Parametric Curves & Surfaces
What is a parametric curve?

2D parametric curve takes the form

\[
\begin{pmatrix}
x \\
y
\end{pmatrix} = \begin{pmatrix}
f(t) \\
g(t)
\end{pmatrix}
\]

Where \( f(t) \) and \( g(t) \) are functions of \( t \)

Example: Line thru points \( a \) and \( b \)

\[
\begin{pmatrix}
x \\
y
\end{pmatrix} = \begin{pmatrix}
(1-t) a_x + t b_x \\
(1-t) a_y + t b_y
\end{pmatrix}
\]

Mapping of the real line to 2D: here \( t \) in \([0,1]\) \( \rightarrow \) line segment \( a,b \)
What is a parametric curve?

3D curves defined similarly

\[
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
= \begin{pmatrix}
  f(t) \\
  g(t) \\
  h(t)
\end{pmatrix}
\]

Example: helix

\[
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
= \begin{pmatrix}
  \cos(t) \\
  \sin(t) \\
  t
\end{pmatrix}
\]
Bézier Curves

Polynomial parametric curves

\[ f(t), g(t), h(t) \text{ are polynomial functions} \]

\[
\begin{align*}
\text{Bézier curve } & b(t) \\
\text{Bézier control points } & b_i \\
\text{Bézier polygon} & \\
\end{align*}
\]

Curve mimics shape of polygon

t in \([0,1]\) maps to curve “between” polygon

\[ b(0) = b_0 \text{ and } b(1) = b_n \]
Bézier Curves

Examples

linear: \( \mathbf{b}(t) = (1-t) \mathbf{b}_0 + t \mathbf{b}_1 \)  \( n=1 \)

quadratic: \( \mathbf{b}(t) = (1-t)^2 \mathbf{b}_0 + 2(1-t)t \mathbf{b}_1 + t^2 \mathbf{b}_2 \)  \( n=2 \)

cubic: \( \mathbf{b}(t) = (1-t)^3 \mathbf{b}_0 + 3(1-t)^2 t \mathbf{b}_1 + 3(1-t)t^2 \mathbf{b}_2 + t^3 \mathbf{b}_3 \)  \( n=3 \)

Bernstein basis \( B^n_i (t) = \binom{n}{i} (1 - t)^{n-i} \cdot t^i \)
Bézier Curves

- $f(t) = (x(t), y(t), z(t))$
- Then

$$\frac{\partial}{\partial t} f(t) = (\frac{\partial}{\partial t} x(t), \frac{\partial}{\partial t} y(t), \frac{\partial}{\partial t} z(t))$$

- This is a vector function of $t$: for each $t$ we have a vector called the Tangent vector. Normalizing it:

$$Tangent(f(t)) = \left( \frac{x'(t)}{\Delta}, \frac{y'(t)}{\Delta}, \frac{z'(t)}{\Delta} \right)$$

$$\Delta = \sqrt{(x'(t))^2 + (y'(t))^2 + (z'(t))^2}$$
Reminder: Tangent to a Function

• Given a smooth function $y = f(x)$, the tangent line passing through $(x_0, y_0)$ will be the line

$$y - y_0 = f'(x_0) \cdot (x - x_0)$$

• It touches the function curve at one point only $(x_0, y_0)$
Tangent Vector

$$v = \left( \frac{dx}{\Delta}, \frac{dy}{\Delta} \right) \quad \Delta = \sqrt{dx^2 + dy^2}$$
Tangent Vector to Curve (2D)

- Touches only at a point
- The best linear approximation
- Designates the “direction” of the curve

\[
\vec{T} = \left( \frac{x'(t)}{\sqrt{(x'(t))^2 + (y'(t))^2}}, \frac{y'(t)}{\sqrt{(x'(t))^2 + (y'(t))^2}} \right)
\]
Normal to the Curve (2D)

\[ \vec{N} = \left(-\frac{y'(t)}{\sqrt{(x'(t))^2 + (y'(t))^2}}, \frac{x'(t)}{\sqrt{(x'(t))^2 + (y'(t))^2}}\right) \]
De-Casteljau Algorithm

\[ b_0^0 = P_0, \quad b_1^0 = P_1, \quad b_2^0 = P_2, \quad b_3^0 = P_3, \quad b_4^0 = P_4 \]

\[ b_0^1 = tb_1^0 + (1-t)b_0^0 \]
\[ b_1^1 = tb_2^0 + (1-t)b_1^0 \]
\[ b_2^1 = tb_3^0 + (1-t)b_2^0 \]
\[ b_3^1 = tb_4^0 + (1-t)b_3^0 \]

\[ b_0^2 = tb_1^1 + (1-t)b_0^1 \]
\[ b_1^2 = tb_2^1 + (1-t)b_1^1 \]
\[ b_2^2 = tb_3^1 + (1-t)b_2^1 \]

\[ b_0^3 = tb_1^2 + (1-t)b_0^2 \]
\[ b_1^3 = tb_2^2 + (1-t)b_1^2 \]
\[ b_2^3 = tb_3^2 + (1-t)b_2^2 \]

\[ b_0^4 = tb_1^3 + (1-t)b_0^3 = b(t) \]
Opening for Cubic Bezier

\[ b_0^0 = P_0 \quad b_1^0 = P_1 \quad b_2^0 = P_2 \quad b_3^0 = P_3 \]

\[ b(t) = b_0^3 = tb_1^2 + (1-t)b_0^2 = \]
\[ = t(tb_2^1 + (1-t)b_1^1) + (1-t)(tb_1^1 + (1-t)b_0^1) = \]
\[ = t^2b_2^1 + 2t(1-t)b_1^1 + (1-t)^2b_0^1 = \]
\[ = t^2(tb_3^0 + (1-t)b_2^0) + 2t(1-t)(tb_2^0 + (1-t)b_1^0) + (1-t)^2(tb_1^0 + (1-t)b_0^0) = \]
\[ = t^3b_3^0 + t^2(1-t)b_2^0 + 2t^2(1-t)b_2^0 + 2t(1-t)^2b_1^0 + t(1-t)^2b_1^0 + (1-t)^3b_0^0 = \]
\[ = t^3b_3^0 + 3t^2(1-t)b_2^0 + 3t(1-t)^2b_1^0 + (1-t)^3b_0^0 = \]
\[ = t^3P_3 + 3t^2(1-t)P_2 + 3t(1-t)^2P_1 + (1-t)^3P_0 \]

\[ Cubic \ Polynomial \ in \ t \]
General Bezier Curve

• Given a set of points \( \{P_0, P_1, ..., P_n\} \), a Bezier curve is defined as a weighted sum of all points:

\[
\sum_{i=0}^{n} b(t) B_i^n(t) \cdot P_i
\]

for \( t \in [0,1] \):

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

• Where the blending functions are Bernstein-Bezier basis functions:
Cubic Bezier in Matrix Form

\[
b(t) = (1 - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2 (1 - t) P_2 + t^3 P_3 = \]
\[
(-t^3 + 3t^2 - 3t + 1) P_0 + (3t^3 - 6t^2 + 3t) P_1 + (-3t^3 + 3t^2) P_2 + t^3 P_3 = \]
\[
(-P_0 + 3P_1 - 3P_2 + P_3) t^3 + (3P_0 - 6P_1 + 3P_2) t^2 + (P_0 + 3P_1) t + P_0 \]

\[
\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} \]

The curve is \( b(t) = T \cdot M \cdot G \)

\( T = (t^3, t^2, t, 1)^T \), \( G = \text{Geometry (input)} \), \( M = \text{Blending functions matrix} \)
Coefficients vs. Blending Functions

\[ T(MG) = \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} \begin{pmatrix} -P_0 + 3P_1 - 3P_2 + P_3 \\ 3P_0 - 6P_1 + 3P_2 \\ P_0 + 3P_1 \\ P_0 \end{pmatrix} \]

\[ (TM)G = \begin{pmatrix} (1 - t)^3 & 3t(1 - t)^2 & 3t^2(1 - t) & t^3 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} \]
Examples

\[ b(t) = (1 - t)^2 P_0 + 2t(1 - t)P_1 + t^2 P_2 = (1 - 2t + t^2)P_0 + (2t - 2t^2)P_1 + t^2 P_2 \]
\[ b'(t) = (2t - 2)P_0 + (-4t + 2)P_1 + 2tP_2 \]

\[
\begin{align*}
    b'(0) &= 2(P_1 - P_0) \\
    b'(1) &= 2(P_2 - P_1)
\end{align*}
\]

\[ b(t) = (1 - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2 (1 - t)P_2 + t^3 P_3 = \\
(1 - 3t + 3t^2 - t^3)P_0 + (3t - 6t^2 + 3t^3)P_1 + (3t^2 - 3t^3)P_2 + t^3 P_3 \]
\[ b'(t) = -3(1 - t)^2 P_0 + 3(-2t(1 - t) + (1 - t)^2)P_1 + 3(-t^2 + 2t(1 - t))P_2 + 3t^2 P_3 = \\
= -3(1 - t)^2 P_0 + (3 - 12t + 9t^2)P_1 + (6t - 9t^2)P_2 + 3t^2 P_3 \]

\[
\begin{align*}
    b'(0) &= 2(P_1 - P_0) \\
    b'(1) &= 2(P_3 - P_2)
\end{align*}
\]

In general \( b'(0) = n(P_1 - P_0) \) \( b'(1) = n(P_n - P_{n-1}) \)
Bézier Properties: Overview

• We will now look at some properties of Bézier curves.
  – Generally “Good” Properties 😊
    • Endpoint Interpolation
    • Smooth Joining
    • Affine Invariance
    • Convex-Hull Property
  – Generally “Bad” Properties 😞
    • Not Interpolating
    • No Local Control
Bézier Properties:

😊 Endpoint Interpolation

- Bézier curve generally do not pass through all control points.
- However, they do pass through the first & last control points.

- So, to make two Bézier curves join, make the first control point of one equal to the last control point of the other.
  - This may not result in a smooth curve overall.
Bézier Properties: 

😊 Easy Smooth Joining

- At its start, the velocity vector (first derivative) of a Bézier curve points from the first control point, towards the second.
- At its end, the velocity vector points from the last control point away from the next-to-last.
- To make two Bézier curves join smoothly put the last two control points of one and the first two of the other in a line, as shown:
Bézier Properties:

😊 Affine Invariance

• An *affine transformation* is a transformation that can be produced by doing a linear transformation followed by a translation.

• All of the transformations we have dealt with, except for perspective projection, are affine. These include:
  – Translation.
  – Rotation.
  – Scaling.
  – Shearing.
  – Reflection.

• Bézier curves have the property that applying an affine transformation to each control points results in the same transformation being applied to every point on the curve.
  – For example, to rotate a Bézier curve, apply a rotation to the control points.
  – In short: Transformations act the way you want them to.
Bézier Properties:

😊 Convex-Hull Property

- The *convex hull* of a set of points is the smallest convex region containing them.
  - Informally speaking, “lasso” (or shrink-wrap) the points; the region inside the lasso is the convex hull.

- A Bézier curve lies entirely in the convex hull of its control points.
  - This property makes it easy to specify where a curve will not go.
  - Smooth interpolating splines never have this property.
Bézier Properties:

😊 Not-So-Good Stuff

• Again Bézier curves do not interpolate all their control points.
  – So we can easily specify where it does not go, but not where it does go.

• Bézier curves also do not have “local control”.
  – A curve has local control if moving a single control point only changes a small part of the curve (the part near the control point).
  – Moving any control point on a Bézier curve changes the whole curve.
  – This is the main reason we do not use Bézier curves with a large number of control points.
    • Instead, we piece together several 3- or 4-point Bézier curves in a smooth way.
    • This multiple-Bézier curve does have local control.
Piecewise Cubic Bezier = Splines

- Piece together Bézier curves
- Solve for "interior" control vertices
  - Positional (C0) continuity
  - Derivative (C1) continuity
- Properties:
  - Local control
  - Interpolating (every third)
Constraints

End nodes:
interpolating + tangent direction

Internal nodes:
interpolating + continuity
Splitting a Cubic Bezier

\( p_0, p_1, p_2, p_3 \) determine a cubic Bezier polynomial and its convex hull

Consider left half \( l(u) \) and right half \( r(u) \)
l(u) and r(u)

Since l(u) and r(u) are Bezier curves, we should be able to find two sets of control points \{l_0, l_1, l_2, l_3\} and \{r_0, r_1, r_2, r_3\} that determine them.
Convex Hulls

\{l_0, l_1, l_2, l_3\} and \{r_0, r_1, r_2, r_3\} each have a convex hull that is closer to \(p(u)\) than the convex hull of \{p_0, p_1, p_2, p_3\}. This is known as the *variation diminishing property*.

The polyline from \(l_0\) to \((l_3 = r_0)\) to \(r_3\) is an approximation to \(p(u)\). Repeating recursively we get better approximations.
Equations

Start with Bezier equations $p(u) = u^T M_B p$

$l(u)$ must interpolate $p(0)$ and $p(1/2)$

\[
\begin{align*}
l(0) &= l_0 = p_0 \\
l(1) &= l_3 = p(1/2) = 1/8( p_0 + 3 p_1 + 3 p_2 + p_3 )
\end{align*}
\]

Matching slopes, taking into account that $l(u)$ and $r(u)$ only go over half the distance as $p(u)$

\[
\begin{align*}
l'(0) &= 3(l_1 - l_0) = p'(0) = 3/2(p_1 - p_0) \\
l'(1) &= 3(l_3 - l_2) = p'(1/2) = 3/8(- p_0 - p_1 + p_2 + p_3)
\end{align*}
\]

Symmetric equations hold for $r(u)$
Efficient Form

\[ l_0 = p_0 \]
\[ r_3 = p_3 \]
\[ l_1 = \frac{1}{2}(p_0 + p_1) \]
\[ r_1 = \frac{1}{2}(p_2 + p_3) \]
\[ l_2 = \frac{1}{2}(l_1 + \frac{1}{2}(p_1 + p_2)) \]
\[ r_1 = \frac{1}{2}(r_2 + \frac{1}{2}(p_1 + p_2)) \]
\[ l_3 = r_0 = \frac{1}{2}(l_2 + r_1) \]

Requires only shifts and adds!
Bézier Surface (Patch)

Polynomial parametric surface

\[ f(u,v), g(u,v), h(u,v) \] are polynomial functions written in the Bernstein basis

Bézier surface \( b(u,v) \)

Bézier control points \( b_{ij} \)

Bézier control net
Bézier Surface

Properties

- boundary curves lie on surface
- boundary curves defined by boundary polygons
Normals

• For rendering we need the normals if we want to shade
  – Can compute from parametric equations
    \[ \mathbf{n} = \frac{\partial \mathbf{p}(u, v)}{\partial u} \times \frac{\partial \mathbf{p}(u, v)}{\partial v} \]
  – Can use vertices of corner points to determine
  – OpenGL can compute automatically
Utah Teapot

- Most famous data set in computer graphics
- Widely available as a list of 306 3D vertices and the indices that define 32 Bezier patches
Example

• Consider the previous evaluator that was set up for a cubic Bezier over (0,1)

• Suppose that we want to approximate the curve with a 100 point polyline

    glBegin(GL_LINE_STRIP)
    for(i=0; i<100; i++)
        glEvalCoord1f((float) i/100.0);
    glEnd();
Equally Spaced Points

Rather than use a loop, we can set up an equally spaced mesh (grid) and then evaluate it with one function call

\[
glMapGrid(100, 0.0, 1.0);
\]

sets up 100 equally-spaced points on (0,1)

\[
glEvalMesh1(GL_LINE, 0, 99);
\]

renders lines between adjacent evaluated points from point 0 to point 99
Beziers Surfaces

• Similar to 1D but use 2D evaluators in \( u \) and \( v \)
• Set up with ...

\[
glMap2f(type,\n  u\_min, u\_max, u\_stride, u\_order,\n  v\_min, v\_max, v\_stride, v\_order,\n  pointer\_to\_data);\n\]

• Evaluate with ...

\[
-glEvalCoord2f(u,v)\]

Angel: Interactive Computer Graphics 5E © Addison-Wesley 2009
Example

bicubic patch over $(0,1) \times (0,1)$

```c
point data[4][4] = { .......... };
glMap2f(GL_MAP_VERTEX_3,
        0.0, 1.0, 3, 4,
        0.0, 1.0, 12, 4,
        data);
```

Note that in v direction data points are separated by 12 floats since `data` array is stored by rows.
Rendering with Lines

must draw line strips through sample point \((u=j/100, v=j/100)\) in both directions \((u \& v)\)

```cpp
for(j=0; j<100; j++) {
    glBegin(GL_LINE_STRIP);
    for(i=0; i<100; i++) // for all 0<=u<1 and v=j/100
        glEvalCoord2f((float) i/100.0, (float) j/100.0);
    glEnd();
    glBegin(GL_LINE_STRIP);
    for(i=0; i<100; i++) // for u=j/100 and all 0<=v<1
        glEvalCoord2f((float) j/100.0, (float) i/100.0);
    glEnd();
}
```
Rendering with Quadrilaterals

We can form a quad mesh and render with lines

```c
for(j=0; j<99; j++) // for each strip (v=j/100)
{
    glBegin(GL_QUAD_STRIP);
    for(i=0; i<100; i++) // for all 0<=u<1
    {
        // find point u=i/100 and v = j/100
        glEvalCoord2f((float)i/100.0,(float)j/100.0);
        // find point u=(i+1)/100 and v = j/100
        glEvalCoord2f((float)(i+1)/100.0,(float)j/100.0);
    }
    glEnd();
}
```
Uniform Meshes

• We can form a 2D mesh (grid) in a similar manner to 1D for uniform spacing

    glMapGrid2 (u_num, u_min, u_max, 
              v_num, v_min, v_max);

• Can evaluate as before with lines or if we want filled polygons ...

    glEvalMesh2 (GL_FILL, 
                 u_start, u_num, 
                 v_start, v_num);
Rendering with Lighting

• If we use filled polygons, we have to shade or we will see solid color uniform rendering

• Can specify lights and materials but we need normals
  – Let OpenGL find them

```c
GLuint Enable(GL_Auto_Normal);
```
NURBS

• OpenGL supports NURBS surfaces through the GLU library

• Why GLU?
  – Can use evaluators in 4D with standard OpenGL library
  – However, there are many complexities with NURBS that need a lot of code
  – There are five NURBS surface functions plus functions for trimming curves that can remove pieces of a NURBS surface
Quadrics

• Quadrics are in both the GLU and GLUT libraries
  – Both use polygonal approximations where the application specifies the resolution
  – Sphere: lines of longitude and latitude

• GLU: disks, cylinders, spheres
  – Can apply transformations to scale, orient, and position

• GLUT: Platonic solids, torus, Utah teapot, cone
GLUT Objects

Each has a wire and a solid form

- glutWireCone()
- glutWireTorus()
- glutWireTeapot()
GLUT Platonic Solids

\begin{itemize}
\item \texttt{glutWireTetrahedron()}
\item \texttt{glutWireOctahedron()}
\item \texttt{glutWireDodecahedron()}
\item \texttt{glutWireIcosahedron()}
\end{itemize}