Beziers Curves, B-Splines, NURBS
Example Application: Font Design and Display

- Curved objects are everywhere
- There is always need for:
  - mathematical fidelity
  - high precision
  - artistic freedom and flexibility
  - physical realism
Example Application: Graphic Design and Arts
Example Application: Tool Path Generation and Motion Planning
Functional Representations

- **Explicit Functions:**
  - representing one variable with another
  - fine if \( \exists \) only one \( x \) value for each \( y \) value
  - Problem: what if I have a sphere?

\[
z = \sqrt{r^2 - x^2 - y^2}
\]

- Multiple values …. (not used in graphics)
Functional Representations

• **Implicit Functions:**
  - curves/surfaces represented as “the zeros”
  - good for rep. of $n-1$-D objects in $n$-D space
  - Sphere example:
  - What class of function?
    - *polynomial*: linear combo of integer powers of $x, y, z$
    - *algebraic curves & surfaces*: rep’d by implicit polynomial functions
    - *polynomial degree*: total sum of powers, i.e. polynomial of degree 6:
      
      $$x^2 + y^2 + z^2 - r^2 = 0$$
Functional Representations

- **Parametric Functions:**
  - 2D/3D curve: two functions of one parameter
    \[(x(u), y(u)) \quad (x(u), y(u), z(u))\]
  - 3D surface: three functions of two parameters
    \[(x(u,v), y(u,v), z(u,v))\]
  - Example: Sphere
    Note: rep. not algebraic, but is parametric

\[
x(\theta, \phi) = \cos \phi \cos \theta \\
y(\theta, \phi) = \cos \phi \sin \theta \\
z(\theta, \phi) = \sin \phi
\]
Functional Representations

• Which is best??
  – It depends on the application
  – Implicit is good for
    • computing ray/surface intersection
    • point inclusion (inside/outside test)
    • mass & volume properties
  – Parametric is good for
    • subdivision, faceting for rendering
    • Surface & area properties
    • popular in graphics
Issues in Specifying/Designing Curves/Surfaces

- Note: the internal mathematical representation can be very complex
  - high degree polynomials
  - hard to see how parameters relate to shape

- How do we deal with this complexity?
  - Use *curve control points* and either
    - Interpolate
    - Approximate
Points to Curves

- **The Lagrangian interpolating polynomial**
  - \( n+1 \) points, the unique polynomial of degree \( n \)
  - curve wiggles thru each control point
  - Issue: not good if you want smooth or flat curves

- **Approximation** of control points
  - points are *weights* that tug on the curve or surface

![Interpolation vs Approximation](image)
Parametric Curves

- General rep:
  \[ x = x(t), \ y = y(t), \ z = z(t) \]

- Properties:
  - individual functions are single-valued
  - approximations are done with piecewise poly curves
  - Each segment is given by three cubic polynomials \((x,y,z)\) in parameter \(t\)
  - Concise representation
Cubic Parametric Curves

• Balance between
  – Complexity
  – Control
  – Wiggles
  – Amount of computation
  – Non-planar
Parametric Curves

- Cubic Polynomials that define a parametric curve segment are of the form

\[ Q(t) = [x(t) \ y(t) \ z(t)]^T \]

- Notice we restrict the parameter \( t \) to be

\[ 0 \leq t \leq 1. \]

\[
\begin{align*}
x(t) &= a_xt^3 + b_xt^2 + c_xt + d_x, \\
y(t) &= a_yt^3 + b_yt^2 + c_yt + d_y, \\
z(t) &= a_zt^3 + b_zt^2 + c_zt + d_z, \\
\end{align*}
\]

\[ 0 \leq t \leq 1. \]
Parametric Curves

- If coefficients are represented as a matrix

\[
C = \begin{bmatrix}
  a_x & b_x & c_x & d_x \\
  a_y & b_y & c_y & d_y \\
  a_z & b_z & c_z & d_z \\
\end{bmatrix}
\]

\[
t^T = \begin{bmatrix}
  t^3 & t^2 & t & 1 \\
\end{bmatrix}^T
\]

\[
Q(t) = [x(t) \ t(t) \ z(t)]^T = C \cdot t
\]
Parametric Curves

• $Q(t)$ can be defined with four constraints
  – Rewrite the coefficient matrix $C$ as
    \[ C = G \cdot M \]
    where $M$ is a 4x4 basis matrix, and $G$ is a four-element constraint matrix (geometry matrix)

• Expanding $Q(t)$ gives:

\[
Q(t) = T \cdot G \cdot M \cdot [x(t), y(t), z(t)]^T
\]

$Q(t)$ is a weighted sum of the columns of the geometry matrix, each of which represents a point or vector in 3-space
Parametric Curves

- Multiplying out $x(t) = G_x \cdot M \cdot \dot{T}$ gives

$$x(t) = (t^3m_{11} + t^2m_{21} + tm_{31} + m_{41})g_{1x} + (t^3m_{12} + t^2m_{22} + tm_{32} + m_{42})g_{2x} + (t^3m_{13} + t^2m_{23} + tm_{33} + m_{43})g_{3x} + (t^3m_{14} + t^2m_{24} + tm_{34} + m_{44})g_{4x}$$

(i.e. just weighted sums of the elements)

- The weights are cubic polynomials in $t$ (called the blending functions, $B=MT$)

- $M$ and $G$ matrices vary by curve
  - Hermite, Bézier, spline, etc.
Warning, Warning, Warning: Pending Notation Abuse

- $t$ and $u$ are used interchangeably as a parameterization variable for functions.

Why?
- $t$ historically is “time”, certain parametric functions can describe “change over time” (e.g. motion of a camera, physics models).
- $u$ comes from the 3D world, i.e. where two variables describe a B-spline surface.
  - $u$ and $v$ are the variables for defining a surface.

Choice of $t$ or $u$ depends on the text/reference.
Continuity

Two types:

- **Geometric Continuity, \( G^i \):**
  - endpoints meet
  - tangent vectors’ directions are equal

- **Parametric Continuity, \( C^i \):**
  - endpoints meet
  - tangent vectors’ directions are equal
  - tangent vectors’ magnitudes are equal

- In general: \( C \) implies \( G \) but not vice versa
Parametric Continuity

- **Continuity** (recall from the calculus):
  - Two curves are $C^i$ continuous at a point $p$ iff the $i$-th derivatives of the curves are equal at $p$
Continuity

- The derivative of $Q(t)$ is the parametric tangent vector of the curve:

$$\frac{d}{dt} Q(t) = Q'(t) = \left[ \frac{d}{dt} x(t) \quad \frac{d}{dt} y(t) \quad \frac{d}{dt} z(t) \right]^T = \frac{d}{dt} C \cdot T = C \cdot \left[ \begin{array}{ccc} 3t^2 & 2t & 1 \\ 0 & 1 & 0 \end{array} \right]^T = \left[ \begin{array}{ccc} 3a_x t^2 + 2b_x t + c_x & 3a_y t^2 + 2b_y t + c_y & 3a_z t^2 + 2b_z t + c_z \end{array} \right]^T$$
Continuity

- What are the conditions for $C^0$ and $C^1$ continuity at the joint of curves $x^l$ and $x^r$?
  - tangent vectors at end points equal
  - end points equal

$$x^l(1) = x^r(0), \quad \frac{d}{dt}x^l(1) = \frac{d}{dt}x^r(0)$$
Continuity

- In 3D, compute this for each component of the parametric function:
  - For the x component:
    \[ x^l(1) = x^r(0) = P_{4x}, \quad \frac{d}{dt} x^l(1) = 3(P_{4x} - P_{3x}), \quad \frac{d}{dt} x^r(0) = 3(P_{5x} - P_{4x}) \]
  - Similar for the y and z components.
Convex Hulls

- The smallest convex container of a set of points
- Both practically and theoretically useful in a number of applications
Some Types of Curves

- **Hermite**
  - def’d by two end points and two tangent vectors

- **Bézier**
  - two end points plus two control points for the tangent vectors

- **Splines**
  - **Basis Splines**
  - def’d w/ 4 control points
  - Uniform, nonrational B-splines
  - Nonuniform, nonrational B-splines
  - Nonuniform, rational B-splines (**NURBS**)
Bézier Curves

- Pierre Bézier @ Rénault ~1960
- Basic idea
  - four points
  - Start point $P_0$
  - End point $P_3$
  - Tangent at $P_0$, $P_0P_1$
  - Tangent at $P_3$, $P_3P_2$
Bézier Curves

An Example:

- **Geometry matrix** is
  \[
  G_B = \begin{bmatrix}
    P_1 & P_2 & P_3 & P_4 \\
  \end{bmatrix}
  \]
  where \( P_i \) are control points for the curve

- **Basis Matrix** is

  \[
  M_B = \begin{bmatrix}
    -1 & 3 & -3 & 1 \\
    3 & -6 & 3 & 0 \\
    -3 & 3 & 0 & 0 \\
    1 & 0 & 0 & 0 \\
  \end{bmatrix}
  \]
Bézier Curves

- The general representation of a Bézier curve is

\[ Q(t) = G_B \cdot M_B \cdot T \]

where \( G_B \) - Bézier Geometry Matrix

\( M_B \) - Bézier Basis Matrix

which is (multiplying out):

\[ Q(t) = (1 - t)^3 P_1 + 3t(1 - t)^2 P_2 + 3t^2(1 - t)P_3 + t^3 P_4 \]
Bernstein Polynomials

- The general form for the $i$-th Bernstein polynomial for a degree $k$ Bézier curve is
  \[
  b_{ik}(u) = \binom{k}{i} (1 - u)^{k-i} u^i.
  \]

- Some properties of BPs:
  - Invariant under transformations
  - Form a *partition of unity*, i.e. summing to 1
  - Low degree BPs can be written as high degree BPs
  - BP derivatives are linear combo of BPs
  - Form a basis for space of polynomials w/ deg≤$k$
General Bezier Curve

$$s(t) = \sum_{i=0}^{n} p_i B_{n,i}(t)$$

Bernstein basis

$$B_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

The Quadratic and Cubic Curves of Java 2D are Bezier Curves with n=2 and n=3

The $p_i$ are the control points
Bernstein Polynomials

For those that forget combinatorics

\[ b_{ik}(u) = \frac{k!}{i!(k - i)!} (1 - u)^{k - i} u^i \]
Joining Bézier Segments: The Bernstein Polynomials

- Observe

\[ Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4 \]

The Four Bernstein polynomials
- also defined by

\[ B_B = M_B \cdot T \]

- These represent the blending proportions among the control points
Joining Bézier Segments: The Bernstein Polynomials

- The four cubic Bernstein polynomials

\[ B_B = M_B \cdot T \]

- Observe:
  - at \( t=0 \), only \( B_{B_1} \) is >0
    - curve interpolates P1
  - at \( t=1 \), only \( B_{B_4} \) is >0
    - curve interpolates P4
Joining Bézier Segments: The Bernstein Polynomials

- Cubic Bernstein blending functions

- Observe: the coefficients are just rows in Pascal’s triangle

\[
\begin{align*}
b_{03}(u) &= (1-u)^3 \\
b_{13}(u) &= 3u(1-u)^2 \\
b_{23}(u) &= 3u^2(1-u) \\
b_{33}(u) &= u^3.
\end{align*}
\]
Properties of Bézier Curves

- Affine invariance
- Invariance under affine parameter transformations
- Convex hull property
  - curve lies completely within original control polygon
- Endpoint interpolation
- Intuitive for design
  - curve mimics the control polygon
Issues with Bézier Curves

- Creating complex curves may (with lots of wiggles) requires many control points
  - potentially a very high-degree polynomial
- Bézier blending functions have *global support* over the whole curve
  - move just one point, change whole curve
- *Improved Idea:* link ($C^1$) lots of low degree (cubic) Bézier curves end-to-end
Bezier Curves, B-Splines, NURBS
Some Types of Curves

- **Hermite**
  - def’d by two end points and two tangent vectors

- **Bézier**
  - two end points plus two control points for the tangent vectors

- **Splines**
  - *Basis Splines*
  - def’d w/ 4 control points
  - Uniform, nonrational *B*-splines
  - Nonuniform, nonrational B-splines
  - Nonuniform, rational *B*-splines (*NURBS*)
Bézier Curves

- Pierre Bézier @ Rénault ~1960
- Basic idea
  - four points
  - Start point $P_0$
  - End point $P_3$
  - Tangent at $P_0$, $P_0P_1$
  - Tangent at $P_3$, $P_3P_2$
General Bezier Curve

\[ s(t) = \sum_{i=0}^{n} p_i B_{n,i}(t) \]

Bernstein basis

\[ B_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i} \]

The Quadratic and Cubic Curves of Java 2D are Bezier Curves with n=2 and n=3

The \( p_i \) are the control points
Joining Bézier Segments: The Bernstein Polynomials

- Cubic Bernstein blending functions

- Observe: the coefficients are just rows in Pascal’s triangle

\[
\begin{align*}
b_{03}(u) &= (1 - u)^3 \\
b_{13}(u) &= 3u(1 - u)^2 \\
b_{23}(u) &= 3u^2(1 - u) \\
b_{33}(u) &= u^3.
\end{align*}
\]
B-Spline Curve

\[ p(t) = \sum_{i=0}^{n} p_i N_{k,i}(t) \]

Defined only on \([t_3, t_{n+k-2})\)

Normalized B-spline blending functions

\[ N_{0,i}(t) = \begin{cases} 1, & t \in [t_i, t_{i+1}) \\ 0, & \text{otherwise} \end{cases} \]

\[ N_{k,i}(t) = \frac{t - t_i}{t_{i+k} - t_i} N_{k-1,i}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{k-1,i+1}(t) \]

n+1 control points and n+k+2 parameters known as knots
If the knots are uniformly distributed

\[ b_{-1} = (p_{i-1} + 2p_i)/3 \]
\[ b_1 = (2p_i + p_{i+1})/3 \]
\[ b_0 = (b_{-1} + b_1)/2 \]
\[ b_2 = (p_i + 2p_{i+1})/3 \]
\[ b_4 = (2p_{i+1} + p_{i+2})/3 \]
\[ b_3 = (b_2 + b_4)/2 \]
B-splines: Basic Ideas

- Similar to Bézier curves
  - Smooth blending function times control points
- But:
  - Blending functions are non-zero over only a small part of the parameter range (giving us *local support*)
  - When nonzero, they are the “concatenation” of smooth polynomials
B-spline Blending Functions

- $B_{k,0}(t)$ is a step function that is 1 in the interval.
  - spans two intervals and is a piecewise linear function that goes from 0 to 1 (and back).

- $B_{k,1}(t)$ spans three intervals and is a piecewise quadratic that grows from 0 to 1/4, then up to 3/4 in the middle of the second interval, back to 1/4, and back to 0.
  - is a cubic that spans four intervals growing from 0 to 1/6 to 2/3, then back to 1/6 and to 0.

B-spline blending functions
B-spline Blending Functions: Example for 2\textsuperscript{nd} Order Splines

- **Note**: can’t define a polynomial with these properties (both 0 and non-zero for ranges)
- **Idea**: subdivide the parameter space into *intervals* and build a *piecewise polynomial*
  - Each interval gets different polynomial function
B-spline Blending Functions: Example for 3\textsuperscript{rd} Order Splines

- Observe:
  - at t=0 and t=1 just three of the functions are non-zero
  - all are >=0 and sum to 1, hence the convex hull property holds for each curve segment of a B-spline
B-splines: Setting the Options

- Specified by
  - \( m \geq 3 \)
  - \( m+1 \) control points, \( P_0 \ldots P_m \)
  - \( m-2 \) cubic polynomial curve segments, \( Q_3 \ldots Q_m \)
  - \( m-1 \) knot points, \( t_4 \ldots t_{m+1} \)
  - segments \( Q_i \) of the B-spline curve are
    - defined over a knot interval \( [t_i, t_{i+1}] \)
    - defined by 4 of the control points, \( P_{i-3} \ldots P_i \)
  - segments \( Q_i \) of the B-spline curve are blended together into smooth transitions via (the new & improved) blending functions
Example: Creating a B-spline Curve Segment

\[ Q_i \]

\[ t_{i-1} \quad t_i \]

\[ P_i \]
B-splines: Knot Selection

- Instead of working with the parameter space $0 \leq t \leq 1$, use $t_{\min} \leq t_0 \leq t_1 \leq t_2 \ldots \leq t_{m-1} \leq t_{\max}$.
- The knot points:
  - joint points between curve segments, $Q_i$
  - Each has a knot value
  - $m-1$ knots for $m+1$ points

$P_0\rightarrow P_1\rightarrow P_2\rightarrow P_3\rightarrow Q_3\rightarrow Q_4\rightarrow P_4\rightarrow P_5\rightarrow P_6\rightarrow Q_6\rightarrow Q_7\rightarrow P_7\rightarrow P_8\rightarrow P_9$
B-spline: Knot Sequences

- Even distribution of knots
  - uniform B-splines
  - Curve does not interpolate end points
    - first blending function not equal to 1 at \( t=0 \)

- Uneven distribution of knots
  - non-uniform B-splines
  - Allows us to tie down the endpoints by repeating knot values (in Cox-deBoor, \( 0/0=1 \))
  - If a knot value is repeated, it increases the effect (weight) of the blending function at that point
  - If knot is repeated \( d \) times, blending function converges to 1 and the curve interpolates the control point
Creating a Non-Uniform B-spline: Knot Selection

- Given curve of degree $d=3$, with $m+1$ control points $p_0, \ldots, p_m$
  - first, create $m-1+2(d-1)$ knot points
  - use knot values $(0,0,0,1,2,\ldots, m-2, m-1,m-1,m-1)$
    (adding two extra 0’s and $m-1$’s)
  - Note
    - Causes Cox-deBoor to give added weight in blending to the first and last points when $t$ is near $t_{\min}$ and $t_{\max}$
Watching Effects of Knot Selection

- 8 knot points (initially)
  - Note: knots are distributed parametrically based on $t$, hence why they “move”
- 10 control points
- Curves have as many segments as they have non-zero intervals in $u$
B-splines: Local Control Property

- **Local Control**
  - polynomial coefficients depend on a few points
  - moving control point ($P_4$) affects only local curve
  - Why: Based on curve def’n, affected region extends at most 1 knot point away
Control: Bézier vs B-splines

Observe the effect on the whole curve when controls are moved.
B-splines: Setting the Options

- How to space the knot points?
  - Uniform
    - equal spacing of knots along the curve
  - Non-Uniform

- Which type of parametric function?
  - Rational
    - $x(t), y(t), z(t)$ defined as ratio of cubic polynomials
  - Non-Rational