# CS-184: Computer Graphics

## Lecture #12: Curves and Surfaces

Prof. James O'Brien
University of California, Berkeley

V2013-S-12-1.0

1

## Today

- General curve and surface representations
- Splines and other polynomial bases

2

2

# Geometry Representations

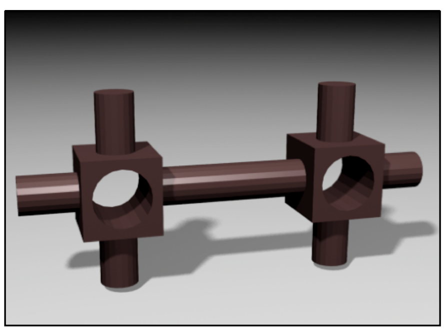
- Constructive Solid Geometry (CSG)
- Parametric
  - Polygons
  - Subdivision surfaces
- Implicit Surfaces
- Point-based Surface

- Not always clear distinctions
  - *i.e.* CSG done with implicits

3

3

# Geometry Representations



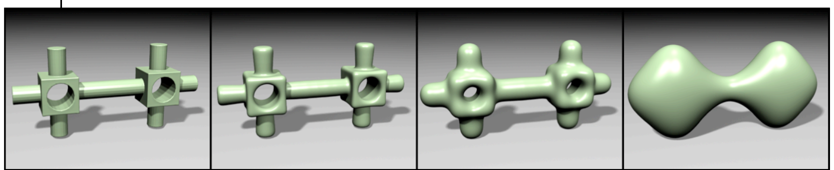
Object made by CSG Converted to polygons

4

4

# Geometry Representations

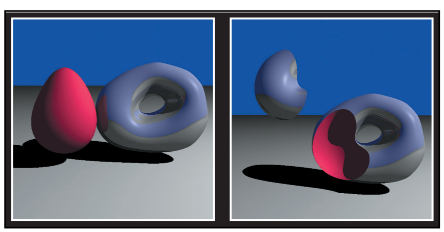
Object made by CSG
Converted to polygons
Converted to implicit surface



5

5

# Geometry Representations

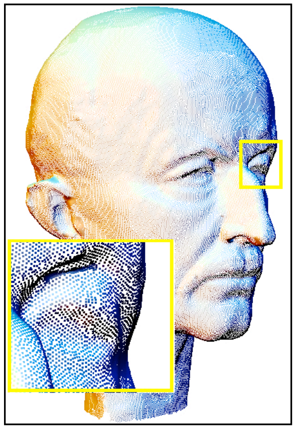


CSG on implicit surfaces

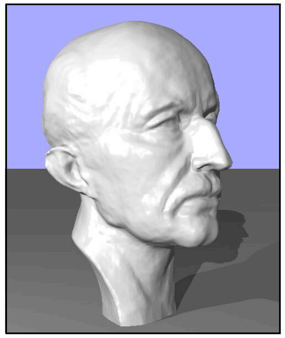
6

6

# Geometry Representations



Point-based surface descriptions



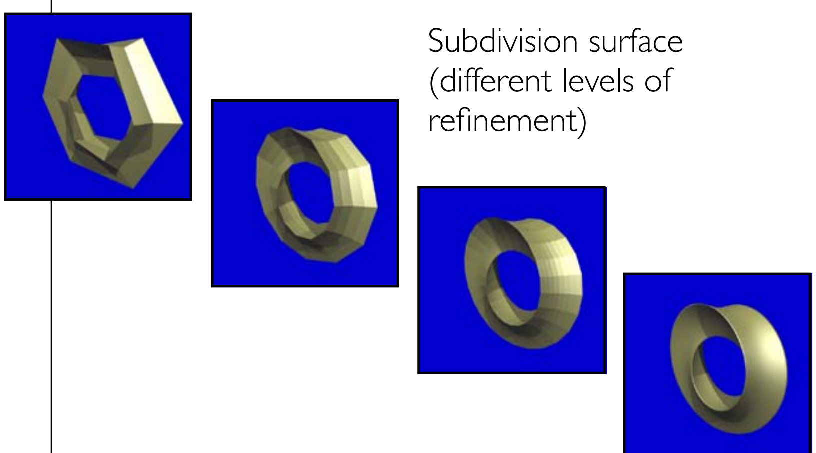
Ohtake, *et al.*, SIGGRAPH 2003

7

7

# Geometry Representations

Subdivision surface (different levels of refinement)



Images from Subdivision.org

8

8

# Geometry Representations

- Various strengths and weaknesses
  - Ease of use for design
  - Ease/speed for rendering
  - Simplicity
  - Smoothness
  - Collision detection
  - Flexibility (in more than one sense)
  - Suitability for simulation
  - *many others...*

9

9

# Parametric Representations

Curves: $\boldsymbol{x} = \boldsymbol{x}(u)$ $\quad \boldsymbol{x} \in \Re^n \quad u \in \Re$

Surfaces: $\boldsymbol{x} = \boldsymbol{x}(u, v)$ $\quad \boldsymbol{x} \in \Re^n \quad u, v \in \Re$

$\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{u}) \quad \boldsymbol{u} \in \Re^2$

Volumes: $\boldsymbol{x} = \boldsymbol{x}(u, v, w)$ $\quad \boldsymbol{x} \in \Re^n \quad u, v, w \in \Re$

$\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{u}) \quad \boldsymbol{u} \in \Re^3$

*and so on...*

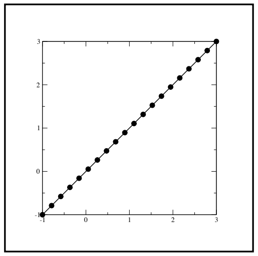
Note: a vector function is really $\boldsymbol{n}$ scalar functions

10

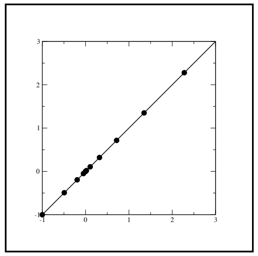
10

# Parametric Rep. Non-unique

- Same curve/surface may have multiple formulae



$\boldsymbol{x}(u) = [u, u]$



$\boldsymbol{x}(u) = [u^3, u^3]$

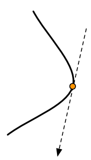




# Simple Differential Geometry

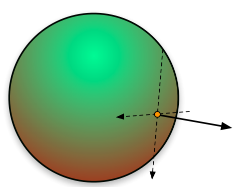
- Tangent to curve

$$\boldsymbol{t}(u) = \left.\frac{\partial \boldsymbol{x}}{\partial u}\right|_u$$



- Tangents to surface

$$\boldsymbol{t}_u(u,v) = \left.\frac{\partial \boldsymbol{x}}{\partial u}\right|_{u,v} \qquad \boldsymbol{t}_v(u,v) = \left.\frac{\partial \boldsymbol{x}}{\partial v}\right|_{u,v}$$



- Normal of surface

$$\hat{\boldsymbol{n}} = \frac{\boldsymbol{t}_u \times \boldsymbol{t}_v}{||\boldsymbol{t}_u \times \boldsymbol{t}_v||}$$

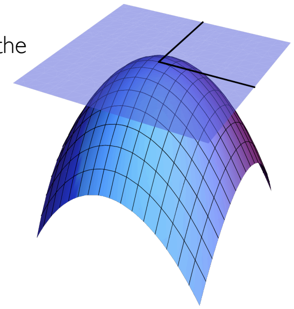
- Also: curvature, curve normals, curve bi-normal, ***others…***
- Degeneracies: $\partial \boldsymbol{x}/\partial u = 0$ or $\boldsymbol{t}_u \times \boldsymbol{t}_v = 0$

# Tangent Space

- The *tangent space* at a point on a surface is the vector space spanned by

$$\frac{\partial \mathbf{x}(\mathbf{u})}{\partial u} \qquad \frac{\partial \mathbf{x}(\mathbf{u})}{\partial v}$$

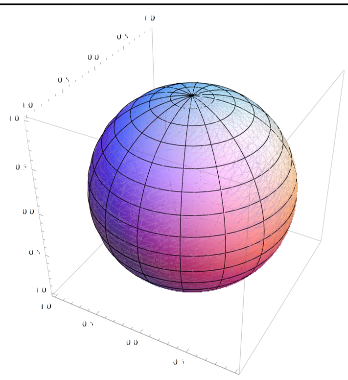
  - Definition assumes that these directional derivatives are linearly independent.
  - Tangent space of surface may exist even if the parameterization is bad
- For surface the space is a plane
  - Generalized to higher dimension manifolds



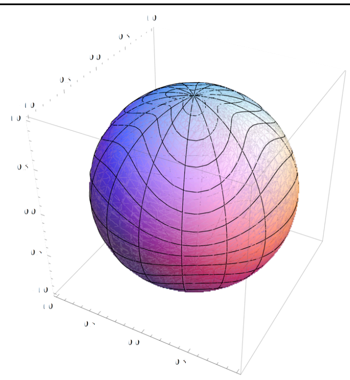




# Non Orthogonal Tangents



$$\begin{bmatrix} \cos(\theta 2\pi)\cos(\phi pi/2) \\ \sin(\theta 2\pi)\cos(\phi\pi/2) \\ \sin(\phi\pi/2) \end{bmatrix}$$

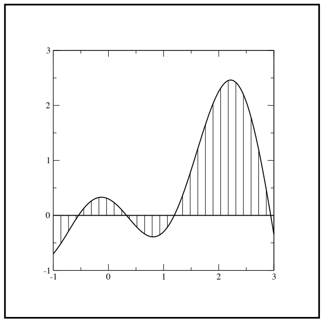


$$\begin{bmatrix} \cos(2\pi\theta)\cos\left(\frac{1}{2}\pi\left(\frac{1}{2}(1-|\phi|)\cos(6\pi\theta)\phi+\phi\right)\right) \\ \cos\left(\frac{1}{2}\pi\left(\frac{1}{2}(1-|\phi|)\cos(6\pi\theta)\phi+\phi\right)\right)\sin(2\pi\theta) \\ \sin\left(\frac{1}{2}\pi\left(\frac{1}{2}(1-|\phi|)\cos(6\pi\theta)\phi+\phi\right)\right) \end{bmatrix}$$

$\theta \in [0..1] \quad \phi \in [-1..1]$

# Discretization

- Arbitrary curves have an uncountable number of parameters




***i.e.* specify function value at all points on real number line**

15

15

# Discretization

- Arbitrary curves have an uncountable number of parameters
- Pick ***complete*** set of basis functions
  - Polynomials, Fourier series, ***etc.***

$$x(u) = \sum_{i=0}^{\infty} c_i \phi_i(u)$$

- Truncate set at some reasonable point

$$x(u) = \sum_{i=0}^{3} c_i \phi_i(u) = \sum_{i=0}^{3} c_i\, u^i$$

- Function represented by the vector (list) of $c_i$
- The $c_i$ may themselves be vectors

$$\boldsymbol{x}(u) = \sum_{i=0}^{3} \boldsymbol{c}_i \phi_i(u)$$

16

16

# Polynomial Basis

- Power Basis

$$x(u) = \sum_{i=0}^{d} c_i\, u^i$$

$$x(u) = \boldsymbol{C} \cdot \boldsymbol{\mathcal{P}}^d \qquad \boldsymbol{C} = [c_0, c_1, c_2, \ldots, c_d] \qquad \boldsymbol{\mathcal{P}}^d = [1, u, u^2, \ldots, u^d]$$

The elements of $\boldsymbol{\mathcal{P}}^d$ are ***linearly independant***

***i.e.*** no good approximation

$$u^k \not\approx \sum_{i \neq k} c_i\, u^i$$

Skipping something would lead to bad results... odd stiffness

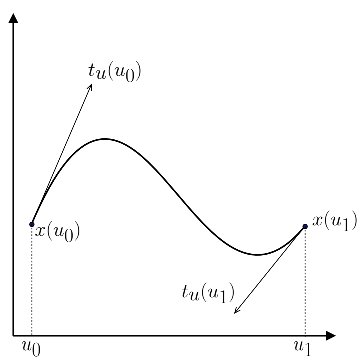
17

17

# Specifying a Curve

Given desired values (constraints) how do we determine
the coefficients for cubic power basis?

For now, assume
$u_0 = 0 \quad u_1 = 1$



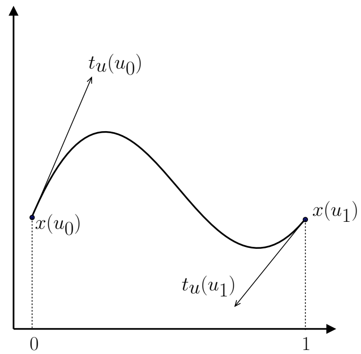

18

18

# Specifying a Curve

Given desired values (constraints) how do we determine
the coefficients for cubic power basis?

$$x(0) = c_0 \quad = x_0$$
$$x(1) = \Sigma\, c_i \; = x_1$$
$$x'(0) = c_1 \quad = x'_0$$
$$x'(1) = \Sigma\, i\, c_i = x'_1$$




19

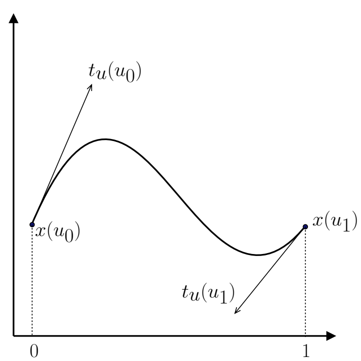
19

# Specifying a Curve

Given desired values (constraints) how do we determine
the coefficients for cubic power basis?

$$\begin{bmatrix} x_0 \\ x_1 \\ x'_0 \\ x'_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$\mathbf{p} = \mathbf{B} \cdot \mathbf{c}$$




20

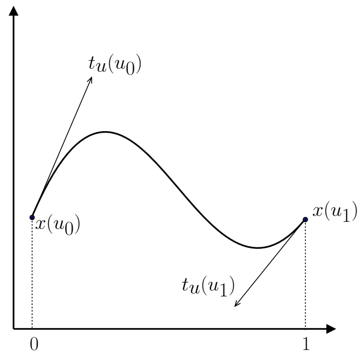
20

# Specifying a Curve

Given desired values (constraints) how do we determine
the coefficients for cubic power basis?

$$\mathbf{c} = \beta_{\text{H}} \cdot \mathbf{p}$$

$$\boldsymbol{\beta}_{\text{H}} = \mathbf{B}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & 1 \\ 2 & -2 & 1 & 1 \end{bmatrix}$$




21



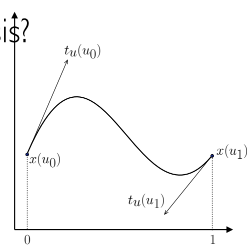
# Specifying a Curve

Given desired values (constraints) how do we determine
the coefficients for cubic power basis?

$$\mathbf{c} = \beta_{\text{H}} \cdot \mathbf{p}$$

$$x(u) = \mathcal{P}^3 \cdot \mathbf{c} = \boxed{\mathcal{P}^3 \boldsymbol{\beta}_{\text{H}}} \mathbf{p}$$

$$= \begin{bmatrix} 1 + 0u - 3u^2 + 2u^3 \\ 0 + 0u + 3u^2 - 2u^3 \\ 0 + 1u - 2u^2 + 1u^3 \\ 0 + 0u - 1u^2 + 1u^3 \end{bmatrix} \mathbf{p}$$




$$\boldsymbol{\beta}_{\text{H}} = \mathbf{B}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & 1 \\ 2 & -2 & 1 & 1 \end{bmatrix}$$

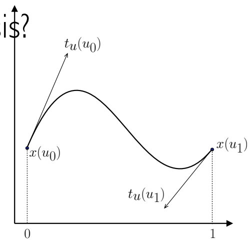
22

# Specifying a Curve

Given desired values (constraints) how do we determine
the coefficients for cubic power basis?

$$\mathbf{c} = \beta_H \cdot \mathbf{p}$$

$$x(u) = \begin{bmatrix} 1 + 0u - 3u^2 + 2u^3 \\ 0 + 0u + 3u^2 - 2u^3 \\ 0 + 1u - 2u^2 + 1u^3 \\ 0 + 0u - 1u^2 + 1u^3 \end{bmatrix} \mathbf{p}$$




$$x(u) = \sum_{i=0}^{3} p_i b_i(u)$$

Hermite basis functions

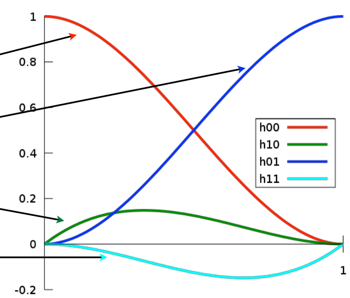


23

# Specifying a Curve

Given desired values (constraints) how do we determine
the coefficients for cubic power basis?

$$x(u) = \begin{bmatrix} 1 + 0u - 3u^2 + 2u^3 \\ 0 + 0u + 3u^2 - 2u^3 \\ 0 + 1u - 2u^2 + 1u^3 \\ 0 + 0u - 1u^2 + 1u^3 \end{bmatrix} \mathbf{p}$$




$$x(u) = \sum_{i=0}^{3} p_i b_i(u)$$

Hermite basis functions



24

# Hermite Basis

- Specify curve by
  - Endpoint values
  - Endpoint tangents (derivatives)
- Parameter interval is arbitrary (most times)
  - Don't need to recompute basis functions
- These are ***cubic*** Hermite
  - Could do construction for any odd degree
  - $(d-1)/2$ derivatives at end points

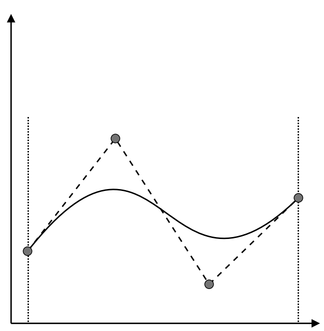
25

25

# Cubic Bézier

- Similar to Hermite, but specify tangents indirectly

$$x_0 = p_0$$
$$x_1 = p_3$$
$$x_0' = 3(p_1 - p_0)$$
$$x_1' = 3(p_3 - p_2)$$



Note: all the control points are points in space, no tangents.

26

26

# Cubic Bézier

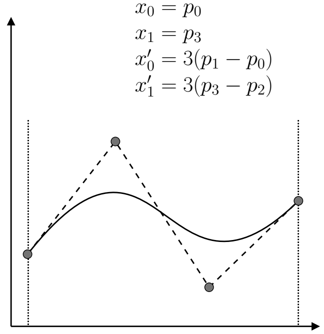
- Similar to Hermite, but specify tangents indirectly

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \mathbf{c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \mathbf{p}$$

$$\mathbf{c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \mathbf{p}$$

$$\mathbf{c} = \boldsymbol{\beta}_Z\, \mathbf{p}$$

$$x_0 = p_0$$
$$x_1 = p_3$$
$$x'_0 = 3(p_1 - p_0)$$
$$x'_1 = 3(p_3 - p_2)$$



27



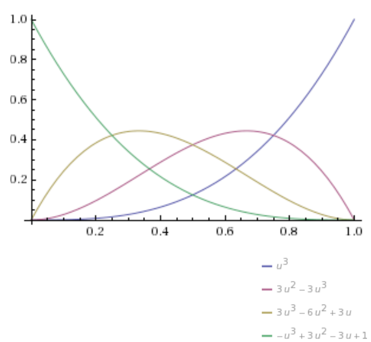
# Cubic Bézier

Bézier basis functions

$$\mathbf{c} = \boldsymbol{\beta}_Z\, \mathbf{p} \qquad \mathbf{c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \mathbf{p}$$

$$x(u) = \mathcal{P}^3 \cdot \mathbf{c}$$

$$x(u) = \begin{bmatrix} 1 - 3u + 3u^2 - 1u^3 \\ 0 + 3u - 6u^2 + 3u^3 \\ 0 + 0u + 3u^2 - 3u^3 \\ 0 + 0u + 0u^2 + 1u^3 \end{bmatrix} \mathbf{p}$$




28

# Changing Bases

- Power basis, Hermite, and Bézier all are still just cubic polynomials
  - The three basis sets all span the same space
  - Like different axes in
- Changing basis $\Re^{X}$ $\Re^4$

$$\mathbf{c} = \boldsymbol{\beta}_{\mathrm{Z}}\, \mathbf{p}_{\mathrm{Z}}$$

$$\mathbf{c} = \boldsymbol{\beta}_{\mathrm{H}}\, \mathbf{p}_{\mathrm{H}}$$

$$\mathbf{p}_{\mathrm{Z}} = \boldsymbol{\beta}_{\mathrm{Z}}^{-1}\, \boldsymbol{\beta}_{\mathrm{H}}\, \mathbf{p}_{\mathrm{H}}$$

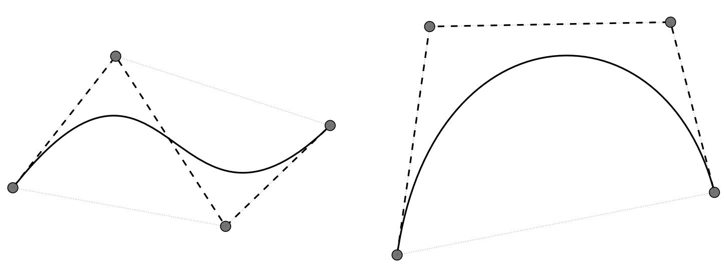




# Useful Properties of a Basis

- Convex Hull
  - All points on curve inside convex hull of control points
    - Bézier basis has convex hull property

$$\sum_i b_i(u) = 1 \qquad b_i(u) \geq 0 \qquad \forall u \in \Omega$$



## Useful Properties of a Basis

- Invariance under class of transforms
  - Transforming curve is same as transforming control points
    - Bézier basis invariant for affine transforms
    - Bézier basis NOT invariant for perspective transforms
      - NURBS are though...

$$\boldsymbol{x}(u) = \sum_i \boldsymbol{p}_i b_i(u) \Leftrightarrow \mathcal{T}\boldsymbol{x}(u) = \sum_i (\mathcal{T}\boldsymbol{p}_i) b_i(u)$$





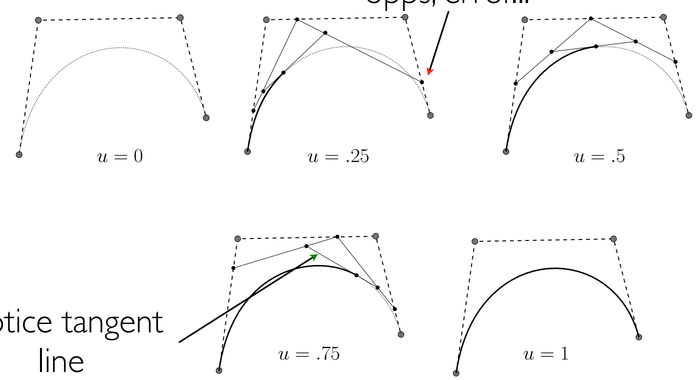
## Useful Properties of a Basis

- Local support
  - Changing one control point has limited impact on entire curve
- Nice subdivision rules
- Orthogonality ( $\int_\Omega b_i(u) b_j(u) \mathrm{d}u = \delta_{ij}$ )
- Fast evaluation scheme
- Interpolation -vs- approximation

# DeCasteljau Evaluation

- A geometric evaluation scheme for Bézier

opps, error...

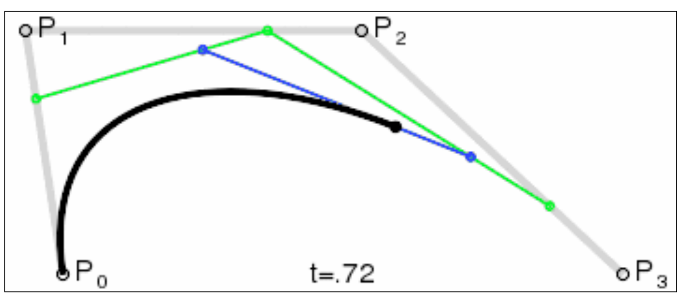



Notice tangent line

33

# DeCasteljau Evaluation

Blue line is always tangent to curve.



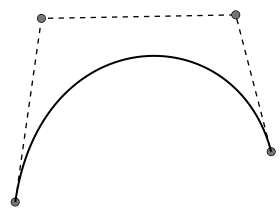

From Wikipedia

34

# Adaptive Tessellation

- Midpoint test subdivision
- Possible problem
  - Simple solution if curve basis has ***convex hull*** property


If curve inside convex hull and the convex hull is nearly flat: curve is nearly flat and can be drawn as straight line



Better: draw convex hull

Works for Bézier because the ends are interpolated





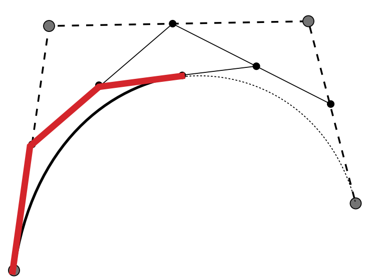
# Bézier Subdivision

- Form control polygon for half of curve by evaluating at $u$=0.5

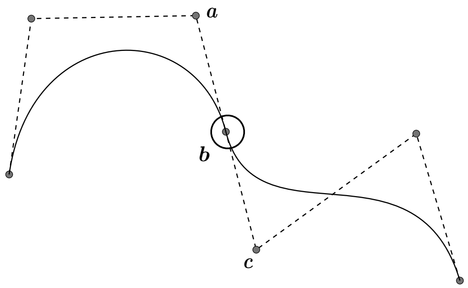
Repeated subdivision makes smaller/flatter segments

Also works for surfaces...

We'll extend this idea later on...



# Joining




$\mathcal{C}^0 \Leftrightarrow \boldsymbol{b} = \boldsymbol{b}$

$\mathcal{C}^1 \Leftrightarrow \boldsymbol{b} - \boldsymbol{a} = \boldsymbol{c} - \boldsymbol{b}$

$\mathcal{G}^1 \Leftrightarrow \frac{\boldsymbol{b} - \boldsymbol{a}}{||\boldsymbol{b} - \boldsymbol{a}||} = \frac{\boldsymbol{c} - \boldsymbol{b}}{||\boldsymbol{c} - \boldsymbol{b}||}$

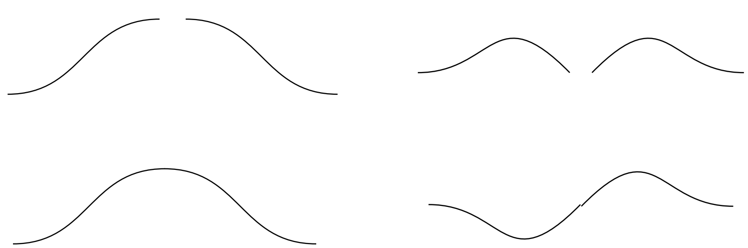
If you change ***a***, ***b***, or ***c*** you must change the others

But if you change ***a***, ***b***, or ***c*** you do not have to change beyond those three. *LOCAL SUPPORT*



# "Hump" Functions

- Constraints at joining can be built in to make new basis



# Tensor-Product Surfaces

- Surface is a curve swept through space
- Replace control points of curve with other curves

$$x(u,v) = \Sigma_i\, p_i\, b_i(u)$$
$$\Sigma_i\, q_i(v)\, b_i(u) \qquad q_i(v) = \Sigma_j\, p_{ji}\, b_j(v)$$

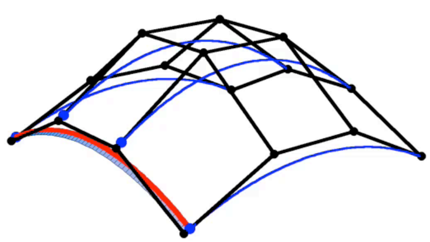
$$x(u,v) = \sum_{ij} p_{ij} b_i(u) b_j(v) \qquad b_{ij}(u,v) = b_i(u) b_j(v)$$

$$x(u,v) = \sum_{ij} p_{ij} b_{ij}(u,v)$$

39

39

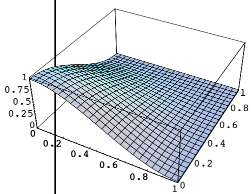
# Tensor-Product Surfaces

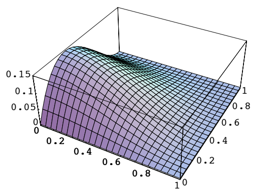


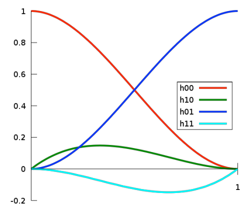
40

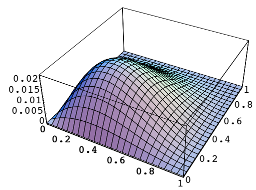
40

# Hermite Surface Bases



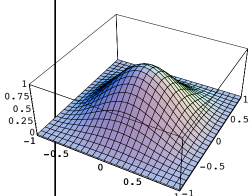



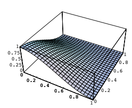

Plus symmetries...

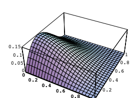
41

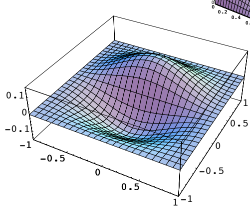
41

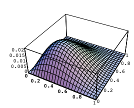
# Hermite Surface Hump Functions

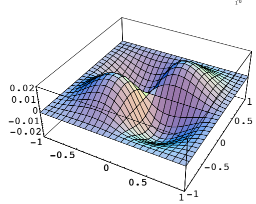






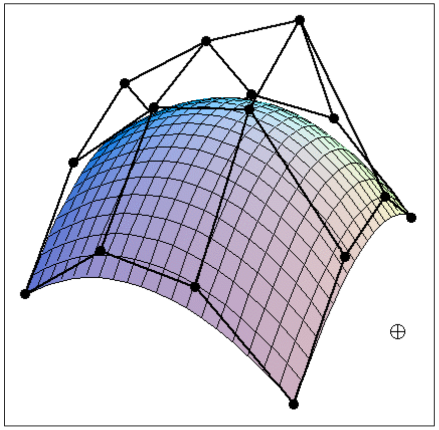



Plus symmetries...

42

42

# Bézier Surface Patch

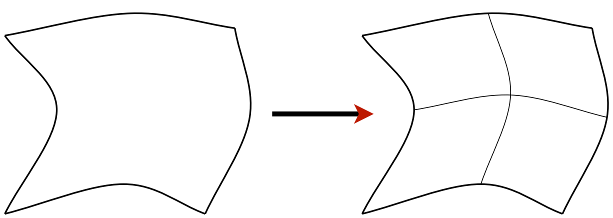


Bezier surface and 4 x 4 array of control points



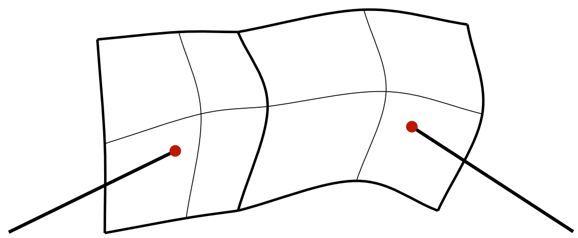
# Adaptive Tessellation

- Given surface patch
  - If close to flat: draw it
  - Else subdivide 4 ways



# Adaptive Tessellation

• Avoid cracking

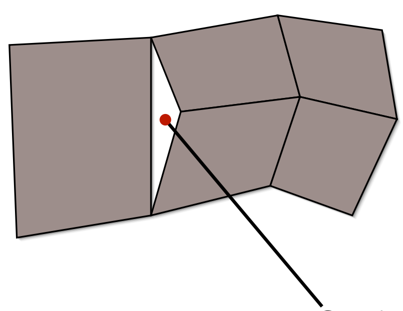



45

45

# Adaptive Tessellation

• Avoid cracking



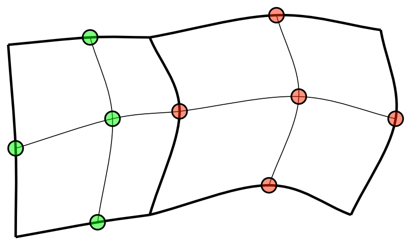

Cracks may be okay in some contexts...

46

46

# Adaptive Tessellation

- Avoid cracking

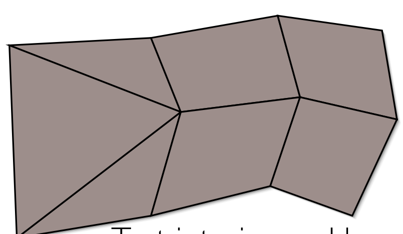


47

47

# Adaptive Tessellation

- Avoid cracking



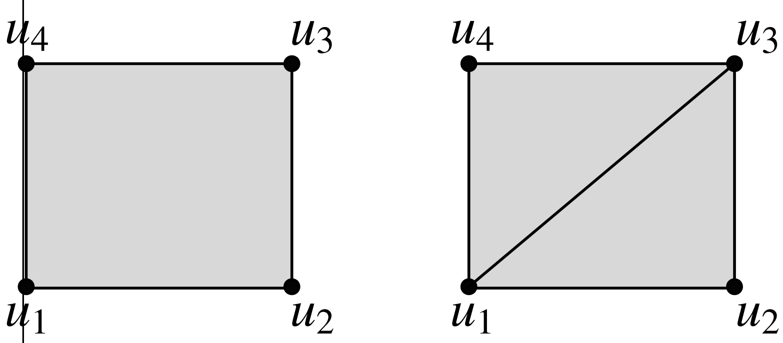
Test interior and boundary of patch
Split boundary based on boundary test
Table of polygon patterns
May wish to avoid "slivers"

48

48

# Adaptive Tessellation

- Triangle Based Method (no cracks)

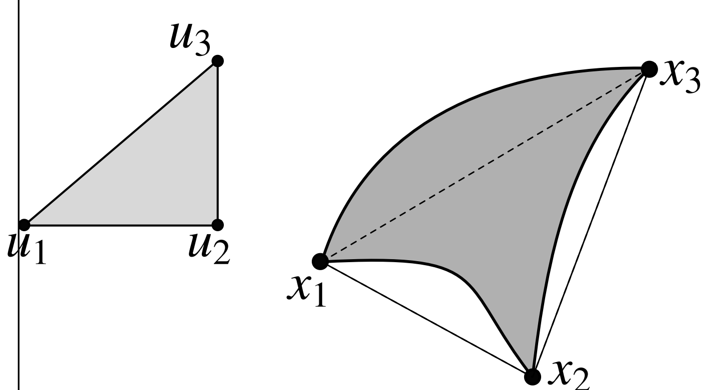



49

49

# Adaptive Tessellation

- Triangle Based Method (no cracks)

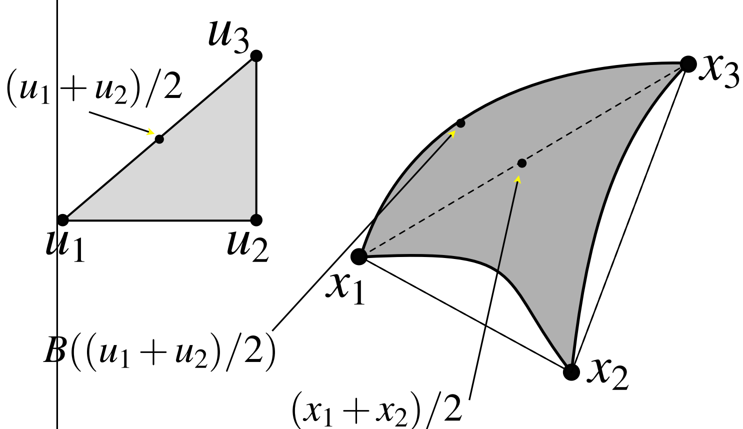



50

50

# Adaptive Tessellation

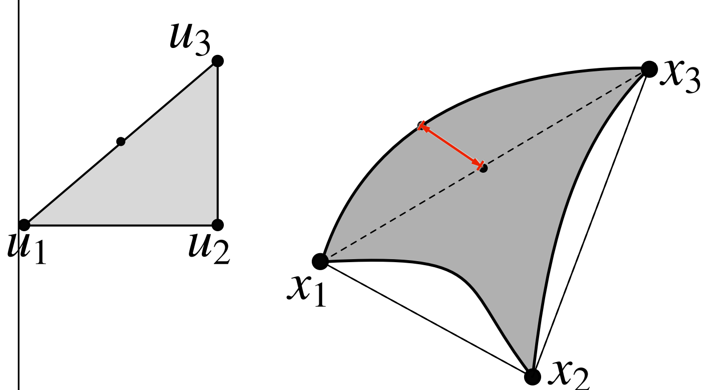
- Triangle Based Method (no cracks)




51

# Adaptive Tessellation

- Triangle Based Method (no cracks)

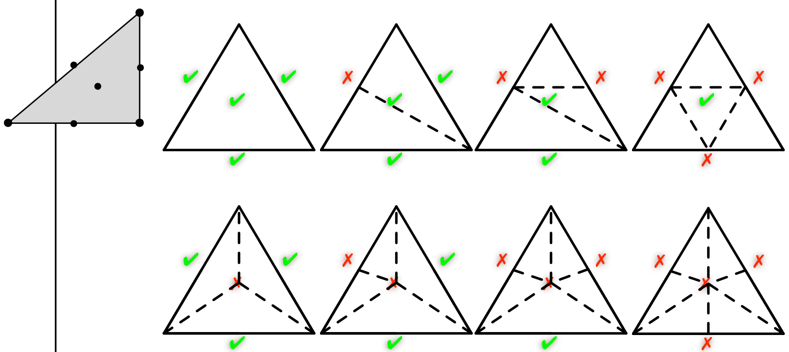



$||B((u_1 + u_2)/2) - (x_1 + x_2)/2|| < \tau$ ?

52

# Adaptive Tessellation

- Triangle Based Method (no cracks)

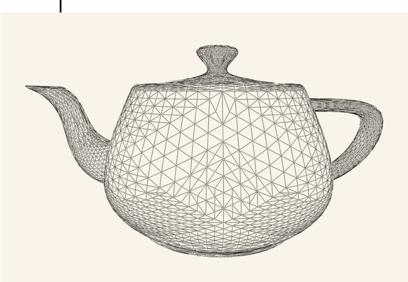


Center test tends to generate slivers.
Often better to leave it out.

53

53

# Adaptive Tessellation



Without center test

With center test

Yiding Jia, CS184 S08

54

54

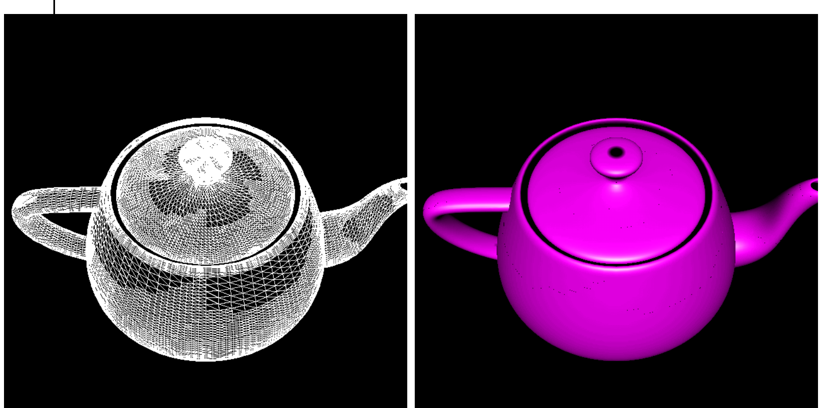
# Adaptive Tessellation

Second row shows typical error of swapping tests.

Yiding Jia, CS184 S08 -- I broke his code to make this example.



# Adaptive Tessellation



Visible artifacts from cracks.

Apollo Ellis, CS184 S08

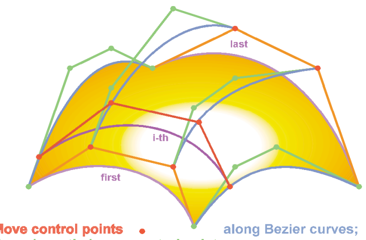
# Bezier Surfaces. Smooth Operators.

```
# given the control points of a bezier curve
# and a parametric value, return the curve
# point and derivative
bezcurveinterp(curve, u)
    # first, split each of the three segments
    # to form two new ones AB and BC
    A = curve[0] * (1.0-u) + curve[1] * u
    B = curve[1] * (1.0-u) + curve[2] * u
    C = curve[2] * (1.0-u) + curve[3] * u

    # now, split AB and BC to form a new segment DE
    D = A * (1.0-u) + B * u
    E = B * (1.0-u) + C * u

    # finally, pick the right point on DE,
    # this is the point on the curve
    p = D * (1.0-u) + E * u

    # compute derivative also
    dPdu = 3 * (E - D)

    return p, dPdu
```

```
# given a control patch and (u,v) values, find
# the surface point and normal
bezpatchinterp(patch, u, v)
    # build control points for a Bezier curve in v
    vcurve[0] = bezcurveinterp(patch[0][0:3], u)
    vcurve[1] = bezcurveinterp(patch[1][0:3], u)
    vcurve[2] = bezcurveinterp(patch[2][0:3], u)
    vcurve[3] = bezcurveinterp(patch[3][0:3], u)

    # build control points for a Bezier curve in u
    ucurve[0] = bezcurveinterp(patch[0:3][0], v)
    ucurve[1] = bezcurveinterp(patch[0:3][1], v)
    ucurve[2] = bezcurveinterp(patch[0:3][2], v)
    ucurve[3] = bezcurveinterp(patch[0:3][3], v)

    # evaluate surface and derivative for u and v
    p, dPdv = bezcurveinterp(vcurve, v)
    p, dPdu = bezcurveinterp(ucurve, u)

    # take cross product of partials to find normal
    n = cross(dPdu, dPdv)
    n = n / length(n)

    return p, n
```

## Bicubic Bezier Patch

Continuously Moved and Deformed Bezier Curve




Move control points • along Bezier curves;
these have their own control points •;
leading to a total of 16 control points for the cubic case.

```
# given a patch, perform uniform subdivision
subdividepatch(patch, step)
    # compute how many subdivisions there
    # are for this step size
    numdiv = ((1 + epsilon) / step)

    # for each parametric value of u
    for (iu = 0 to numdiv)
        u = iu * step

        # for each parametric value of v
        for (iv = 0 to numdiv)
            v = iv * step

            # evaluate surface
            p, n = bezpatchinterp(patch, u, v)
            savesurfacepointandnormal(p,n)
```

Split?

| e3 | e2 | e1 | |
|---|---|---|---|
| 0 | 0 | 0 | output as is |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 1 | 0 | 0 | |
| 0 | 1 | 1 | |
| 1 | 1 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 1 | |