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

Splines - Intuition

- define segment by “blending” control points

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

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

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

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

yes

no
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
  \[
  \begin{align*}
  p(0) &= p_0 \\
  p(1) &= p_1 \\
  \end{align*}
  \]
  \[ a = p_1 - p_0 \]
  \[ b = p_0 \]
  \[ p(t) = p_0 + t(p_1 - p_0) \quad t \in [0,1] \]

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

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 \( p^0(t) \):
    \[
    \begin{align*}
    p^0(0) &= p_0^0 \\
    p^0(1) &= p_1^0
    \end{align*}
    \]
  - second segment \( p^1(t) \):
    \[
    \begin{align*}
    p^1(0) &= p_0^1 \\
    p^1(1) &= p_1^1
    \end{align*}
    \]
  - implies \( p^0(1) = p^1(0) \)
    \[
    p^1_0 = p_0^0
    \]
- 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
  \[
  p(t) = at^3 + bt^2 + ct + d
  \]
  \[
  p'(t) = 3at^2 + 2bt + c
  \]
- impose endpoints and tangents constraints
  \[
  \begin{align*}
  p(0) &= p_0 \\
  p(1) &= p_1 \\
  p'(0) &= p'_0 \\
  p'(1) &= p'_1
  \end{align*}
  \]

Hermite

- matrix formulation
  \[
  \begin{bmatrix}
  2 & -2 & 1 & 2 \\
  -3 & 3 & -2 & -1 \\
  0 & 0 & 1 & 0 \\
  1 & 0 & 0 & 0
  \end{bmatrix}
  \begin{bmatrix}
  p_0 \\
  p_1 \\
  p'_0 \\
  p'_1
  \end{bmatrix}
  \]
- blending functions
Beziers Splines

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

\[
\begin{bmatrix}
q_0 \\
q_1 \\
q_2 \\
q_3
\end{bmatrix} = \begin{bmatrix}
1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} \begin{bmatrix}
p_0 \\
p_1 \\
p_2 \\
p_3
\end{bmatrix}
\]

Piecewise Cubic Splines - Smoothness

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

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

**Bezier 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 zero
**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 Subvision - 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

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

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