Announcements

- First assignment on the WWW.
- Course reader is here.
- Thursday: plants and trees. *CIS*.
- Tuesday: polar forms. *CIS*.

The assignment is due in two weeks, right before PDI field trip. It looks long, but it could have been 2 pages if I did not include lots of helpful hints and review material. The programming part is easy; my solution is just 90 lines long including comments.

The course reader is available on time. It's expensive due to copyrights. There are extra copies for auditors. There is one copy on reserve at the MCS library.

Are Milton Chen and Anthony Ramirez here?

Most color figures from the papers, as well as some animations, are on the WWW. I also brought the papers to class to see the pictures.

Today

Parametric polynomial curves:

- Shape.
- Interpolating curves.
- Bases.
- Bézier curves.
- Hermite curves.

Conventions

- Our world is the plane.
- Cartesian coordinates.

Extention to space is straightforward.

The coordinate systems and notations from the last lecture will be useful later in the course.

Parametric Polynomial Curves

Parametric polynomial curve $\mathbf{p}(t)$:

Curve: set of points $\mathbf{p} = (x, y)$ with connectivity of line.

Parametric: $x = x(t)$ and $y = y(t)$; t is *parameter*.

Polynomial: $x(t)$ and $y(t)$ are polynomials, e.g.

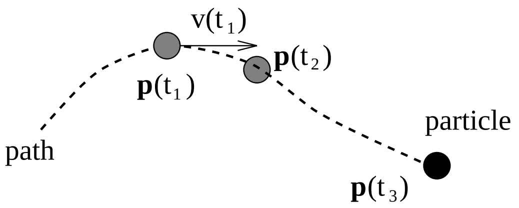
$$x(t) = 3t^2 + 4t + 1 \quad \text{and} \quad y(t) = t^3 + 5.$$

If I stand on a point of a curve, I am surrounded by a finite number of other points, usually 2 (or 4 at intersections); this, in topologists' lingo, is a point's *neighborhood*. Points of *surfaces* are surrounded by disks, i.e. an infinite number of other points.

The non-parametric form is $y(x)$. Its problem is that it is not very pliable since each x has only one y; hence, it is far from ideal for modeling.

Any kind of function is possible, such as sines and cosines which yield Lissajous figures, familiar to electrical engineers. We like polynomials because it's easy to prove their properties and we can evaluate them quickly.

Shape: Moving Particle



Curve shape is trace of a moving particle:

Time: real number t, the parameter.

Position: point $\mathbf{p}(t) = (x(t), y(t))$.

Velocity: vector $\vec{v}(t) = (x'(t), y'(t))$.

Acceleration: vector $\vec{a}(t) = (x''(t), y''(t))$.

For example, we can measure time in seconds since the Big Bang.

The position is a point on the curve.

Look at the diagram: when $\mathbf{p}(t_1)$ and $\mathbf{p}(t_2)$ come close to each other, their difference is tangent to the curve. Hence the velocity is tangent, too, as it is just a scalar multiple of this difference.

The acceleration is the difference of the velocity vectors. If the curve exists in 3D space, then the position, velocity, and acceleration define *osculating* plane.

Shape: Parameterization

Time is relative: two curves $\mathbf{p}_1(t)$ and $\mathbf{p}_2(t)$ can have different formulas, same shape, e.g.

- Let $\mathbf{p}_1(t) = (t, t^2)$, t in seconds since Big Bang.
- Then $\hat{t} = t/60 - 10$ is time in minutes, starting 10 minutes after Big Bang.
- Rewrite as $t = 60(\hat{t} + 10)$.
- $\mathbf{p}_2(\hat{t}) = (60(\hat{t} + 10), 60^2(\hat{t} + 10)^2)$ has same shape as $\mathbf{p}_1(t)$.

Pretend that you are a Borg, using the Borg calender: the motion of the same particle will have different formulas than the ones humans use since aliens measure time differently. This does not change the path that the particle followed!

What we show is an *affine parameter transformation*. Remember last lecture: an affine transformation is a multiplication by a matrix (a 1-by-1, or just a scalar, in this case) followed by translation.

It is true that same parameter values yield different points on the two curves. But the overall shape, the path of the particle, is the same and that's what counts.

Shape: Degree

A curve $\mathbf{p}(t) = (x(t), y(t))$ has degree d iff

$$d \geq \deg x(t) \quad \text{and} \quad d \geq \deg y(t).$$

Higher degree has more *expressive power*:

Deg 0: $\mathbf{p}(t) = (a_x, a_y)$. Points.

Deg 1: $\mathbf{p}(t) = (a_x + b_x t, a_y + b_y t)$. Lines.

Deg 2: Parabolas.

Deg 3: Cubics...

Some people define the degree of a curve as the maximum of the degrees of its component polynomials. Our definition allows the component polynomials to have degree *at most* d.

More expressive power means the curve can assume more complex shapes. Essentially, higher degree polynomials have more coefficients, and so their shapes have more degrees of freedom.

Deg 0: The particle is a couch potato, stationary at point (a_x, a_y).

Deg 1: Both $x(t)$, $y(t)$ change linearly over time. It is only possible to move along a straight line!

Deg 2: The formula is similar and thus omitted. Although not trivial to see, the 3D version of a degree 2 curve is always planar. So, we still need higher degree than 2 to do anything interesting.

Deg 3: Comes next...

Shape: Cubics

Variety of shapes:

Humpy: $\mathbf{p}(t) = (t^2, t^3 + t)$.

Loopy: $\mathbf{p}(t) = (t^2, t^3 - t)$.

Pointy: $\mathbf{p}(t) = (t^2, t^3)$.

S-shaped: $\mathbf{p}(t) = (t, t^3)$.

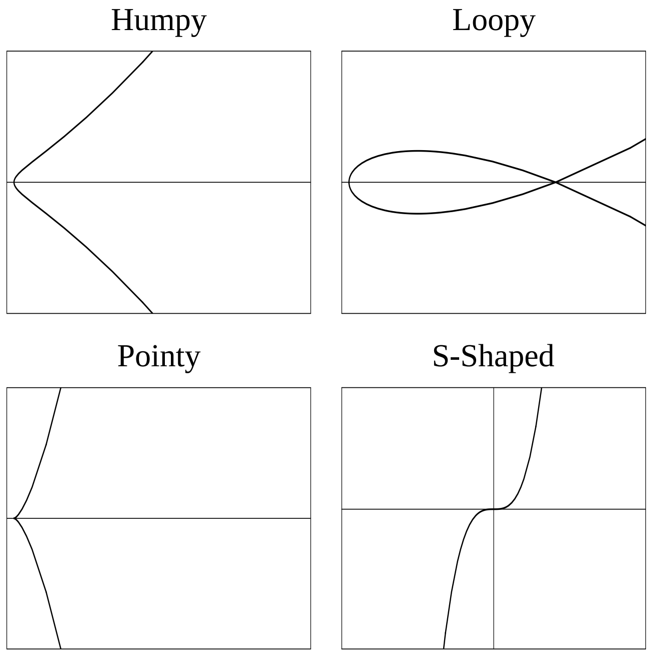
and of course...

Parabolic: $\mathbf{p}(t) = (t, t^2)$.

Linear: $\mathbf{p}(t) = (t, t)$ or $\mathbf{p}(t) = (t^3, t^3)$.

... and point (stationary).

Shape: Cubics



The horizontal axis is $x(t)$ and the vertical is $y(t)$. The ranges on the axes vary from plot to plot; they are not listed because they do not really matter: it's just the overall shape of the curve that is important here.

Shape: Specifying

Assume cubics from now on:

$$\mathbf{p}(t) = \mathbf{a}_1 + \vec{a}_2 t + \vec{a}_3 t^2 + \vec{a}_4 t^3.$$

User may define shape by setting $\mathbf{a}_1, \vec{a}_2, \vec{a}_3, \vec{a}_4$: unintuitive!

Better: user specifies points *on* curve.

Problem: given points on curve, find curve that *interpolates* these points.

Everything we are about to discuss can be trivially extended to arbitrary degree polynomial curves. The reason we discuss cubics is that they are the most common building block for *splines*, curves that are piecewise polynomial: cubics are flexible enough to assume many shapes, but they have a low degree which makes them more manageable and makes their evaluation efficient.

This form of a cubic is really a collection of equations, one for each dimension x, y, and so on. Each dimension has its own coefficients, collectively represented by $\mathbf{a}_1$ and the $\vec{a}_i$'s. We use this notation because it hides the details of the particular dimension, and allows us to write formulas which are valid for arbitrary dimensions: all that changes for higher dimension is the dimension of $\mathbf{a}_1$, of the $\vec{a}_i$'s, and of the result $\mathbf{p}(t)$.

The reason $\mathbf{a}_1$ is a point can be seen by considering

$$\mathbf{p}(0) = \mathbf{a}_1.$$

Since $\mathbf{p}(t)$ is a point for all t, this equality forces us to conclude that $\mathbf{a}_1$ must be a point, as well. Also, rewriting the general form as

$$\mathbf{p}(t) - \mathbf{a}_1 = \vec{a}_2 t + \vec{a}_3 t^2 + \vec{a}_4 t^3,$$

and employing site notation, we can see that the left-hand side is always a difference of points and thus a vector. It follows that the right-hand side must also be a vector for all t; this is only possible if the $\vec{a}_i$'s all have a weight of 0, and hence they are all vectors.

Interpolating Curves: Definition

Given *control points*

$$\mathbf{p}_1 = (x_1, y_1), \ldots, \mathbf{p}_n = (x_n, y_n),$$

and times

$$t_1, \ldots, t_n,$$

find the curve $\mathbf{p}(t)$ that passes through the given points at the corresponding times, i.e.

$$\mathbf{p}(t_1) = \mathbf{p}_1, \ldots, \mathbf{p}(t_n) = \mathbf{p}_n.$$

Interpolating Curves: Vandermonde

Find $\mathbf{a}_1$, $\vec{a}_i$'s by solving system of equations

$$\begin{aligned}\mathbf{p}_1 &= \mathbf{p}(t_1) = \mathbf{a}_1 + \vec{a}_2 t_1 + \vec{a}_3 t_1^2 + \vec{a}_4 t_1^3 \\ &\vdots \\ \mathbf{p}_n &= \mathbf{p}(t_n) = \mathbf{a}_1 + \vec{a}_2 t_n + \vec{a}_3 t_n^2 + \vec{a}_4 t_n^3.\end{aligned}$$

Rewrite as

$$\begin{pmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_n \end{pmatrix} = \begin{pmatrix} 1 & t_1 & t_1^2 & t_1^3 \\ 1 & t_2 & t_2^2 & t_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & t_n & t_n^2 & t_n^3 \end{pmatrix} \begin{pmatrix} \mathbf{a}_1 \\ \vec{a}_2 \\ \vec{a}_3 \\ \vec{a}_4 \end{pmatrix}.$$

If t_i's are distinct,

$n < 4$: infinite solutions.

$n > 4$: no solutions.

$n = 4$: makes matrix square; one solution.

Each equation stands for two equations, one for the x components of $\mathbf{a}_1$ and the $\vec{a}_i$'s, one for the y components. So, in effect, these n equations describe *two* separate systems of n equations.

What should n be? That is, how many points need be specified to fully define a particular cubic?

Requiring that the t_i's are distinct means that we are not asking the moving particle to be mr. Houdini, and appear at two different places at the same time. Well, of course, we can allow same t_i's provided corresponding $\mathbf{p}_i$'s are the same, but that's redundant.

$n < 4$: Under-determined system; cubics are versatile (pliable) enough for many different cubics to pass through the given points.

$n > 4$: Overdetermined system; cubics are not *that* versatile!

$n = 4$: The system has a unique solution iff the matrix is invertible, which it is iff the t_i's are distinct. Note that if two t_i's were the same, two rows of the matrix would become identical and its determinant would vanish; this would be an indication that the system has either infinite solutions or no solution.

In general, to unambiguously and successfully specify a $d = n - 1$ degree curve, we need $d + 1 = n$ points. Or, a curve has to have degree $d = n - 1$ (at least) to pass through $d + 1 = n$ *arbitrary* points.

Interpolating Curves: Lagrange

Let

$$L_1(t) = \frac{(t - t_2)(t - t_3)(t - t_4)}{(t_1 - t_2)(t_1 - t_3)(t_1 - t_4)}.$$

Then

$$\begin{aligned} L_1(t_1) &= 1 \\ L_1(t_j) &= 0, \text{ if } j \neq 1. \end{aligned}$$

General cubic *Lagrange polynomials*:

$$\begin{aligned} L_i(t) &= \frac{\prod_{j=1, j\neq i}^{j=4} (t - t_j)}{\prod_{j=1, j\neq i}^{j=4} (t_i - t_j)} \\ L_i(t_i) &= 1 \\ L_i(t_j) &= 0, \text{ if } j \neq i. \end{aligned}$$

Interpolant:

$$\mathbf{p}(t) = \mathbf{p}_1 L_1(t) + \mathbf{p}_2 L_2(t) + \mathbf{p}_3 L_3(t) + \mathbf{p}_4 L_4(t).$$

The same idea works for higher degree than cubics. In the formulas for the Lagrange polynomials for degree $d = n - 1$ curves, the product signs go up to $d + 1 = n$ instead of just 4.

Recall that the site notation requires that the Lagrange polynomials sum to one. They do, although it's not trivial to prove this.

This method is numerically unstable. That is, in theory it works fine, but if you try to implement it, you get roundoff errors, overflows, underflows, etc.

Interpolating Curves: Summary

Vandermonde: yields coefficients

$$\mathbf{a}_1, \vec{a}_2, \vec{a}_3, \vec{a}_4.$$

Lagrange: expresses answer as

$$\mathbf{p}(t) = \mathbf{p}_1 L_1(t) + \mathbf{p}_2 L_2(t) + \mathbf{p}_3 L_3(t) + \mathbf{p}_4 L_4(t).$$

Other interpolation methods:

- Newton.
- Aitken.
- ... and more.

Of course, by expanding the $L_i(t)$'s, we can find $\mathbf{a}_1$ and the $\vec{a}_i$'s.

Bases: Definition

Function space: a subset of all possible functions, e.g. all cubic polynomials.

Basis: a minimal collection of members $B_i(t)$ of a function space which can produce all other members as linear combinations:

$$\mathbf{p}(t) = \mathbf{b}_1 B_1(t) + \cdots + \mathbf{b}_n B_n(t).$$

Example bases for cubics:

Monomials: The $B_i(t)$'s are $1, t, t^2, t^3$. The $\mathbf{b}_i$'s are $\mathbf{a}_1$ and the $\vec{a}_i$'s.

Lagrange: The $B_i(t)$'s are the $L_i(t)$'s. The $\mathbf{b}_i$'s are the $\mathbf{p}_i$'s.

A function space, precisely defined, is more than just any subset. It is a subset, along with some operations defined on it, like addition; also, the sum of any two elements in a function space should be a member of the function space, too. For such tedious details, you may consult a linear algebra primer.

By a *minimal* collection we mean that we have just as many functions in the basis as we need, and no more. This requirement can be stated in two other, equivalent, ways:

- $B_1(t), \ldots, B_n(t)$ are linearly independent; that is, none of them can be expressed as a linear combination of the others.
- Every $\mathbf{p}(t)$ has one and only one set of coefficients; that is, there is a one-to-one correspondence between curves $\mathbf{p}(t)$ and value assignments for the $\mathbf{b}_i$'s.

All bases for the cubics have four members. This is the *dimension* of the function space. All this is very similar to *vector spaces*.

Only for this slide and the next one, the $\mathbf{b}_i$'s are *sites* (either points or vectors), even though we denote them using the notation usually reserved for points.

Bases: Curve Specification

General interface to specifying shape of

$\mathbf{p}(t) = \mathbf{b}_1 B_1(t) + \mathbf{b}_2 B_2(t) + \mathbf{b}_3 B_3(t) + \mathbf{b}_4 B_4(t)$:

1. Pick a basis $B_1(t), \ldots, B_4(t)$.
2. User directly specifies $\mathbf{b}_1, \ldots, \mathbf{b}_4$.

Need bases where:

- $\mathbf{b}_1, \ldots, \mathbf{b}_4$ have intuitive meaning.
- $B_1(t), \ldots, B_4(t)$ are simple, numerically stable.

We will now look for such bases, and find out that we can not only satisfy these requirements, but also get bases with other nice properties as regards rendering and transformations.

© 1995 Apostolos Lerios
Stanford University
http://graphics.stanford.edu/~tolis/

Bases: Redundancy

Lagrange basis, the $L_i(t)$'s, depends on $t_1, \ldots, t_4$.

- In modeling, we care about *path* of particle, not *how* particle carved the path: just pick one of alternate parameterizations.
- Same shape defined by different collections of t_i's and $\mathbf{p}_i$'s. e.g. curve $\mathbf{p}(t) = (t, 2t)$ passes through

$$\begin{aligned} \mathbf{p}_1 &= (0,0) \quad \text{at} \quad t_1 = 0 \\ \mathbf{p}_2 &= (1,2) \quad \text{at} \quad t_2 = 1 \\ \mathbf{p}_3 &= (2,4) \quad \text{at} \quad t_3 = 2. \end{aligned}$$

  Given any two of the three pairs of $\mathbf{p}_i$'s and t_i's, I'll always get the same curve!

Remove redundancy by fixing $t_1, \ldots, t_4$. Usually, evenly spaced in $[0,1]$: *uniform parameterization.*

© 1995 Apostolos Lerios
Stanford University
http://graphics.stanford.edu/~tolis/

Redundancy may add flexibility, as I can say *where* I want my particle to be *when*, but it is an overkill: we can instead just fix the times of interest, and ask the user to tell us where the particle should be at the requested times.

Advantages:

- We avoid erroneous input (e.g. the particle requested to appear at different points at the same time).
- It is easier to pre-calculate the basis, i.e. do some advance calculations that will make repeated evaluations of $\mathbf{p}(t)$ faster.

From now on, in the bases we are about to design, we will avoid redundancy.

© 1995 Apostolos Lerios
Stanford University
http://graphics.stanford.edu/~tolis/

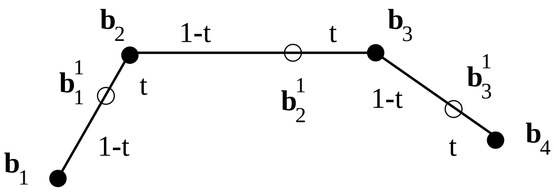
Bézier: Derivation

Bernstein basis: derive from the *de Casteljau* construction algorithm.

Given $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4$, find $\mathbf{p}(t)$ in three steps:

Step 1: Find points $\mathbf{b}_1^1, \mathbf{b}_2^1, \mathbf{b}_3^1$ by

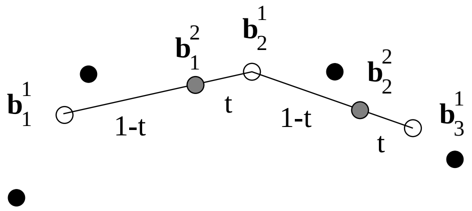
$$\begin{aligned}\mathbf{b}_1^1 &= (1-t)\mathbf{b}_1 + t\mathbf{b}_2\\ \mathbf{b}_2^1 &= (1-t)\mathbf{b}_2 + t\mathbf{b}_3\\ \mathbf{b}_3^1 &= (1-t)\mathbf{b}_3 + t\mathbf{b}_4.\end{aligned}$$



Bézier: Derivation

Step 2: Find points $\mathbf{b}_1^2, \mathbf{b}_2^2$ by

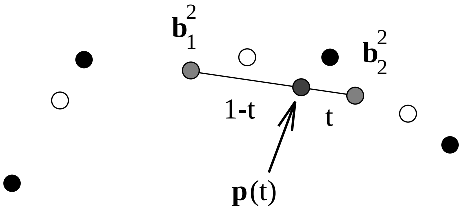
$$\begin{aligned}\mathbf{b}_1^2 &= (1-t)\mathbf{b}_1^1 + t\mathbf{b}_2^1\\ \mathbf{b}_2^2 &= (1-t)\mathbf{b}_2^1 + t\mathbf{b}_3^1.\end{aligned}$$



Bézier: Derivation

Step 3: Find point $\mathbf{p}(t)$ by

$$\mathbf{p}(t) = (1-t)\mathbf{b}_1^2 + t\mathbf{b}_2^2.$$



We are performing repeated linear interpolations.

The superscript j of the $\mathbf{b}_i^j$'s indicates the step number, or *level* of interpolation.

Using different weights t_1, t_2, t_3 at each step yields $\mathbf{p}(t_1, t_2, t_3)$, a function of three arguments: the *blossom*. Lyle will talk about blossoms next Tuesday, hopefully.

Bézier: Formula

Expanding expressions for $\mathbf{b}_i^j$'s we get

$\mathbf{p}(t) = \mathbf{b}_1 B_1(t) + \mathbf{b}_2 B_2(t) + \mathbf{b}_3 B_3(t) + \mathbf{b}_4 B_4(t)$

where $B_i(t)$'s are the *Bernstein polynomials*:

$$\begin{aligned} B_1(t) &= (1-t)^3 \\ B_2(t) &= 3(1-t)^2 t \\ B_3(t) &= 3(1-t)t^2 \\ B_4(t) &= t^3. \end{aligned}$$

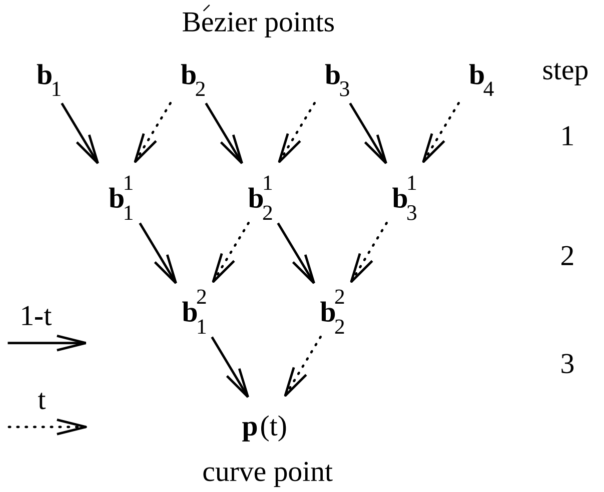
Bézier points: the $\mathbf{b}_i$'s.

Bézier curve: the resulting $\mathbf{p}$(t).

Control polygon: polygon formed by $\mathbf{b}_i$'s.

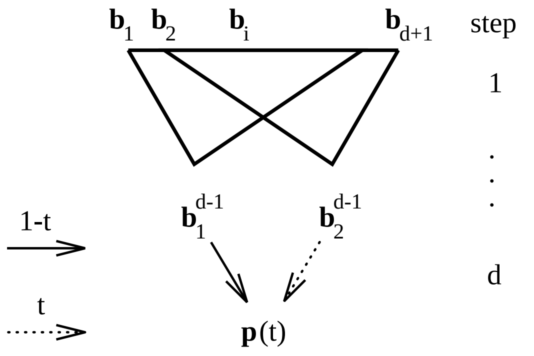
We also use the term *control polygon* for the polygon formed by the interpolating points of a curve.

Bézier: Bernstein Derivation



This is a graphical illustration of the derivation of the Bernstein polynomials. Each level of the tree corresponds to one level of interpolation. Two points from every level are combined to form a point on the next level: we use solid and dashed lines to encode the interpolation weights. Every path from each of the $\mathbf{b}_i$'s to $\mathbf{p}(t)$ indicates a contribution of a $\mathbf{b}_i$ to the final curve point. The arrows encountered along each path encode the total weight of $\mathbf{b}_i$'s contribution to $\mathbf{p}(t)$ via this path. By summing over all the paths from a particular $\mathbf{b}_i$ to $\mathbf{p}(t)$, we find the overall weight of $\mathbf{b}_i$'s contribution, i.e. $B_i(t)$.

Bézier: Bernstein Recursion



Bernstein basis for degree d related to basis for degree $d-1$:

$$B_i^d(t) = (1-t)B_i^{d-1}(t) + tB_{i-1}^{d-1}(t)$$

$\mathbf{b}_i$ influences $\mathbf{p}(t)$ via two different paths: one through $\mathbf{b}_1^{d-1}$ and one through $\mathbf{b}_2^{d-1}$. The influence of $\mathbf{b}_i$ to $\mathbf{b}_1^{d-1}$ is given by $B_i^{d-1}(t)$, from the previous diagram on the derivation of the polynomials. The influence of $\mathbf{b}_i$ to $\mathbf{b}_2^{d-1}$ is given by $B_{i-1}^{d-1}(t)$ from the same diagram, and observing that, in the tree rooted at $\mathbf{b}_2^{d-1}$, $\mathbf{b}_i$ is the $(i-1)$st leaf. By combining the two paths, using the weights of the last interpolation, we obtain the recursion formula.

Bézier: Matrix Form

Since

$$\begin{pmatrix} B_1(t) \\ B_2(t) \\ B_3(t) \\ B_4(t) \end{pmatrix} = \begin{pmatrix} 1 & -3 & 3 & 1 \\ 0 & -3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix},$$

we can write

$$\begin{aligned} \mathbf{p}(t) &= \begin{pmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 & \mathbf{b}_4 \end{pmatrix} \begin{pmatrix} B_1(t) \\ B_2(t) \\ B_3(t) \\ B_4(t) \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 & \mathbf{b}_4 \end{pmatrix} \\ & \quad \begin{pmatrix} 1 & -3 & 3 & 1 \\ 0 & -3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}. \end{aligned}$$

The matrix describes the transformation between the Bernstein basis and the monomial basis.

A row vector multiplied by a column vector yields a scalar. This is the same operation as the dot product, only written as matrix multiplication.

Bézier: Properties

Endpoint interpolation:

$$\mathbf{p}(0) = \mathbf{b}_1$$
$$\mathbf{p}(1) = \mathbf{b}_4.$$

Pseudo-local control:

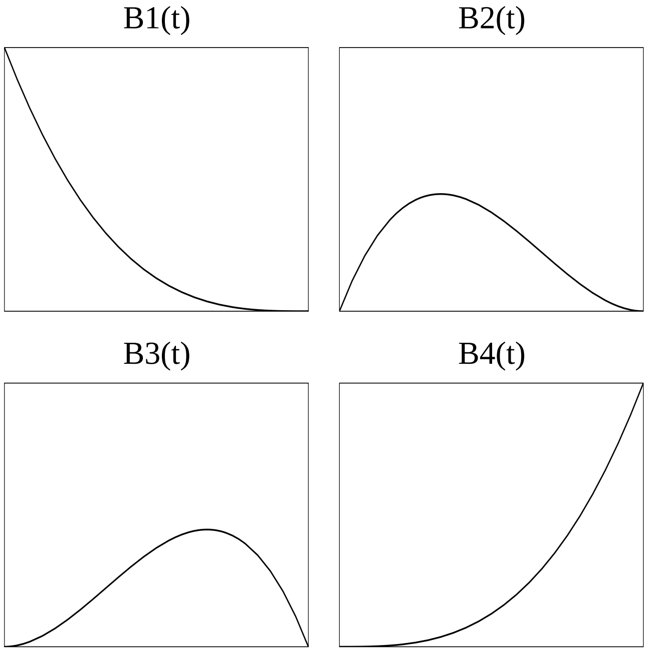
- Moving one control point for interpolating curves changes the whole shape.
- Moving one Bézier point changes the curve *mostly* near that point.

Why? Because of the form of the $B_i(t)$'s...

The intuition behind $\mathbf{b}_1$ and $\mathbf{b}_4$ is that they are the endpoints of the curve.

Interpolating curves are *awful* in terms of local control. They become impossible to use if we want to model a complex shape (in which case we need the pliability of a high degree polynomial): if we move one point here, the curve gets wiggles all over the place. We will show an example of this later.

Bézier: Bernstein Basis

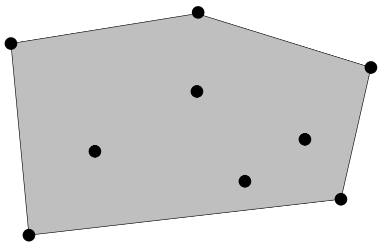


The horizontal axis is t and the vertical is $B_i(t)$. The ranges on all the axes are from 0 to 1.

The Bernstein polynomial $B_i(t)$ encodes the contribution, or influence, of the Bézier point $\mathbf{b}_i$ to the curve $\mathbf{p}(t)$. Observe how, for t near 0, $B_1(t)$ is much larger than the other polynomials. As t increases, each of the other $B_i(t)$'s gets to dominate for some interval of values for t. Hence for different ranges of t, different Bézier points attract the curve. It follows that moving one Bézier point $\mathbf{b}_i$ will mostly influence the curve for the range of times during which $B_i(t)$ is dominant: this is *local* control.

Bézier: Convex Hull

Convex hull of point set: tightly wrapped balloon, hinged at given points, e.g.



Convex hull property: for $t \in [0, 1]$, $\mathbf{p}(t)$ lies in the convex hull of control polygon. Why?

- $1 = [(1-t) + t]^3 = B_1(t) + \cdots + B_4(t)$.
- $B_i(t)$'s are positive if $t \in [0, 1]$.

Or, by construction: interpolations with $t \in [0, 1]$ only produce points between given ones!

Recall that the site notation requires that the basis polynomials sum to one.

This expansion also shows (in hand-waving terms) how the de Casteljau algorithm yields the Bernstein polynomials. Each control point is weighted by a combination of t's and $(1-t)$'s. Each Bernstein polynomial counts one of the possible ways that 3 steps of interpolation, each by t or $1-t$, can be combined; see the earlier tree diagram on the derivation of the Bernstein polynomials. But this is exactly what the terms in the expansion of $1 = [(1-t) + t]^3$ stand for: the possible combinations of the two terms t and $1-t$.

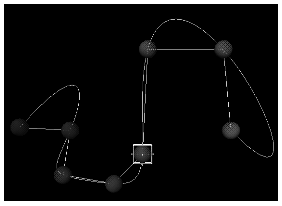
The same expansion suggests how we can find the Bernstein polynomial for degree d Bézier curves: just expand $[(1-t) + t]^d$!

Bézier: Implications

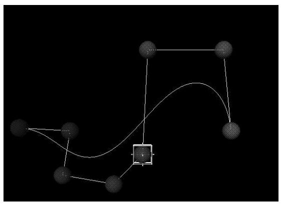
Interference checking more efficient: check control polygon or *minmax* boxes first.

Control polygon approximates shape of curve:

- Interpolating curve:



- Bézier curve:



Interference checking is very common in ray tracing and collision detection. The optimization offered here has as follows: to check if two curve segments intersect, check their control polygons first; if they do not, the curves cannot intersect! An easier check, avoiding the polygon intersection, uses the minmax boxes, i.e. boxes whose diagonal starts at $(\min x, \min y)$ and ends at $(\max x, \max y)$. The minmax box is the smallest possible boxes around a curve segment.

Observe in the drawings the aforementioned wiggles on the interpolating curve of high degree.

Bézier: Affine Invariance

Let

$$\mathbf{q}(t) = \mathbf{A}\mathbf{p}(t) + \vec{d}$$

be *affine* transformation of Bézier curve. Then

$$\begin{aligned}
\mathbf{q}(t) &= \mathbf{A}(\mathbf{b}_1 B_1(t) + \cdots + \mathbf{b}_4 B_4(t)) + \vec{d} \\
&= \mathbf{A}\mathbf{b}_1 B_1(t) + \cdots + \mathbf{A}\mathbf{b}_4 B_4(t) \\
&\quad + (B_1(t) + \cdots + B_4(t))\vec{d} \\
&= (\mathbf{A}\mathbf{b}_1 + \vec{d})B_1(t) + \cdots + (\mathbf{A}\mathbf{b}_4 + \vec{d})B_4(t).
\end{aligned}$$

So $\mathbf{q}(t)$ is a Bézier curve with control points the images of $\mathbf{p}(t)$'s control points.

$\vec{d}$ is the translation. Usually, it is denoted by $\vec{t}$, but this notation would be confusing here, since t denotes time.

This property is very important for warping: to draw the transformed curve, we don't need to transform every point on the original one (an expensive operation, requiring lots of multiplications): we just need to transform the control points, and apply de Casteljau's construction.

Unfortunately, This property does not hold for projective transformations, which include viewing transformations.

Bézier: Derivative

$$\begin{aligned}
\mathbf{p}'(t) &= \mathbf{b}_1 B_1'(t) + \mathbf{b}_2 B_2'(t) + \mathbf{b}_3 B_3'(t) + \mathbf{b}_4 B_4'(t) \\
&= -\mathbf{b}_1 3(1-t)^2 \\
&\quad -\mathbf{b}_2 6t(1-t) + \mathbf{b}_2 3(1-t)^2 \\
&\quad +\mathbf{b}_3 6t(1-t) - \mathbf{b}_3 3t^2 \\
&\quad +\mathbf{b}_4 3t^2 \\
&= 3(1-t)^2(\mathbf{b}_2 - \mathbf{b}_1) + 3t^2(\mathbf{b}_4 - \mathbf{b}_3) \\
&\quad +6t(1-t)(\mathbf{b}_3 - \mathbf{b}_2).
\end{aligned}$$

So

$$\begin{aligned}
\mathbf{p}'(0) &= 3(\mathbf{b}_2 - \mathbf{b}_1) \\
\mathbf{p}'(1) &= 3(\mathbf{b}_4 - \mathbf{b}_3).
\end{aligned}$$

$\mathbf{b}_2$ and $\mathbf{b}_3$ define tangent of curve!

See the earlier drawing of a Bézier curve.

Hermite: Derivation

Define curve by

- Endpoints $\mathbf{p}_1$, $\mathbf{p}_2$.
- Tangents $\vec{m}_1$, $\vec{m}_2$.

Derive formula from Bézier curve:

$$\begin{aligned}
\mathbf{p}(0) &= \mathbf{b}_1 = \mathbf{p}_1 \\
\mathbf{p}(1) &= \mathbf{b}_4 = \mathbf{p}_2 \\
\mathbf{p}'(0) &= 3(\mathbf{b}_2 - \mathbf{b}_1) = \vec{m}_1 \\
\mathbf{p}'(1) &= 3(\mathbf{b}_4 - \mathbf{b}_3) = \vec{m}_2.
\end{aligned}$$

Hermite: Derivation

Carrying on...

$$\begin{aligned}
\mathbf{b}_1 &= \mathbf{p}_1 \\
\mathbf{b}_4 &= \mathbf{p}_2 \\
\mathbf{b}_2 &= \mathbf{p}_1 + \vec{m}_1/3 \\
\mathbf{b}_3 &= \mathbf{p}_2 - \vec{m}_2/3.
\end{aligned}$$

So

$$\begin{aligned}
\mathbf{p}(t) &= \mathbf{p}_1 B_1(t) \\
&\quad + \mathbf{p}_1 B_2(t) + \vec{m}_1 B_2(t)/3 \\
&\quad + \mathbf{p}_2 B_3(t) - \vec{m}_2 B_3(t)/3 \\
&\quad + \mathbf{p}_2 B_4(t) \\
&= \mathbf{p}_1(B_1(t) + B_2(t)) \\
&\quad + \vec{m}_1 B_2(t)/3 \\
&\quad - \vec{m}_2 B_3(t)/3 \\
&\quad + \mathbf{p}_2(B_3(t) + B_4(t)).
\end{aligned}$$

Hermite: Basis

Finally, setting

$$\begin{aligned}
H_1(t) &= B_1(t) + B_2(t) \\
H_2(t) &= B_2(t)/3 \\
H_3(t) &= -B_3(t)/3 \\
H_4(t) &= B_3(t) + B_4(t),
\end{aligned}$$

we obtain

$$\mathbf{p}(t) = \mathbf{p}_1 H_1(t) + \vec{m}_1 H_2(t) + \vec{m}_2 H_3(t) + \mathbf{p}_2 H_4(t).$$

Recall that the site notation requires that $H_1(t)+H_4(t) = 1$, which indeed is the case! Remember this as a useful sanity check...

References

- Böhm: sections 4–8.
- Farin: pages 25–27, 29–36, 41–47, 58–59, 86–95.
- `polly`: curve visualization and manipulation software.

Where are they?

- Course reader.
- Library reserve.
- Available on the WWW. It is a software package I wrote to understand curves. I showed earlier a screen dump of `polly`. Its features include interpolating and Bézier curves and animation of a particle moving on a curve. Play with it to see the power of individual cubics, the most common building block of splines.