
Modeling curves

© 1999, Denis Zorin

Types of Curves

- Graphs: $y = ax+b$, $y = \sin(x)$
- Implicit $ax + by + c = 0$, $x^2+y^2=r^2$
- Parametric:

$$x = a_x + b_x t$$

$$x = \cos t$$

$$y = a_y + b_y t$$

$$y = \sin t$$

Parametric are the most common implicit
are also used,

Graphs: not flexible enough (e.g. cannot make a
circle).

© 1999, Denis Zorin

Parametric curves

Parametric curve can be thought of as a deformation of a straight line. If t is the distance from some fixed point of the line, for each value of t we specify the corresponding point in the plane (or space):

$$x = x(t); y = y(t); z = z(t)$$

How do we choose functions $x(t)$ and $y(t)$?

© 1999, Denis Zorin

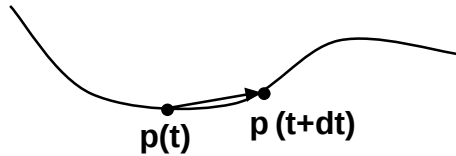
Parametric curves

Typical requirements:

- should be easy to use for modeling, that is, making any desired shape
- should be smooth (intuition: if we zoom in should look like a straight line), or piecewise smooth
- should be easy to draw

© 1999, Denis Zorin

Tangents



Derivatives of the coordinates define the tangent:

$$\text{tangent} = \frac{dp(t)}{dt} = \left(\frac{dx(t)}{dt}, \frac{dy(t)}{dt} \right)$$

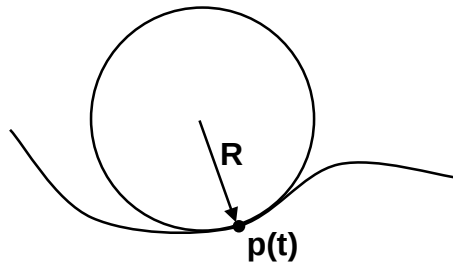
Note: the length of the tangent does not have a geometric meaning, it depends on parameterization.

© 1999, Denis Zorin

Curvature

Intuition: inverse of the radius of the best local approximation of the curve by a circle.

Formula: $\frac{x'y'' - x''y'}{(x'^2 + y'^2)^{3/2}}$ the derivatives are with respect to t .



Curvature at t is $1/R$

© 1999, Denis Zorin

Curvature

Curvature measures how straight a curve is at a point. If curvature is zero everywhere, the curve is a straight line.

If $x = t$, curvature is $\frac{y''}{(1 + y'^2)^{3/2}}$; for small y' , this is approximately the second derivative.

The square of the curvature can be (sometimes) approximated by $x''^2 + y''^2$.

The less is the curvature, the “nicer” is the curve.

© 1999, Denis Zorin

Modeling

We should be able to specify a small number of intuitive controls;

the curve should be automatically computed from controls.

Examples of intuitive controls:

- “Connect the dots” approach: specify several points on the curve, draw a nice line through these points;
- Specify the direction (tangent) of the curve in some places

Unintuitive controls: polynomial coefficients etc.

© 1999, Denis Zorin

Real-life splines

How was it done before computers?

- weights (called ducks) were placed on a sheet of paper at points through which the curve is supposed to pass
- A flexible steel strip is placed between them
- A curve is traced out on paper.

Idea: try to do something similar;

while modeling physics is possible, it is somewhat expensive. Try to imitate behavior qualitatively.

© 1999, Denis Zorin

Control points

One possible way of defining parametric curves:

Define points *near* which the curve should pass:

$p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$...

For each point p_i choose a parameter value t_i .

Simplest choice: 1,2,3,... (uniform parameterization).

For each point specify a blending function $B_i(t)$, which defines how much it contributes to the value of the curve at t .

Define the curve as

$$\begin{aligned} x(t) &= \sum_i x_i B_i(t) \\ y(t) &= \sum_i y_i B_i(t) \end{aligned} \quad \text{or} \quad p(t) = \sum_i p_i B_i(t)$$

© 1999, Denis Zorin

Blending functions

Requirements for blending (basis) functions $B_i(t)$:

- The control point p_i has maximal influence near parameter value i ; the further we are from i , the less B_i should be; if we want local control (curve does not change far enough from a control point) $B_i(t)$ should be zero for t far from i . Should look like this:



- If all control points are the same, the resulting curve should be reduced to the same point.

$$\sum_i p_i B_i(t) = p \sum_i B_i(t) = p, \text{ so } \sum_i B_i(t) = 1$$

© 1999, Denis Zorin

Blending functions

Requirements

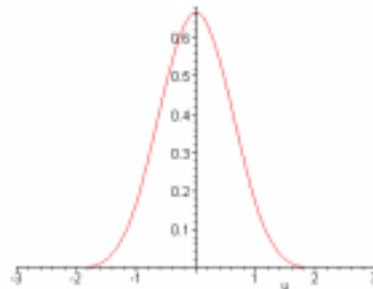
- Should be easy to compute (polynomials are);
- Should be continuous, and differentiable (C^1), preferably twice (C^2).
- Support (the interval where it is not zero) should not be too big.
- Would be nice to interpolate the control points.

Cannot get it all! But can get close if relax one of the requirements.

© 1999, Denis Zorin

Splines

If give up on small support, get natural splines;
every control point influences the whole curve.
If give up on interpolation, get cubic B-splines.



Cubic B-spline basis function

© 1999, Denis Zorin

B-Splines

Need only one basis function, all $B_i(t)$ are obtained
by shifts: $B_i(t) = B(t-i)$.

The basis function is piecewise polynomial:

$$B(t) = \begin{cases} 0 & t \leq -2 \\ \frac{1}{6}t^3 + 2t + \frac{4}{3}t^2 & t \leq -1 \\ \frac{2}{3} - t^2 - \frac{1}{2}t^3 & t \leq 0 \\ \frac{2}{3} - t^2 + \frac{1}{2}t^3 & t \leq 1 \\ \frac{4}{3} - 2t + t^2 - \frac{1}{6}t^3 & t \leq 2 \\ 0 & 2 < t \end{cases}$$

© 1999, Denis Zorin

B-splines

The curve with control points $p_0, p_1, p_2, p_3 \dots p_n$ is computed using

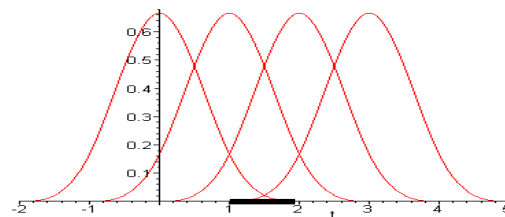
$$p(t) = \sum_{i=0}^n p_i B(t - i)$$

The allowed range of t is from 1 to $n-1$; outside this interval our functions do not sum up to 1, which means in particular that if we move control points together in the same way, the curve outside the interval will not move rigidly.

© 1999, Denis Zorin

B-splines

The minimal number of points required is 4; this corresponds to the interval for t of length 1.

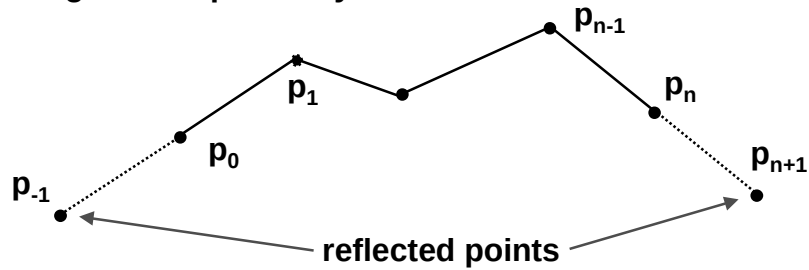


This is inconvenient - but we can always add control points by reflection.

© 1999, Denis Zorin

B-splines

Adding control points by reflection:



$$p_{-1} = 2p_0 - p_1; \quad p_{n+1} = 2p_n - p_{n-1}.$$

© 1999, Denis Zorin

Drawing B-splines

Hardware typically can draw only line segments.

Need to approximate B-spline with piecewise linear curve. Simplest approach:

Choose small Δt .

Compute points $p(0)$, $p(\Delta t)$, $p(2\Delta t)$...

Draw line segments connecting the points.

Not very efficient -- have to evaluate a cubic polynomial (or several) at each point.

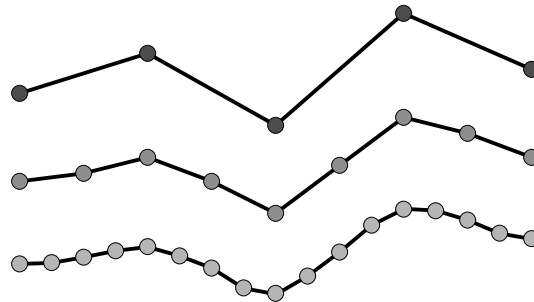
Can do better using a magic algorithm (subdivision).

© 1999, Denis Zorin

B-splines via subdivision

It turns out that the smooth curve can be obtained by subdivision of the original polyline:

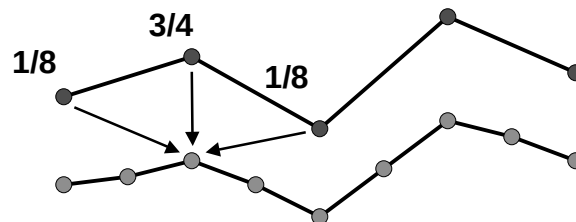
Subdivision adds new control points between the original control points and updates positions of original control points.



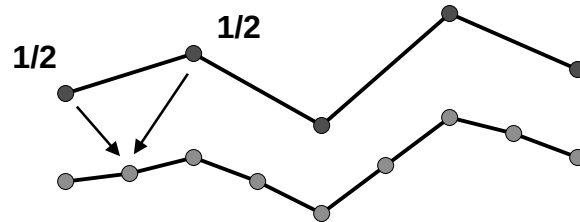
© 1999, Denis Zorin

Subdivision

Subdivision rules for updating old points:



Subdivision rules for inserting new points:

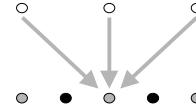


© 1999, Denis Zorin

Subdivision rules

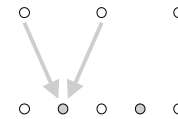
Even rule (in the new sequence of points the points with even numbers are the old points with updated positions).

$$p_{2i}^{j+1} = 1/8(p_{i-1}^j + 6p_i^j + p_{i+1}^j)$$



Odd rule (in the new sequence the points with odd number is a newly inserted point).

$$p_{2i+1}^{j+1} = 1/8(4p_i^j + 4p_{i+1}^j)$$



© 1999, Denis Zorin

Subdivision

Of course, in a finite number of steps subdivision generates only polylines. But they get arbitrarily close to a limit C^2 and this curve is exactly a cubic B-spline.

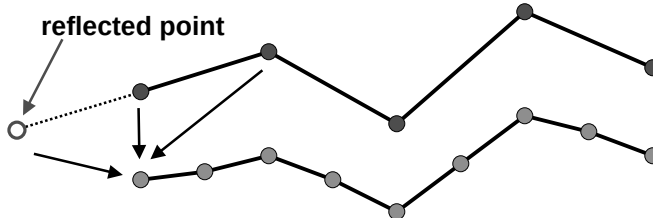
Algorithm:

Start with an array of control points of length $n+1$. Compute from the original points new array of control points of length $2n+1$ using subdivision rules for even and odd points. Then from the new array compute an array of length $4n+1$ etc., (typically 4-5 steps is enough). Then draw the line segments connecting sequential control points.

© 1999, Denis Zorin

Endpoints

What do we do when a point is missing?



In this case, we apply reflection (recall adding missing control points). This results in the following trivial rule:

$$p_0^{j+1} = 1/8(p_1^j + 6p_0^j + (2p_0^j - p_1^j)) = p_0^j$$

that is, just keep the old value.

© 1999, Denis Zorin

Other splines

There many types of splines:

Bezier,
Hermite,
Natural,
Catmull-Rom,

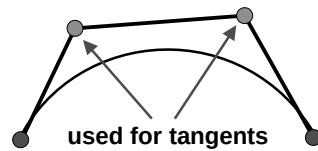
...

B-splines are the most useful, but Bezier and Hermite are used widely. Bezier are often found in drawing programs (e.g. Powerpoint!) although they are not very convenient.

© 1999, Denis Zorin

Bezier splines

Construct a curve out of segments. For each segment define 4 points: two endpoints and two other points determining the tangents at endpoints:



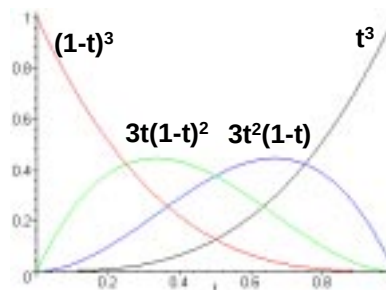
If we ensure that blue points on two sides of each read point are on one line, the whole curve is C^1 , that is, has a tangent everywhere.

© 1999, Denis Zorin

Bezier splines

The idea behind Bezier splines is to use special blending functions for the four points, assuming that t is between 0 and 1:

$(1-t)^3$ for p_0 , $3t(1-t)^2$ for p_1 , $3t^2(1-t)$ for p_2 , and t^3 for p_3 .



© 1999, Denis Zorin

Bezier splines

The 4 functions (Bernstein polynomials of order 3), $P^0(t)$, $P^1(t)$, $P^2(t)$, $P^3(t)$ are (almost) defined by the following properties:

- The sum is 1 for all t
- the endpoints have no influence on the position of the endpoint of the curve and on the tangents at opposite endpoints. This means that $P^0(1) = P^0(1)' = 0$, $P^3(0) = P^3(0)' = 0$
- Two interior (blue) control points affect only the tangent and at the corresponding endpoint, not at the other endpoint, and do not affect the positions of the endpoints of the curve
 $P^1(0) = P^1(1) = P^1(1)' = 0$, $P^2(1) = P^2(0) = P^2(0)' = 0$
- All polynomials are cubic and positive
- Symmetry w.r.t. $t=1/2$: $P^0(1-t) = P^3(t)$, $P^1(1-t) = P^2(t)$

© 1999, Denis Zorin

Bezier curves

The Bezier curves have the following properties:

- endpoints are interpolated
- tangents at endpoints point towards the interior control points;
derivatives are $3(p_1 - p_0)$ and $3(p_3 - p_2)$ respectively;
- the curve is in the convex hull of control points.

The formula for the Bezier curve is

$$p(t) = (1-t)^3 p_0 + 3t(1-t)^2 p_1 + 3t^2(1-t) p_2 + t^3 p_3$$

t between 0 and 1.

© 1999, Denis Zorin