parametric spline curves

curves

• used in many contexts
  – fonts
  – animation paths
  – shape modeling

• different representation
  – implicit curves
  – parametric curves
    • mostly used
**implicit representation for 2D curves**

- curves can be represented implicitly as
  \[ f(p) = f(x, y) = 0 \]

- example: circle of radius \( r \) centered at origin
  \[ x^2 + y^2 - r^2 = 0 \]

---

**parametric representation for 2D curves**

- curves can be represented parametrically as
  \[
  p(u) = \begin{cases} 
  x = f_x(u) \\
  y = f_y(u) 
  \end{cases}
  \]

- example: circle of radius \( r \) centered at origin
  \[
  \begin{cases} 
  x = r \cos(u) \\
  y = r \sin(u) 
  \end{cases}
  \]
parametric representation of splines

• general parametric curve can be written as
  \[ p(t) = f(t) \quad t \in [0, N] \]

• goals when defining \( f \)
  – smoothness
  – predictable and local control
  – efficiency

parametric representation of splines

• splines: piecewise parametric polynomials
  – polynomials are smooth
  – controlled by small number of local control points
  – discontinuities at integer intervals

\[ p(t) = f(t) \quad t \in [0, N] \]
splines - intuition

• define segment by “blending” *control points*

• join segments to form curve

defining splines

• pick segment interpolating function
• impose constraints to define segments
  – i.e. control points that define the spline
• impose constraints to join segments together
interpolating vs. approximating splines

- interpolating
  - pass through control points
- approximating
  - guided by control points

smoothness

- smoothness described by degree of continuity
  - $C^0$: same position at each side of joints
  - $C^1$: same tangent at each side of joints
  - $C^2$: same curvature at each side of joints
  - $C^n$: n-th derivative defined at joints
**control**

- **local control**
  - changing control points only affect locally the curve
    - easy to control
  - true for all splines

**control**

- **convex hull property**
  - convex hull: smallest convex region enclosing all points
    - predictable behavior
    - more efficient operations
  - only some splines

---

*computer graphics • parametric curves © 2009 fabio pellacini • 11

*computer graphics • parametric curves © 2009 fabio pellacini • 12*
efficiency

• affine invariance
  – transforming the spline same as transforming controls
  – efficient algorithms, esp. combined with convex hull
  – true for all used splines

piecewise linear splines

• each segment is a linear function
  \[ p(t) = ta + b \quad t \in [0,1] \]

• impose endpoint constraints
  \[
  \begin{cases}
  p(0) = p_0 \\
  p(1) = p_1 
  \end{cases} \Rightarrow \begin{cases}
  a = p_1 - p_0 \\
  b = p_0 
  \end{cases}
  \]
  \[ p(t) = p_0 + t(p_1 - p_0) \quad t \in [0,1] \]
point blending interpretation

• can interpret as blending of points
  \[ p(t) = (1-t)p_0 + tp_1 = b_0(t)p_0 + b_1(t)p_1 \quad t \in [0,1] \]

• blending functions do not depend on points
  – different intervals only change control points

![Graph showing point blending]

matrix notation

• write blending functions more conveniently
  \[ p(t) = (1-t)p_0 + tp_1 \]
  \[ p(t) = \begin{bmatrix} t & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \end{bmatrix} \]
joining line segments

- impose $C^0$ continuity at joints
  - first segment
    \[ p^0(t) \rightarrow \begin{cases} p^0(0) = p_0 \\ p^0(1) = p_1 \end{cases} \]
  - second segment
    \[ p^1(t) \rightarrow \begin{cases} p^1(0) = p_0 \\ p^1(1) = p_1 \end{cases} \]
  - implies
    \[ p^0(1) = p^1(0) \rightarrow p^0_1 = p^1_0 \]

- general formula
  - appropriately rename control points
    \[ p(t) = b_0(t-k)p_k + b_1(t-k)p_{k+1} \quad t \in [0,N], k = \text{floor}(t) \]

Hermite splines

- each segment is a cubic polynomial function
  \[ p(t) = at^3 + bt^2 + ct + d \]
- impose endpoints and tangents constraints
  \[ \begin{cases} p(0) = p_0 \\ p(1) = p_1 \\ p'(0) = p'_0 \\ p'(1) = p'_1 \end{cases} \]
Hermite splines

\[ p(t) = at^3 + bt^2 + ct + d \]
\[ p'(t) = 3at^2 + 2bt + c \]

\[ p(0) = d \]
\[ p(1) = a + b + c + d \]
\[ p'(0) = c \]
\[ p'(1) = 3a + 2b + c \]

\[ a = 2p_0 - 2p_1 + p'_0 + p'_1 \]
\[ b = -3p_0 + 3p_1 - 2p'_0 - p'_1 \]
\[ c = p'_0 \]
\[ d = p_0 \]

Hermite

- matrix formulation

\[ p(t) = \begin{bmatrix} t^3 & t^2 & t^1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 2 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p'_0 \\ p'_1 \end{bmatrix} \]

- blending functions
Beziers Splines

- Hermite splines has points and vectors controls
  - would like to use just points
  - insight: specify tangents as difference of points
    - choose appropriate scaling value, see later

\[
\begin{bmatrix}
0 & 0 & 0 & 1 \\
3 & 0 & 0 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
q_0 \\
q_1 \\
q_2 \\
q_3 \\
\end{bmatrix}
= \begin{bmatrix}
p_0 \\
p_1 \\
p_2 \\
p_3 \\
\end{bmatrix}
\]

Beziers Splines

\[
p(t) = \begin{bmatrix}
t^3 \\
t^2 \\
t \\
1 \\
\end{bmatrix}
\begin{bmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
q_0 \\
q_1 \\
q_2 \\
q_3 \\
\end{bmatrix}
\]
**Bezier splines**

- blending functions

![Bezier splines diagram]

**piecewise cubic splines – smoothness**

- \( C^1 \) at joints by imposing equal tangents
  - Hermite: same tangents
  - Bezier: collinear control points
    - geometric continuity if length of tangent differs

![Piecewise cubic splines diagram]
### piecewise cubic splines – control

- **local control**
  - comes from the formulation by segments
  - for each segment, curve defined by 4 control points

- **convex hull**
  - when blending positions
    
    \[ b_i(t) \geq 0 \quad \text{and} \quad \sum_{i=0}^{3} b_i(t) = 1 \]

### piecewise cubic splines – affine invariance

- **affine invariance**
  - affine is combination of linear and translation
  - blending functions sum to 1

\[
X(p(t)) = M p(t) + t = M \left( \sum_{i=0}^{3} b_i(t) p_i \right) + t = \\
= \sum_{i=0}^{3} b_i(t) M p_i + \sum_{i=0}^{3} b_i(t) t = \\
= \sum_{i=0}^{3} b_i(t) (M p_i + t) = \sum_{i=0}^{3} b_i(t) X(p_i)
\]
Beziers splines

- widely used, especially in 2D
  - primitive in PDF
- represent $C^1$ and $C^0$ curves with corners
- easily add point at any position

Catmull-Rom splines

- interpolating spline
  - no convex hull property
- as Hermite, derivatives automatically determined
  - using adjacent control points
  - end tangent using either adding point or zers
Catmull-Rom splines

\[ p'_k = \frac{p_{k+1} - p_{k-1}}{2} \]

drawing splines

- approximate with a sequence of line segments
  - efficiency: fast evaluation, small number of segments
  - guarantees on accuracy
- approaches
  - uniform subdivision in \( t \) (fast)
  - recursive subdivision (small number of segments)
**uniform subdivision**

- evaluate spline at fixed $t$ intervals
  - can be done efficiently

**adaptive subdivision - Bezier**

- recursively subdivide spline
- until line segments approximate well curve
De Casteljau algorithm - Bezier

- recursively do
  - connect midpoints of the control polygons
  - connect midpoints of the new segments
  - the midpoint of this last segment is on the curve
  - and splits the curve in two Bezier segments
- stop when control polygon is close to collinear

B-Splines

- would like $C^2$ continuity at joints
  - give up interpolation
- impose 3 continuity constraints at joints

$$p(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} q_{k-1} \\ q_k \\ q_{k+1} \\ q_{k+2} \end{bmatrix}$$
other splines

• many other types

• non-uniform B-splines
  – discontinuities not evenly spaces

• non-uniform rational B-splines (NURBS)
  – ratios of non-uniform B-splines
  – invariance under perspective
  – can represent conic sections exactly
  – often used in 3D

spline equivalence

• all splines seen so far are equivalent
  – represented by 4x4 matrices

• can convert control points from one to other
  – algorithms can be based on the most efficient
  – UIs can be based on the most user-friendly
2D vs. 3D splines

- often use 2D splines in 3D
  - by projecting onto a plane
- 3D parametric splines have same formulation
  - just use 3D vectors vs. 2D ones