Today

• Review of Bézier curves
• Piecewise-cubic Bézier curves
• B-splines
• Surface patches

Curves

• Construct a function \( x(t) \)
  - Moves a point from start to end of curve as \( t \) goes from \( 0 \) to \( 1 \)
  - Tangent vector given by derivative \( x'(t) \)
• Use a few control points to intuitively describe a curve
• Bézier curves

Linear interpolation

• Given two points \( p_0 \) and \( p_1 \)
• “Curve” is line segment between them

Cubic Bézier curve

• Four control points \( p_0, p_1, p_2, p_3 \)
  - Interpolates the endpoints
  - Intermediate points determine tangents at endpoints
• Recursive geometric construction
  - de Casteljau algorithm
• Three ways to express curve
  - Weighted average of the control points, Bernstein polynomials
  - Polynomial in \( t \)
  - Matrix form

Cubic Bernstein polynomials

The cubic Bernstein polynomials:

\[
B_i(t) = \frac{(1-t)^i t^{3-i}}{i!}, \quad i = 0, 1, 2, 3
\]

Partition of unity, weights always add to 1
• Endpoint interpolation, \( B_0 \) and \( B_3 \) go to 1
**Bézier curves properties**

- Convex hull property
- Variation diminishing property
- Affine invariance

![Convex hull property](Image)

![Variation diminishing property](Image)

**Tangent**

- The derivative of a curve represents the tangent vector to the curve at some point

![Tangent](Image)

**Tangent**

- Computing the tangent of a polynomial curve is easy

\[
x(t) = a_3 t^3 + b_2 t^2 + c_1 t + d_0
\]

\[
x'(t) = \frac{dx}{dt}(t) = 3a_3 t^2 + 2b_2 t + c_1
\]

- Notice \( x'(t) \) is a vector
  - Doesn’t depend on \( d \)
  - Doesn’t depend on position of curve

**Drawing Bézier curves**

- Generally no low-level support for drawing curves
- Can only draw *line segments* or individual pixels
- Approximate the curve as a series of line segments (*tessellation*)
  - Uniform sampling
  - Adaptive sampling
  - Recursive subdivision

**Uniform sampling**

- Approximate curve with \( N \) straight segments
  - \( N \) chosen in advance
  - Evaluate \( x_i = x(t_i) \) where \( t_i = \frac{i}{N} \) for \( i = 0, 1, \ldots, N \)

\[
x(t_i) = a \cdot \frac{i^3}{N^3} + b \cdot \frac{i^2}{N^2} + c \cdot \frac{i}{N} + d
\]

- Connect the points with lines
  - Too few points?
    - Bad approximation
    - “Curve” is faceted
  - Too many points?
    - Slow to draw too many line segments
    - Segments may draw on top of each other
Adaptive Sampling

• Use only as many line segments as you need
  – Fewer segments where curve is mostly flat
  – More segments where curve bends
  – Segments never smaller than a pixel

• Various schemes for sampling, checking results, deciding whether to sample more

Recursive Subdivision

• Any cubic curve segment can be expressed as a Bézier curve
• Any piece of a cubic curve is itself a cubic curve
• Therefore:
  – Any Bézier curve can be broken up into smaller Bézier curves

de Casteljau subdivision

• Any cubic curve segment can be expressed as a Bézier curve
• Any piece of a cubic curve is itself a cubic curve
• Therefore:
  – Any Bézier curve can be broken up into smaller Bézier curves

• de Casteljau construction points are the control points of two Bézier sub-segments

Adaptive subdivision algorithm

• Use de Casteljau construction to split Bézier segment
• For each half
  – If flat enough: draw line segment
  – Else: recurse
• Curve is flat enough if hull is flat enough
• Test how far the handles are from a straight segment
  – If it’s about a pixel, the hull is flat

Outline for today

• Summary of Bézier curves
• Piecewise-cubic Bézier curves
• B-splines
• Surface patches

More control points

• Cubic Bézier curve limited to 4 control points
  – Cubic curve can only have one inflection
  – Need more control points for more complex curves
• k-th order Bézier curve with k control points

• Hard to control and hard to work with
  – Intermediate points don’t have obvious effect on shape
  – Changing any control point changes the whole curve
  – Want local support: each control point only influences nearby portion of curve
### Piecewise curves
- Sequence of simple (low-order) curves, end-to-end
  - Known as a **piecewise polynomial curve**
- Sequence of line segments
  - **Piecewise linear curve**
- Sequence of cubic curve segments
  - **Piecewise cubic** curve (here piecewise Bézier)

### Piecewise Bézier curve
- Given $3N+1$ points $p_0, p_1, \ldots, p_N$
- Define $N$ Bézier segments:
  - $x_i(t) = B_i(t)p_0 + B_i(t)p_1 + B_i(t)p_2 + B_i(t)p_3$
  - $x_i(t) = B_i(t)p_3 + B_i(t)p_4 + B_i(t)p_5 + B_i(t)p_6$
  - \( \vdots \)
  - $x_{N-1}(t) = B_1(t)p_{2N-3} + B_1(t)p_{2N-2} + B_1(t)p_{2N-1} + B_1(t)p_{3N}$

### Continuity
- Want smooth curves
- **$C^1$ continuity**
  - No gaps
  - Segments match at the endpoints
  - Tangents/normals are $C^1$ continuous (no jumps)
- **$C^2$ continuity** second derivative is well defined
  - Tangents/normals are $C^2$ continuous
  - Important for high quality reflections

### Piecewise-linear curve
- Given $N+1$ points $p_0, p_1, \ldots, p_N$
- Define curve

\[
x(u) = \begin{cases} 
p_i & \text{if } i = \lfloor u \rfloor \\
p_{i+1} & \text{if } i = \lfloor u \rfloor + 1 \\
p_i, p_{i+1} & \text{if } u \text{ is continuous at } p_i, p_{i+1} 
\end{cases} \\
\text{where } i = \lfloor u \rfloor 
\]

- $N+1$ points define $N$ linear segments
- $C^0$ continuous by construction
- $C^1$ at $p_i$ when $p_{i-1}, p_i, p_{i+1}$ are connected

### Piecewise Bézier curve
- Parameter is in \( 0 \leq u \leq 3N \)

\[
x(u) = \begin{cases} 
x_i(u), & 0 \leq u \leq 3 \\
x_i(u-1), & 3 \leq u \leq 6 \\
\vdots & \\
x_{N-1}(u-(N-1)), & 3N-3 \leq u \leq 3N \\
x_i(u), & \text{where } i = \lfloor u \rfloor 
\end{cases} 
\]
Piecewise Bézier curves

- Used often in 2D drawing programs
- Inconveniences
  - Must have 4 or 7 or 10 or 13 or ... (1 plus a multiple of 3) control points
  - Some points interpolate, others approximate
  - Need to impose constraints on control points to obtain $C^1$ continuity
  - $C^2$ continuity more difficult
- Solutions
  - User interface using “Bézier handles”
  - Generalization to B-splines

Outline for today

- Summary of Bézier curves
- Piecewise-cubic Bézier curves
- B-splines
- Surface patches

Generalization to B-splines

- Evaluate piecewise Bézier curve using sliding window
- Local support
  - Window spans four control points
  - Window contains for Bernstein polynomials
  - Window moves forward 3 units when $u$ passes to next Bézier segment
  - Evaluate matrix form in window

Piecewise Bézier curve

- 3N+1 points define N Bézier segments
- $x(3i) = p_{3i}$
- $C^0$ continuous by construction
- $C^1$ continuous at $p_{3i}$ when $p_{3i} - p_{3i-1} = p_{3i+1} - p_{3i}$
- $C^2$ is harder to get

Bézier handles

- Segment end points (interpolating) presented as curve control points
- Midpoints (approximating points) presented as “handles”
- Can have option to enforce $C^1$ continuity

Generalization to B-splines

- Still sliding window, but same weight function at each point (B-spline blending function)

- Blending functions have local support
- Cubic curve: at each $u$ only 4 weighting functions are non-zero
- Shift “window” by 1, not by 3
B-splines formulations

- Weighted average of control points using B-spline blending functions \( b_i(u) \) (no details here) \( x(u) = \sum b_i(u)p_i \)
- Positive, partition of unity => convex hull property
- Matrix form

\[
x(u) = \begin{bmatrix} t^3 & t^2 & t \end{bmatrix} \begin{bmatrix} -1 & 3 & -6 & 3 & 0 \\ 3 & -6 & 3 & 0 & 0 \\ -3 & 0 & 3 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} p_i \]

where \( t = u - i \) and \( j = \lfloor u \rfloor \)

B-splines properties

- \( n \)-th order b-spline is \( C^{n-1} \) continuous
- Widely used for curve and surface modeling
  - Intuitive behavior
  - Local support: curve only affected by nearby control points
  - Techniques for inserting new points, joining curves, etc.
- Doesn’t interpolate endpoints
  - Not a problem for closed curves
  - Can be generalized to interpolate any point along the curve

But wait, there’s more

Further generalizations

- Rational B-splines
- Non-uniform B-splines
- Non-uniform rational B-splines (NURBS)

Rational curves

- Big drawback of all cubic curves: can’t make circles!
  - Nor ellipses, nor arcs. I.e. can’t make conic sections
- Rational B-spline
  - Add a weight to each control point
  - Homogeneous point: use \( w \)

\[
x(u) = \sum b_i(u)p_i \quad x(u) = \sum b_i(u)w_i\tilde{p}_i \]

Polynomial curve (b-spline, Bézier) Rational curve

Rational curves

- Weight causes point to “pull” more (or less)
- With proper points & weights, can do circles
- Can generate curves for circles etc. with appropriate weights
- Need extra user interface to adjust the weights
- Often, hand-drawn curves are unweighted
Non-uniform rational B-splines (NURBS)

- Non-uniform: don’t assume that control points are equidistant in $u$
- Introduce knot vector
  - Describes the distribution of the control points
  - More flexibility in defining blending functions
- Uniform knot vector
- Nonuniform knot vector

NURBS

- Can make corners (C¹ discontinuity)
- Certain knot values turn out to give a Bézier segment
- Allows mixing interpolating (e.g. at endpoints) and approximating

Outline for today

- Summary of Bézier curves
- Piecewise-cubic Bézier curves
- B-splines
- Surface Patches

Curved surfaces

- Described by a 1D series of control points
- A function $x(t)$
- Segments joined together to form a longer curve

Surfaces

- Described by a 2D mesh of control points
- Parameters have two dimensions (two dimensional parameter domain)
- A function $x(u,v)$
- Patches joined together to form a bigger surface

Parametric surface patch

- $x(u,v)$ describes a point in space for any given $(u,v)$ pair
  - $u,v$ each range from 0 to 1
- 2D parameter domain
Parametric surface patch

- \( x(u,v) \) describes a point in space for any given \((u,v)\) pair
- \( u,v \) each range from 0 to 1
- Parametric curves
  - For fixed \( u_0 \), have a \( v \) curve \( x(u_0,v) \)
  - For fixed \( v_0 \), have a \( u \) curve \( x(u,v_0) \)
  - For any point on the surface, there are a pair of parametric curves that go through point

Tangents

- The tangent to a parametric curve is also tangent to the surface
- For any point on the surface, there are a pair of (parametric) tangent vectors
- Note: not necessarily perpendicular to each other

Surface Normal

- Cross product of the two tangent vectors
- Order matters!

Typically we are interested in the unit normal, so we need to normalize

Bilinear patch

- Control mesh with four points \( p_0, p_1, p_2, p_3 \)
- Compute \( x(u,v) \) using a two-step construction

Bilinear patch (step 1)

- For a given value of \( u \), evaluate the linear curves on the two \( u \)-direction edges
- Use the same value \( u \) for both:
Bilinear patch (step 2)

- Consider that \( q_0, q_1 \) define a line segment
- Evaluate it using \( v \) to get \( x \)

\[
x = \text{Lerp}(v, q_0, q_1)
\]

Bilinear patch

- Combining the steps, we get the full formula

\[
x(u, v) = \text{Lerp}(v, \text{Lerp}(u, p_1, p_2), \text{Lerp}(u, p_3, p_2))
\]

Bilinear patch

- Try the other order
- Evaluate first in the \( v \) direction

\[
r_0 = \text{Lerp}(v, p_0, p_2) \quad r_1 = \text{Lerp}(v, p_1, p_3)
\]

Bilinear patch

- Consider that \( r_0, r_1 \) define a line segment
- Evaluate it using \( u \) to get \( x \)

\[
x = \text{Lerp}(u, r_0, r_1)
\]

Bilinear patch

- The full formula for the \( v \) direction first:

\[
x(u, v) = \text{Lerp}(u, \text{Lerp}(v, p_1, p_3), \text{Lerp}(v, p_3, p_1))
\]

Bilinear patch

- It works out the same either way!

\[
x(u, v) = \text{Lerp}(v, \text{Lerp}(u, p_1, p_2), \text{Lerp}(u, p_3, p_2))
\]

\[
x(u, v) = \text{Lerp}(u, \text{Lerp}(v, p_1, p_2), \text{Lerp}(v, p_3, p_2))
\]
**Bilinear patch**

- Visualization

![Bilinear patch visualization](image)

**Bilinear patches**

- Weighted sum of control points
  \[ x(u, v) = (1 - u)(1 - v)p_0 + u(1 - v)p_1 + (1 - u)v p_2 + uv p_3 \]

- Bilinear polynomial
  \[ x(u, v) = (p_0 - p_2 - p_3 + p_2)u + (p_2 - p_3 + p_3)v + (p_2 - p_3)uv \]

- Matrix form exists, too

**Properties**

- Interpolates the control points
- The boundaries are straight line segments
- If all 4 points of the control mesh are co-planar, the patch is flat
- If the points are not coplanar, get a curved surface
  - saddle shape, AKA hyperbolic paraboloid
  - The parametric curves are all straight line segments!
  - a (doubly) ruled surface: has (two) straight lines through every point

- Not terribly useful as a modeling primitive

**Next time**

- More surfaces
- Back to rendering, shader programming