation of free form curves. The curves that are named after him are an approximating form of polynomial curves that were specifically developed to be convenient for users to control.

A segment of a Bézier curve can be of any degree (polynomial order). Each segment is a single polynomial. Usually, complex curves are made up of a number of low order pieces. Many popular illustration programs (such as Adobe Illustrator and Macromedia Freehand) use cubic Bézier segments, and many font representation schemes (such as the one used in Postscript) use quadratic Bezier segments to describe the outlines of the characters.

Like other polynomial forms, a Bézier segment of order n has $n+1$ control points. No matter what the order of the segment is, it will always begin at its first end point and end at its last endpoint. The other control points influence the shape of the curve, but it will not pass through them. Bézier segments have the property that they always stay inside the convex hull of their control points.

The first derivatives (tangents) of Bézier curves are always proportional to the vector between the end control points and the ones they are adjacent to. The derivative at the beginning is a fraction of the vector between the first and second control points, and the derivative at the end of the segment is a fraction of the vector between the second to last and last end point. The proportionality constant is the order of the curve segment. Therefore the first derivative of a cubic Bézier at its beginning is 3 times the vector between the first and second control point.

Note that cubic Béziers bear a resemblance to cubic Hermite polynomials. Both allow for easy specification of the ends of the segments, and easy specification of the first derivatives at the ends. The key difference is that all of the controls of the Bézier curve are in the same space as the curve itself, whereas the some of the controls of the Hermite polynomial are vectors. Also, remember the scaling factor between the vector between points and the derivatives in the Bézier curves.

Given that we know what the control points of a Bézier curve do, we could write constraints and solve for the parametric form as we did in Section 3.3 (and you will be asked to do this in an Exercise). However, the blending functions for Bézier curves have the common form that works for all orders:

$$b_{k,n}(u) \;=\; C(n,k)\; u^k\; (1-u)^(n-k),$$

where n is the order of the Bézier curve, and k is the blending function number between 0 and n (inclusive). $C(n,k)$ is the binomial coefficients

$$C(n,k) = \frac{n!}{k!\;(n-k)!}.$$

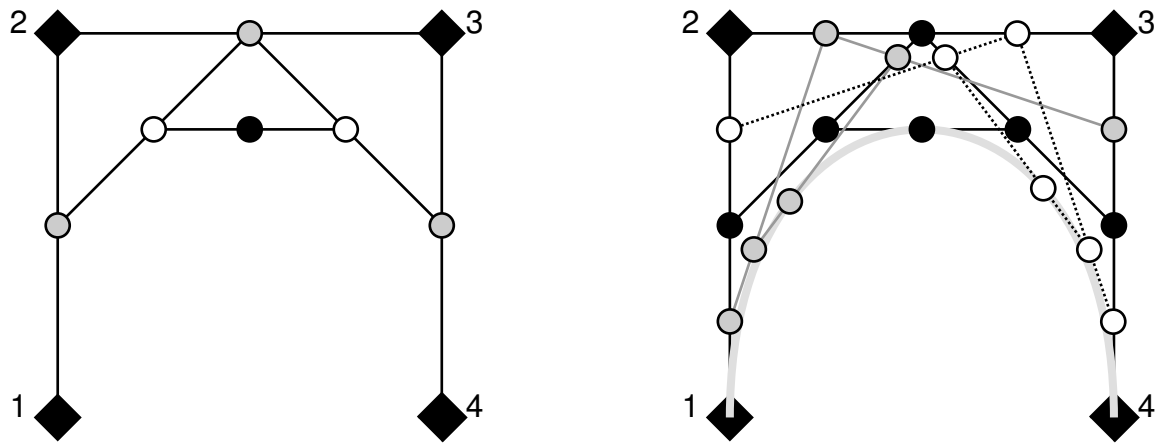

Figure 7: An illustration of the De Casteljau algorithm for a cubic Bézier . The left shows the construction for $u = .5$. The right shows the construction for .25, .5, and .75.

Given the positions of the control points $\mathbf{p_i}$, the function to evaluate the Bézier curve of order n (with $n + 1$ control points) is:

$$\mathbf{p}(u) \;=\; \sum_{k=0}^{n} p_k C(n,k)\; u^k\; (1-u)^(n-k).$$

Continuity between Bézier curve segments must be created by placing the control points of each segment in the right place. For $C(0)$ continuity, the first point of one curve must be the last point of the other. For $G(1)$ continuity, the 2nd to last point of the first curve and the second point of the 2nd curve must be collinear with the equated endpoints. For $C(1)$ continuity, the distances between the points must be equal as well.

6.1.1 The De Casteljeau Algorithm

One nice feature of Bézier Curves is that there is a very simple and general method for computing them. The method, called the *de Casteljau Algorithm,* uses a sequence of linear interpolations to compute the positions along the Bézier curve of arbitrary order.

The De Casteljau algorithm begins by connecting every adjacent set of points with lines, and finding the point on these lines that is the u interpolation, providing a set of $n-1$ points. These points are then connected with straight lines, those lines are interpolated (again by u), giving a set of $n-2$ points. This process is repeated until there is one point. An illustration of this process is shown in Figure 7.

6.2 B-Splines

Suppose we were to develop a type of piecewise polynomial curve that approximated a set of points. Given n points, we would want to define n blending or basis functions, defined over the domain of the curve $u \in [0, n]$. B-Splines provide a method for defining these basis functions for any degree $d-1$ of polynomial (although, the degree must be less than the number of points). B-Splines provide a number of very useful properties:

1. The basis functions (as well as the resulting curve that is the sum of the basis functions) is $C(d-2)$ *continuous.*

2. The basis functions are *local.* That is, they are zero except for a finite size region that depends only on d, not on n. This means that the points give local control over the curve.

3. The basis function sum up to one for any parameter value where there are d active basis functions. This is important since it means that shifting all of the points causes the entire curve to shift.

4. The blending functions have a simple and (in the opinion of many people, including myself) elegant procedure for construction, thanks to the method of Cox and de Boor. This procedure is very general.

5. The curves tend to be very mathematically well behaved, and respond nicely to changes in the point positions. Therefore, even though they do not interpolate the points, they still can be used interactively by having the user drag the points around.

6. The basis functions can be defined for any set of knot values.

To use the B-Splines, we need to have a weight each blending function by the value of its control point,

$$f(t) = \sum_{k=0}^{n} \mathbf{p}_k \; B_{k,d}(t).$$

To illustrate the B-Splines, we'll consider the simple ones with $d = 2$. This means that the polynomials will be of degree 1 (to be consistent with the literature, the degree of the B-Spline is one more than the degree of the polynomial - don't ask me why). The blending functions have the form:

$$B_{i,2}(t) = \begin{array}{ll} 0 & \text{if } t < i-1 \\ 1-(i-t) & \text{if } i-1 \le t < i \\ 1-(t-i) & \text{if } i \le t \le i+1 \\ 0 & \text{if } t > i+1 \end{array}.$$

These blending functions are graphed for various values of i in Figure 8.

Notice that for $d = 2$, the B-Splines create a linear interpolation of the points, and that for this case, all of the useful properties are met (the curve is $C(0)$, moving a point only affects the curve near it, ...). The case of $d = 2$ is unusual for B-Splines because it does interpolate the points.

You should also notice that for this example, the knot values are the integers from 0 to n, inclusive. More general methods for defining B-Splines would allow us to provide a vector of knot values. We denote the knot vector by $\mathbf{u}$, and will use a subscript to refer to an individual knot value. For the $d = 2$ example, knot value $\mathbf{u}_i$ is the parameter value of the curve where the curve interpolated point i. The blending functions would then have the form:

$$B_{i,2}(t) = \begin{array}{ll} 0 & \text{if } t < u_{i-1} \\ 1-\frac{u_i-t}{u_i-u_{i-1}} & \text{if } u_{i-1} \le t < u_i \\ 1-\frac{t-u_i}{u_{i+1}-u_i} & \text{if } u_i \le t \le u_{i+1} \\ 0 & \text{if } t > u_{i+1} \end{array}. \tag{10}$$

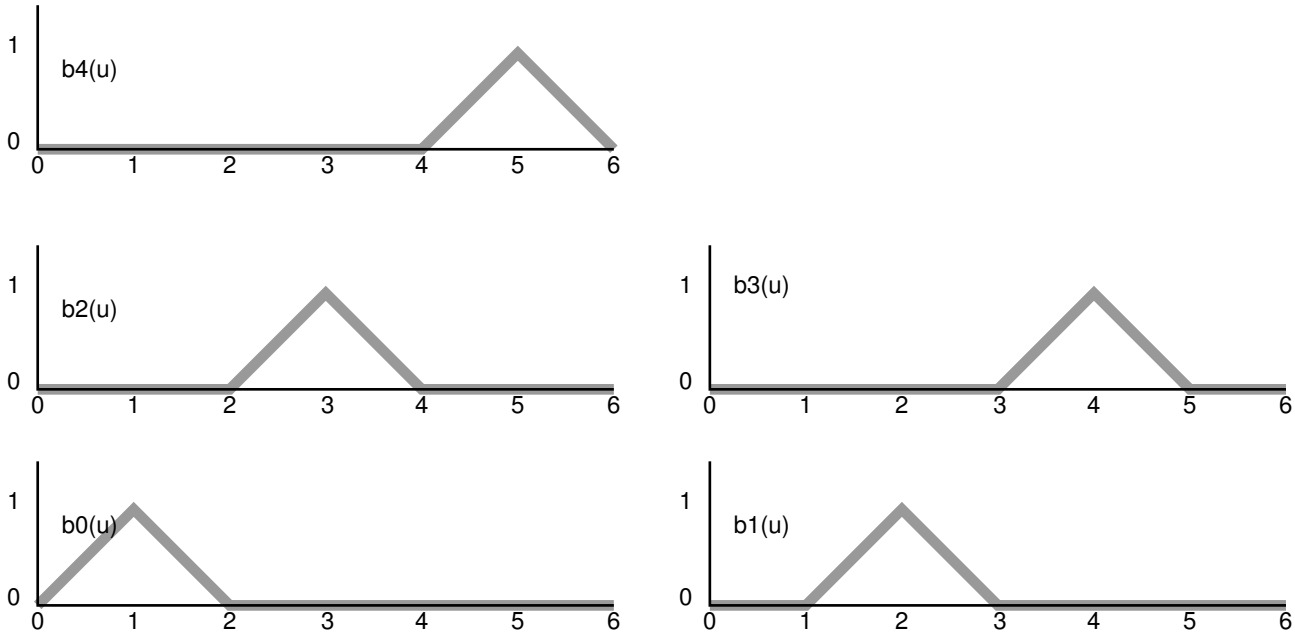

Figure 8: B-Spline blending functions for $d = 2$ and $n = 5$ and uniformly spaced knots at the integers.

The general recurrence (the Cox / de Boor equation) for basis function k of arbitrary B-Spline order d, and knot vector $\mathbf{u}$ is:

$$B_{k,1}(t) = \begin{array}{ll} 1 & \text{if } \mathbf{u}_k \leq t \leq \mathbf{u}_{k+1} \\ 0 & \text{otherwise} \end{array}$$

$$B_{k,d}(t) = \frac{t-\mathbf{u}_k}{\mathbf{u}_{k+d-1}-\mathbf{u}_k} B_{k,d-1}(t) + \frac{\mathbf{u}_{k+d}-t}{\mathbf{u}_{k+d}-\mathbf{u}_{k+1}} B_{k+1,d-1}(t)$$

The B-Spline functions for the knot vector $[0, 1, 2, 3, 4, 5, 6, 7, 8]$ are shown for various values of d in Figures 9 through 11.

6.3 Uniform B-Splines

If the spacing between the knots is uniform, then the B-Spline is referred to as a *uniform* B-Spline. A *non-uniform* B-Spline is a B-Spline that makes use of the extra generality afforded by non-uniform spacing.

For uniform B-Splines, the blending functions are all very similar. In fact, they have the same shape, just shifted to a different "place." For this reason, uniform B-Splines are said to be *periodic.*

The periodic basis functions can create a problem at the beginning and end: because it takes d basis functions to add up to one, before knot $d - 1$, there aren't enough basis functions. Notice in Figure 8 how the functions sum to one between that values of 1 and 5. Within this range, the B-Spline curve caused by summing the basis functions is useful.

There are three different ways to handle the problem of basis functions not summing up:

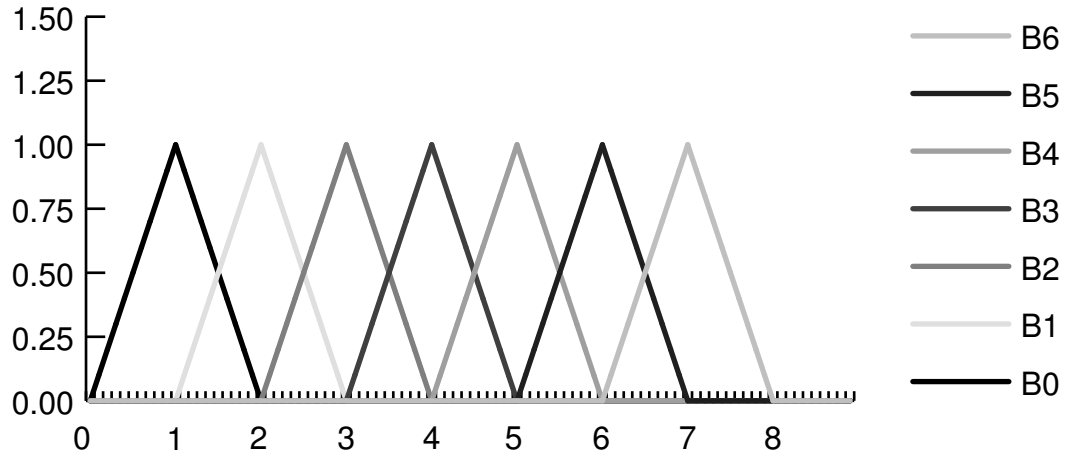

Figure 9: B-Spline functions for knot vector $[0, 1, 2, 3, 4, 5, 6, 7, 8]$ and $d = 2$.

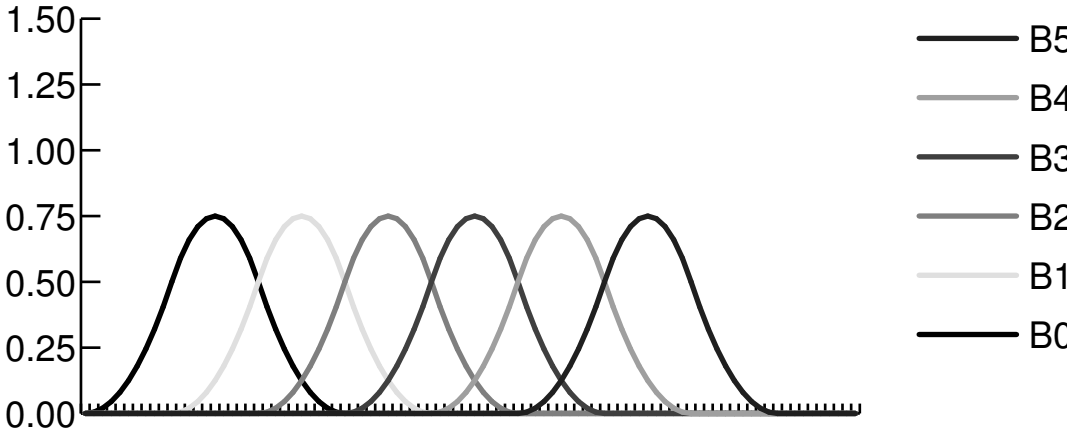

Figure 10: B-Spline functions for knot vector $[0, 1, 2, 3, 4, 5, 6, 7, 8]$ and $d = 3$.

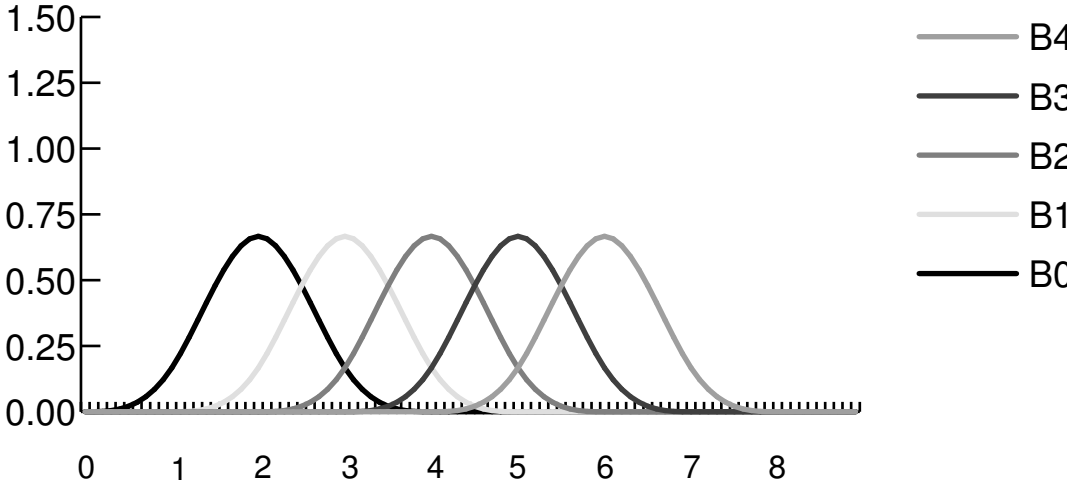

Figure 11: B-Spline functions for knot vector $[0, 1, 2, 3, 4, 5, 6, 7, 8]$ and $d = 4$.

1. We can restrict the values of the parameter that we look at to the useful range. In the example of Figure 8 this is 1 to 5.

2. We can add extra knots that are outside of the range that we want the parameter to span. In the example of Figure 8, we can add a knot (and therefore a basis function) for -1 works over the entire range 0 to 5. This means we need to specify another control point for the new blending function.

3. If the curve is closed, then we can use the control points of the last knots for the beginning, and of the first control points for the ends. This is another way that the uniform B-Splines are periodic (the period is around the closed curve).

If we are willing to use non-uniform B-Splines, we can repeat the first and last knot multiple times in order to get the extra knots we need to make the curve span the correct parameter range. Because we associate control points with knot *values* not their indices, this does not require adding any new control points. If we replicate the first and last knots enough, the B-Splines will interpolate their end control points.

6.3.1 Cubic B-Splines

For reasons discussed earlier, cubic polynomials are popular in graphics. B-Splines of order $d = 4$ (since the B-Spline order is one more than the polynomial order) are often used.

A different way to look at B-Splines is as a bunch of polynomial segments, each using the same set of blending functions on a different set of points. This form makes B-Splines work like the curves we talked about in Sections 3 and 5.

The basis matrix for periodic, uniform, cubic B-Splines is:

$$\mathbf{M_b} = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}.$$

To make segments connect properly (with $C(2)$ continuity) they must share 3 points. That is, if the first segment of the curve uses points $\mathbf{p_0}$, pI, $\mathbf{p_2}$, and $\mathbf{p_3}$, then the next segment should use points $\mathbf{p_1}$, $\mathbf{p_2}$, $\mathbf{p_3}$, $\mathbf{p_4}$.

The uniform, cubic B-Spline segment is similar to a cardinal cubic in that it connects the 2nd and 3rd of its 4 points. Unlike a cardinal spline, however, it does not interpolate its points. For giving up interpolation, you get better continuity (the cubic B-Splines are $C(2)$).

6.4 Curve Fitting

Suppose that you have a set of points that you want a curve to interpolate. You might make an interpolating curve that passes through all of them, but then it might be hard to get the continuity and other nice properties of B-Splines. Instead, you might want to adjust a B-Spline (or other piecewise polynomial curve) to fit the points as closely as possible.

6.5 Exercises

EX1: Basis Matrix for Cubic Bézier The constraints for a segment of a Bézier cubic are:

$$\begin{aligned} f(0) &= p_0 \\ f'(0) &= \tfrac{1}{3}(p_1 - p_0) \\ f(1) &= p_3 \\ f'(1) &= \tfrac{1}{3}(p_3 - p_2) \end{aligned}$$

Using the methods of Section 3, derive the basis matrix and blending functions for a Bézier cubic segment.

EX2: De Castijeau Algorithm Use the De Castijeau algorithm to evaluate the position of the cubic Bézier curve with its control points at (0,0), (0,1), (1,1) and (1,0) for parameter values $u = .5$ and $u = .75$. Drawing a sketch will help you do this.

EX3: Cox / de Boor Recurrence Use the Cox / de Boor recurrence to derive Equation 10.