parametric spline curves
curves

- used in many contexts
  - fonts (2D)
  - animation paths (3D)
  - shape modeling (3D)
- different representation
  - implicit curves
  - parametric curves (mostly used)
- 2D and 3D curves are mostly the same
  - use vector notation to simplify treatment
implicit representation

- implicit curve representation

\[ f(P) = 0 \]

- e.g. XY circle: \( f(P) = 0 \rightarrow x^2 + y^2 - r^2 = 0 \)
parametric representation

- parametric curve representation

\[ \mathbf{P}(u) = (f_x(u), f_y(u), f_z(u)) \]

- e.g. XY circle: \( \mathbf{P}(u) = (r\cos \theta, r\sin \theta, 0) \)
parametric representation

- goals when defining $f$
  - smoothness
  - efficiency
  - local control
    - curve controls only affect a piece of the curve
  - predictable behaviour
    - easy to use for designers
splines
splines

- splines: piecewise parametric polynomials
  - split curve $\mathbf{P}(t)$ into $N$ segments $\mathbf{P}^s(t - t^s)$ at intervals $t^s$
  - uniform splines: split at integer intervals $t_s \in 0, 1, 2 \cdots$
  - each segment $\mathbf{P}^s$ is a polynomial: smooth and efficient
  - piecewise curve, i.e. splitting: local control

$$\mathbf{P}(t) = \mathbf{P}^s(t - t^s) \text{ with } t \in [0, N), t - t^s \in [0, 1)$$
splines

- intuition: define segment by "blending" control points

- intuition: join segments to form a curve
Splines

- approximating: guided by control points

- interpolating: pass through control points
defining splines

• pick segment interpolating function
  ◦ smoothness

• pick segment control points
  ◦ local control

• impose constraints to define segments
  ◦ join segments together
  ◦ ensure smoothness
defining splines

- 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
defining splines

- local control: control points affects the curve locally
  - easy to control, true for all splines
defining splines

- convex hull: smallest convex region enclosing all points
  - predictable and efficient, but only some splines
defining splines

- affine invariance: transform controls equiv. transform spline
  - efficient, all splines
linear splines
linear splines

- segment: linear function

\[ P(t) = ta + b \text{ with } t \in [0, 1) \]

- control points: end points \( P(0) = P_0 \) and \( P(1) = P_1 \)

\[ P(t) = (1 - t)P_0 + tP_1 \]
linear splines

- blending functions: interpret as blending control points

\[ \mathbf{P}(t) = b_0(t)\mathbf{P}_0 + b_1(t)\mathbf{P}_1 \]

\[ b_0(t) = (1 - t) \]

\[ b_1(t) = t \]
linear splines

- joining segments: impose $C^0$ continuity
  - segments share endpoints
- $C^1$ continuity only for straight lines

\[ P_0^0(1) = P_1^1(0) \rightarrow P_1^0 = P_0^1 \]
bezier cubic splines
bezier cubic splines

- segment: cubic function

\[ P(t) = t^3 a + t^2 b + tc + d \text{ with } t \in [0, 1) \]

- control points: end points \( P_0, P_3 \) and tangents

\[ P'(0) \propto P_1 - P_0 \text{ and } P'(1) \propto P_2 - P_3 \]
bezier cubic splines

- blending functions: Bernstein polynomials

\[ P(t) = b_0(t)P_0 + b_1(t)P_1 + b_2(t)P_2 + b_3(t)P_3 \]

\[ b_0(t) = (1 - t)^3 \]
\[ b_1(t) = 3t(1 - t)^2 \]
\[ b_2(t) = 3t^2(1 - t) \]
\[ b_3(t) = t^3 \]
bezier cubic splines

- joining segments: impose $C^0$ continuity
  - segments share endpoints
- $C^1$ continuity by collinear tangents

\[ P^0(1) = P^1(0) \rightarrow P^0_3 = P^1_0 \]
bezier cubic splines

- properties: local control
  - comes from the formulation by segments
  - for each segment, curve defined by 4 control points
bezier cubic splines

- properties: convex hull
  - each segment is convex sum of control points
  - since $b_i(t) \geq 0$ and $\sum_i b_i(t) = 1$
bezier cubic splines

- properties: affine invariance

\[ X(P(t)) = MP(t) + t = \]
\[ = M \left( \sum_i b_i(t)P_i \right) + t = \]
\[ = \sum_i b_i(t)MP_i + \left( \sum_i b_i(t) \right)t = \]
\[ = \sum_i b_i(t)(MP_i + t) = \sum_i b_i(t)X(P_i) \]
rendering splines
rendering splines

- tessellation: approximate splines with line segments
  - segments are efficient to draw in hardware and software
  - more segments to provide better approximation
- uniform tessellation: split $t$ interval uniformly
  - fast to compute and simple to implement
  - generates many segments
- adaptive tessellation: split recursively until good enough
  - more complex to implement
  - fewer segments with guaranteed approximation
uniform tesselation

- split into $K$ segments uniformly at $t_k = 1/k$
adaptive tessellation

- De Casteljau algorithm: recursively split to small splines
  - if flat enough: draw control segments, otherwise
  - split each control segment: $Q_i = (P_i + P_{i+1})/2$
adaptive tessellation

- De Casteljau algorithm: recursively split to small splines
  - split new segments: $R_i = (Q_i + Q_{i+1})/2$
  - split again: $S = (R_i + R_{i+1})/2$
adaptive tesselation

• De Casteljau algorithm: recursively split to small splines
  ○ two Bezier splines: \( \{P_0, Q_0, R_0, S\}, \{S, R_1, Q_2, P_3\} \)
  ○ recurse algorithm as above
other splines
other cubic splines types

- Bezier splines
  - most used today (2D APIs, PDF, fonts)
- Hermite splines: approximating
  - control: end points and tangents
- Catmull-Rom splines: interpolating
  - used very little
- B-splines: $C^2$ continuity at joints
  - impose 3 continuity constraints
- cubic splines can be converted into one another by changing control points
other spline degree

- linear and cubic most used
- can define splines for other polynomial degree
- e.g. Beziers use Bernstein polynomials of degree $n$
other spline functions

- uniform splines: split segments at integers
- non-uniform splines: split segments at arbitrary points
- non-uniform rational B-splines (NURBS)
  - ratios of B-splines
  - invariance under perspective
  - can represent conic sections exactly
  - often used in 3D