Parametric Curves & Surfaces

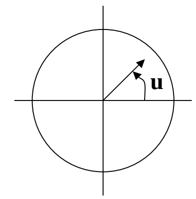
Parametric v. Implicit
Parametric Curves (Splines)
Parametric Surfaces

Carnegie Mellon

9 Feb. 2000

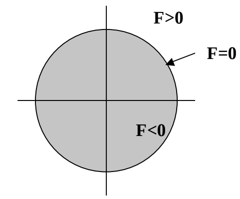
Two Ways to Define a Circle

Parametric



x(u) = r cos (u)
y(u) = r sin (u)

Implicit



$F(x,y) = x^2 + y^2 - r^2$

Curve Representations

- **Explicit: y = y(x)**

  $y = mx + b \qquad y = x^2$

  – must be a function (single-valued): big limitation

- **Parametric: (x,y) = (x(u),y(u))**

  $(x, y) = (\cos u, \sin u)$

  + easy to specify, modify, control
  – extra "hidden" variable u, the *parameter*

- **Implicit: f(x,y) = 0**

  $x^2 + y^2 - r^2 = 0$

  + y can be multiple valued function of x
  – hard to specify, modify, control

Surface Representations

- **Parametric surface — x(u,v), y(u,v), z(u,v)**
  – e.g. plane, sphere, cylinder, torus, bicubic surface, swept surface
  – parametric functions let you *iterate* over the surface by incrementing u and v in nested loops
  – great for making polygon meshes, etc
  – terrible for intersections: ray/surface, point-inside-boundary, etc
- **Implicit surface: F(x,y,z) = 0**
  – e.g. plane, sphere, cylinder, quadric, torus, blobby models
  – terrible for iterating over the surface
  – great for intersections, morphing
- **Subdivision surfaces**
  – defined by a control mesh and a recursive subdivision procedure
  – good for interactive design

Parametric Modeling - Outline

- **Why piecewise polynomials, why cubics?**
- **Curves**
  - **Hermite Splines**
  - **Bezier Splines**
  - **Catmull-Rom Splines**
  - **Natural Cubic Splines**
  - **B-Splines**
  - **NURBS**
- **Surfaces**

Building Curves

- **Okay now we know how to represent curves**
- **But how to we *specify* them?**
- **Building blocks for a useful smooth curve:**
  - **flexible, springy line**
  - **definable points for it to pass through (control points)**
- **Draftsmen used such building blocks for plane curves**
  - **weights called *ducks* to hold down a thin flexible strip called a *spline***
  - **ducks are placed at points the curve is supposed to pass through, and the springiness makes the spline fill in smoothly**
  - **The final curve is traced onto paper**
- **How to define this mathematically?**

Polynomial Interpolation

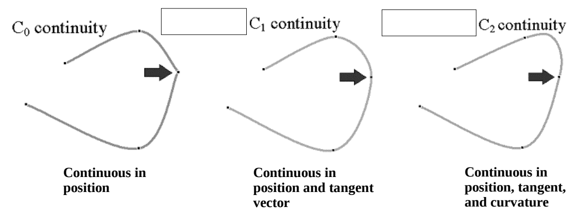
- **An *n*-th degree polynomial fits a curve to n+1 points**
  - **Example: fit a second degree curve to three points**
    - » $x(u)= au^2 + bu + c$
    - » **control points to interpolate**: $(u_1, x_1), (u_2, x_2), (u_3, x_3)$
    - » **solve for coefficients** (a, b, c): **3 linear eqns, 3 unknowns**
  - **called Lagrange Interpolation**
  - **result is a curve that is too wiggly, change to any control point affects entire curve (nonlocal) – *this method is poor***
- **We usually want the curve to be as smooth as possible**
  - **minimize the wiggles**
  - **high-degree polynomials are bad**
- **Good approach: simulate springy wire**

Splines: Piecewise Polynomials

- **A springy wire minimizes its bending energy (subject to constraints)**
- **Bending energy approximated by the integral of squared curvature**
  - **minimize this and the curve looks real**
  - **2nd derivative approximates curvature**
- **Intuitively: try to make curvature zero everywhere**
  - **If you can't, distribute bends as uniformly as possible**
- **A spline is a *piecewise polynomial* - many low degree polynomials are used to interpolate (pass through) the control points**
- ***Cubic* polynomials are the most common:**
  - **minimize the 2nd derivative**
  - **lowest order polynomials that interpolate two points and allow the gradient at each point to be defined - C^1 continuity is possible**
  - **Higher or lower degrees are possible, of course**

Piecewise Polynomials

- **Spline: lots of little polynomials pieced together**
- **Want to make sure they fit together nicely**



Cubic Space Curves

- **Cubic polynomial**
  - $x(u) = au^3+bu^2+cu+d$ = **ua**
  - **where u** = $[u^3\ \ u^2\ \ u\ \ 1]$, **a** = $[a\ \ b\ \ c\ \ d]^T$
- **Three cubics, one for each coordinate**
  - $x(u) = a_xu^3+b_xu^2+c_xu+d_x$
  - $y(u) = a_yu^3+b_yu^2+c_yu+d_y$
  - $z(u) = a_zu^3+b_zu^2+c_zu+d_z$
- **In matrix notation**

$$[x(u)\quad y(u)\quad z(u)] = [u^3\quad u^2\quad u\quad 1]\begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}$$

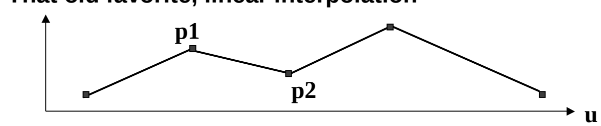
- **Or simply x = uA**

Visualizing Parametric Curves

- **Functional views: x(u), y(u), z(u)**
- **Parametric view: curve in xyz space, ignoring u**
- ***Parametrization* of a curve: how a given curve in xyz space is decomposed into functions x(u), y(u), z(u).**
- **There are an infinite number of parametrizations of a given curve! Slow, fast, speed continuous or discontinuous, clockwise (CW) or CCW…**
- **An special one: arc-length-parametrization: u is arc length. (but these very hard to compute exactly)**

A Linear Piecewise Polynomial

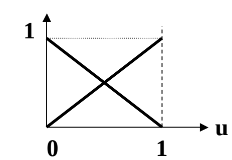
That old favorite, linear interpolation



Each segment is of the form: ***(this is a vector equation)***

$$p(u) = up_1 + (1-u)\,p_2$$

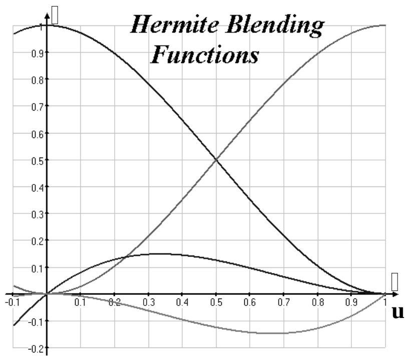
Two basis functions



For Cubics, Use Four Basis Functions

$$p(u) = \begin{bmatrix} 2u^3 - 3u^2 + 1 \\ -2u^3 + 3u^2 \\ u^3 - 2u^2 + u \\ u^3 - u^2 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

4 Basis Functions



Every cubic Hermite spline is a linear combination (blend) of these 4 functions

Why these in particular?

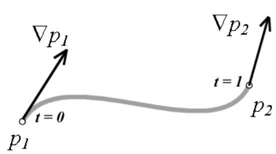
- **These are the only cubic basis functions satisfying:**

$$p(0) = p_1$$
$$p(1) = p_2$$
$$\left.\frac{dp}{du}\right|_{u=0} = \nabla p_1$$
$$\left.\frac{dp}{du}\right|_{u=1} = \nabla p_2$$

Wow! We've found a way to specify the end points and the slope at the end points!

An Illustrative Example

- **Cubic Hermite Splines**



Hermite Specification

Piecing together Hermite Curves

- **It's easy to make a multi-segment Hermite spline**
  - **each piece is specified by a cubic Hermite curve**
  - **just specify the position and tangent at each "joint"**
  - **the pieces fit together with matched positions and first derivatives**
  - **gives C1 continuity**
- **Given a list of points & tangents, you can build a piecewise cubic that passes through all the points by calculating the Hermite cubic for each segment**
- **The points that the curve has to pass through are called *knots* or *knot points***

Deriving Hermite Splines, 1

- **Four constraints:** value and slope (or in 3-D, position and tangent vector) at beginning and end of interval [0,1]

  $x(0) = x_1$

  $x(1) = x_2$

  $x'(0) = x_1'$

  $x'(1) = x_2'$

  primes on left side denote derivative; primes on right denote slope constants

- **Assume cubic form:** $x(u) = au^3+bu^2+cu+d$
- **Four unknowns:** a, b, c, d

Deriving Hermite Splines, 2

- **Since** $x(u) = au^3+bu^2+cu+d$,
  its derivative is $x'(u) = 3au^2+2bu+c$
- **Rewriting the constraints yields four linear equations:**

  $x(0) = x_1 = d$

  $x(1) = x_2 = a+b+c+d$

  $x'(0) = x_1' = c$

  $x'(1) = x_2' = 3a+2b+c$

- **Solve for** a, b, c, d:

  $a=2x_1-2x_2+x_1'+x_2'$, $b=-3x_1+3x_2-2x_1'-x_2'$, $c= x_1'$, $d= x_1$

- **Simpler if we use matrix notation…**

Hermite Splines in Matrix Notation

$$\underbrace{\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \frac{dx_1}{du} & \frac{dy_1}{du} & \frac{dz_1}{du} \\ \frac{dx_2}{du} & \frac{dy_2}{du} & \frac{dz_2}{du} \end{bmatrix}}_{\text{control vector}} = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}}_{\text{inv basis}} \underbrace{\begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}}_{\text{spline coefficients}}$$

Solve for ⬇ spline coefficients

$$\underbrace{\begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}}_{\text{spline coefficients}} = \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\text{basis}} \underbrace{\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \frac{dx_1}{du} & \frac{dy_1}{du} & \frac{dz_1}{du} \\ \frac{dx_2}{du} & \frac{dy_2}{du} & \frac{dz_2}{du} \end{bmatrix}}_{\text{control vector}}$$

The Cubic Hermite Spline Equation

$$\underbrace{\begin{bmatrix} x & y & z \end{bmatrix}}_{\text{point}} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\text{basis}} \underbrace{\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \frac{dx_1}{du} & \frac{dy_1}{du} & \frac{dz_1}{du} \\ \frac{dx_2}{du} & \frac{dy_2}{du} & \frac{dz_2}{du} \end{bmatrix}}_{\text{control matrix}}$$

Bezier Curves

- Another variant of the same game
- Instead of endpoints and tangents, four control points
  - points P1 and P4 are on the curve
  - points P2 and P3 are off the curve
  - X(0) = P1, X(1) = P4,
  - X'(0) = 3(P2-P1), X'(1) = 3(P4 - P3)
- Variant of the Hermite spline
  - basis matrix derived from the Hermite basis (or from scratch)
- Convex Hull property
  - curve contained within convex hull of control points
- Gives more uniform control knobs (series of points) than Hermite
- Scale factor (3) is chosen to make "velocity" approximately constant

From Hermite to Bezier

- Instead of endpoints and tangents, four control points
  - X(0) = P1, X(1) = P4,
  - X'(0) = 3(P2-P1), X'(1) = 3(P4 - P3)

$$\underbrace{\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \frac{dx_1}{du} & \frac{dy_1}{du} & \frac{dz_1}{du} \\ \frac{dx_2}{du} & \frac{dy_2}{du} & \frac{dz_2}{du} \end{bmatrix}}_{\text{Hermite control vector}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix}}_{\text{Bezier to Hermite}} \underbrace{\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}}_{\text{Bezier control vector}}$$

The Bezier Spline Matrix

$$[x \ y \ z] = [u^3 \ u^2 \ u \ 1] \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\text{Hermite basis}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix}}_{\text{Bezier to Hermite}} \underbrace{\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}}_{\text{Bezier control vector}}$$

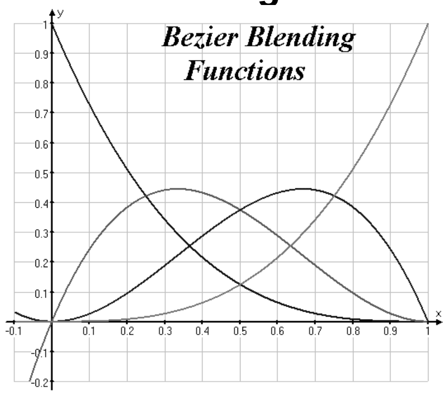
$$= [u^3 \ u^2 \ u \ 1] \underbrace{\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\text{Bezier basis}} \underbrace{\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}}_{\text{Bezier control vector}}$$

Bezier Basis Functions

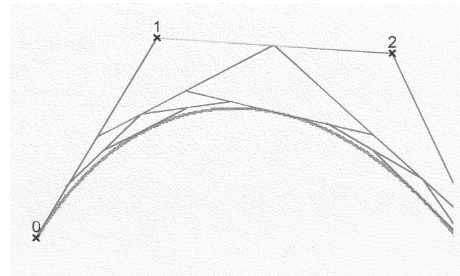
$$p(t) = \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix}^{\mathbf{T}} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

- Also known as the order 4, degree 3 Bernstein polynomials
- Nonnegative, sum to 1
- Hence the entire curve lies inside the polyhedron bounded by the control points!

Bezier Blending Functions



It's easy to subdivide Bezier curves!



- Each half is a Bezier curve!
- Note that this makes it easy to draw them by subdivision

Catmull-Rom Splines

- With Hermite splines, the designer must arrange for consecutive tangents to be collinear, to get C^1 continuity. Similar for Bezier. This gets tedious.
- Catmull-Rom: an interpolating cubic spline with *built-in C^1 continuity.*
- Compared to Hermite/Bezier: fewer control points required, but less freedom.
- Given n control points in 3-D: $p_1, p_2, \ldots, p_n$,
  - Tangent at p_i given by $s(p_{i+1} - p_{i-1})$ for i=2..n-1, for some s
  - What about endpoint tangents? (several good answers: extrapolate, or use extra control points p_0, p_{n+1})
  - Now we have positions and tangents at each knot – a Hermite specification.
  - Curve between p_i and p_{i+1} is determined by $p_{i-1}, p_i, p_{i+1}, p_{i+2}$

Catmull-Rom Spline Matrix

$$\underbrace{\begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}}_{\text{spline coefficients}} = \underbrace{\begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}}_{\text{CR basis}} \underbrace{\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}}_{\text{control vector}}$$

- Derived similarly to Hermite and Bezier
- s is tension parameter; typically s=1/2

Splines with More Continuity?

- So far, only C^1 continuity.
- How could we get C^2 continuity at knots?

- Possible answers:
  - Use higher degree polynomials
    degree 4 = quartic, degree 5 = quintic, … but these get computationally expensive, and sometimes wiggly
  - Give up local control → natural cubic splines
    A change to any control point affects the entire curve
  - Give up interpolation → cubic B-splines
    Curve goes near, but not through, the control points

Natural Cubic Splines

- If you want 2nd derivatives at joints to match up, the resulting curves are called *natural cubic splines*
- It's a simple computation to solve for the cubics' coefficients. (See *Numerical Recipes in C* book for code.)
- A real-time implementation with mouse-based moving, insertion, and deletion of control points feels a lot like a physical spline
- Finding all the right weights is a *global* calculation (solve tridiagonal linear system)

Degrees of Freedom with Natural Cubic Splines

- A natural cubic curve of *n* segments has *4n* DOFs
  - joining *n-1* pairs of points removes *n-1* DOFs (leaving *3n+1*)
  - matching gradients to give C1 continuity removes *n-1* DOFs (leaving *2n+2)*
  - fixing segment end points removes *n+1* DOFs (leaving *n+1*)
  - either:
    - » making curve C2 continuous removes *n-1* DOFs (leaving *2*)
    - » defining all gradient removes *n+1* DOFs (leaving 0)
- Natural cubic splines use up these DOFs either by
  - free boundaries: setting end 2nd derivatives to zero
  - clamped boundaries: setting the end tangents to any value
- But, other types of cubic splines can spend their DOFs differently

Interpolation, Continuity, and Local Control

- Hermite/Catmull-Rom/Bezier splines (and other varieties)
  - Interpolate control points
  - Local control
  - No C2 continuity for cubics
- Natural splines
  - Interpolate control points
  - No local control - moving one control point affects whole curve
  - C2 continuity for cubics
- Summary
  - Can't get C2, interpolation and local control with cubics - very sad.

B-Splines

- **Give up interpolation**
  - **the curve passes *near* the control points**
  - **best generated with interactive placement (because it's hard to guess where the curve will go)**
- **Curve obeys the convex hull property**
- **C2 continuity and local control are good compensation for loss of interpolation**

B-Spline Basis

- **We always need 3 more control points than spline pieces**

$$M_{Bs} = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

$$G_{Bsi} = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}$$

Comparison of Basic Cubic Splines

Type	Control points	Continuity	Interpolation
Hermite	2 points, 2 slopes	C1	Y
Bezier	4 points	C1	Y
Catmull-Rom	4 points, shared	C1	Y
Natural	n points, shared	C2	Y
B-Splines	4 points, shared	C2	N

Non-Uniform Splines

- **The *knot sequence* is the parameter spacing of knots.**
- **So far we've been assuming a uniform knot sequence: u in [0,1] for each piece of the spline.**
- **A non-uniform knot sequence relaxes this assumption.**
- **Different knot spacings yield different basis matrices (so when doing non-uniform splines you typically don't use matrices).**
- **Tangents are scaled, this gives *tension* control**
  - **curve can be regarded as a time trajectory - tangents are velocities**
- **Each segment has a different basis functions!**
- **Advantages**
  - **Can reduce continuity by duplicating knots (& control points)**
  - **Can add detail to a curve without changing knot spacing globally**

Two Types of Continuity

- **Parametric Continuity: C^k**
  - x(u), y(u), z(u) each individually continuous through kth derivative
  - (value, slope, 2^{nd} derivative, …)
  - Important for motion curves (parameter is time).
- **Geometric Continuity: G^k**
  - curve in xyz (independent of u) continuous through kth derivative
  - (position, tangent, curvature, …)
  - Important if the parameter is irrelevant.
- **A curve can be C^k without being G^k and vice versa!**
  - can you think of examples?

Reducing B-Spline Continuity

- **One can duplicate control points**
  - no duplicates- C^2, G^2
  - 1 duplicate- C^2, G^1
  - 2 duplicates- C^2, G^0; linear segments
  - 3 duplicates- C^2, G^0; more linearity
- **Or, can duplicate knots (generally better)**
  - no duplicates- C^2, G^2
  - 1 duplicate- C^1, G^1
  - 2 duplicates- C^0, G^0
  - 3 duplicates- curve is discontinuous

Rational Cubic Curves

- **Problems with cubics**
  - Cannot do exact conics (circles, ellipses)
    - » proof: how to write x(u) = r cos(u) as a cubic? You can't!
  - Not *invariant* under perspective transformations
    - » projection of a cubic is *not* a cubic
    - » Q: why is this a problem?
    - » A: can't just project control points - must project entire curve
- **Solution: homogeneous coordinates**
  - Define 4 cubics: $X(u)$, $Y(u)$, $Z(u)$ and $W(u)$
  - Curve is now: $x(u) = X(u)/W(u)$, etc…
- **The curve is now a rational function**
- **Non-Uniform Rational B–Splines (NURBS) widely used in computer-aided geometric design (CAGD)**

How to Draw Spline Curves

- **Basis matrix eqn. allows same code to draw any spline type**
- **<u>Method 1</u>: brute force**
  - Calculate the coefficients
  - For each cubic segment, vary *u* from *0* to *1* (fixed step size)
  - Plug in *u* value, matrix multiply to compute position on curve
  - Draw line segment from last position to current position
- **What's wrong with this approach?**
  - Draws in even steps of u
  - Even steps of u $\neq$ even steps of x
  - Line length will vary over the curve
  - Want to bound line length
    - » too long: curve looks jagged
    - » too short: curve is slow to draw

Drawing Splines, 2

- **Method 2: recursive subdivision - vary step size to draw short lines**

```
Subdivide(u0,u1,maxlinelength)
   umid = (u0 + u1)/2
   x0 = F(u0)
   x1 = F(u1)
   if |x1 - x0| > maxlinelength
      Subdivide(u0,umid,line)
      Subdivide(umid,u1,line)
   else drawline(x0,x1)
```

- **Variant on Method 2 - subdivide based on curvature**
  - **replace condition in "if" statement with straightness criterion**
  - **draws fewer lines in flatter regions of the curve**
- **Method 3: forward differences - rearrange terms to eliminate multiplies**

Bicubic Surfaces

- **To represent surfaces use *bicubic* functions**
  - **x(u,v), y(u,v), z(u,v) are cubic polyomials both in *u* and *v***
  - **16 terms for combination of powers of *u* and *v***

$$x(u,v) = a_{11}u^3v^3 + a_{12}u^3v^2 + \star + a_{44}$$

- **A bicubic surface can be represented by a 4x4 matrix A**

$$\mathbf{u} = [u^3, u^2, u, 1], \quad \mathbf{v} = [v^3, v^2, v, 1]$$

$$x(u,v) = \mathbf{u}^{\mathrm{T}}\mathbf{A}\,\mathbf{v}$$

- **Also known as *tensor product surfaces***

From Curves to Surfaces

- **Bicubic surfaces work like spline curves**
  - **Use control points and basis matrices**
  - **Define 16 control parameters (points and/or derivatives)**
  - **Solve for coefficients (using a rank-4 *tensor* = 4x4x4x4 array!!!)**
- **Symmetry between u and v allows this tensor to be two copies of a regular matrix**
  - **B is the basis matrix, P the matrix of control values:**

$$x(u,v) = \mathbf{u}^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{P}\mathbf{B}\mathbf{v}$$

- **Basis surface view**
  - **4 basis cubics for u and 4 for v (may or may not be same basis)**
  - **16 basis surfaces from multiplying each pair of cubics together**
  - **Bicubic surface is a linear combo (weighted sum) of basis surfaces**
- **Surface obtained by piecing together bicubic patches**

In Summary...

- **Summary:**
  - **piecewise cubic is generally sufficient**
  - **define conditions on the curves and their continuity**
  - **solve the linear system; if knot spacing is uniform, you obtain a constant basis matrix, so same basis functions for every piece**
  - **can use a nonuniform knot spacing, changing the *u*-values, to vary tension**
- **Things to know:**
  - **basic curve properties (what are the conditions, controls, and properties for each spline type)**
  - **generic matrix formula for uniform cubic splines x(u) = uBG**
  - **given definition derive a basis matrix (do not memorize matrices themselves)**
  - **how curves are generalized to surfaces**