Computer Graphics

-- Explicit Representation

Junjie Cao @ DLUT

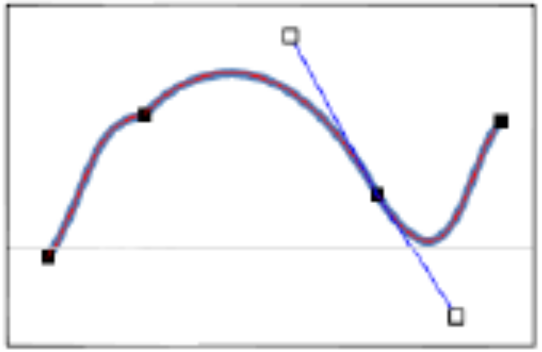
Spring 2019

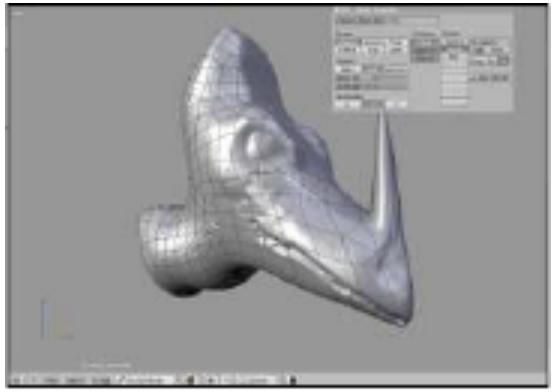
http://jjcao.github.io/ComputerGraphics/

Explicit Shape Representation

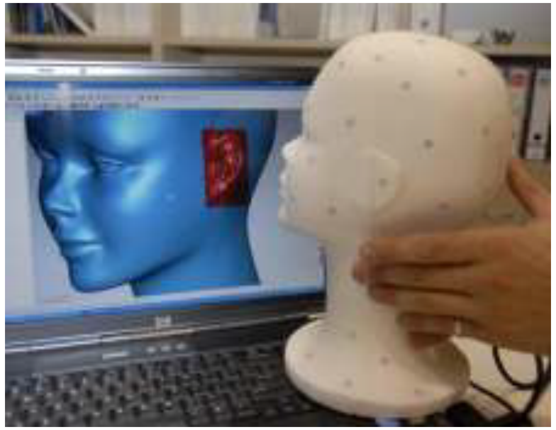
Where does the shape come from?

- **Modeling "by hand"**
  - Higher-level representations,
  - Amenable to modification, control





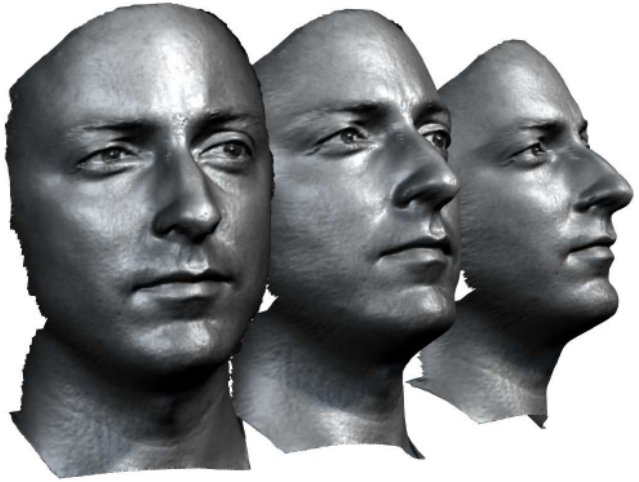
- **Acquired real-world objects**
  - Discrete sampling
  - Points, meshes



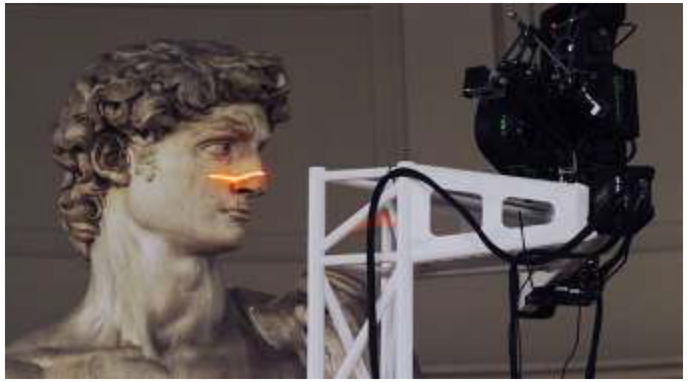
Shape Acquisition

Sampling of real world objects

- Scanners
  - Laser
  - Depth imaging
- Properties & Operations
  - Potentially noisy, with outliers
  - Registration of multiple images
  - Non-uniform sampling, sparse, holes

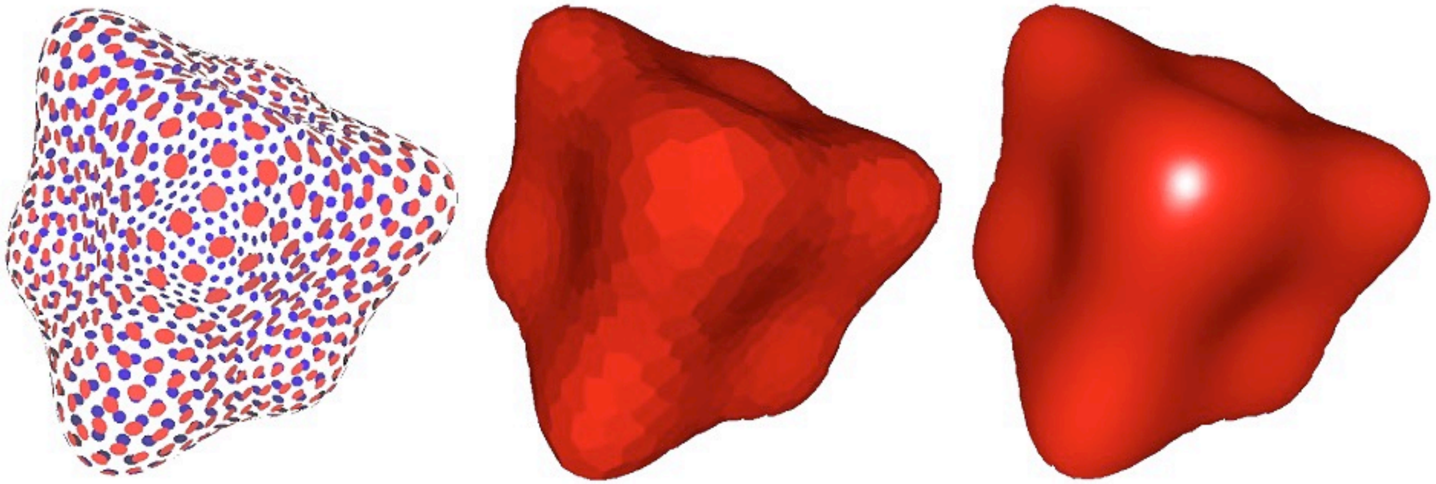


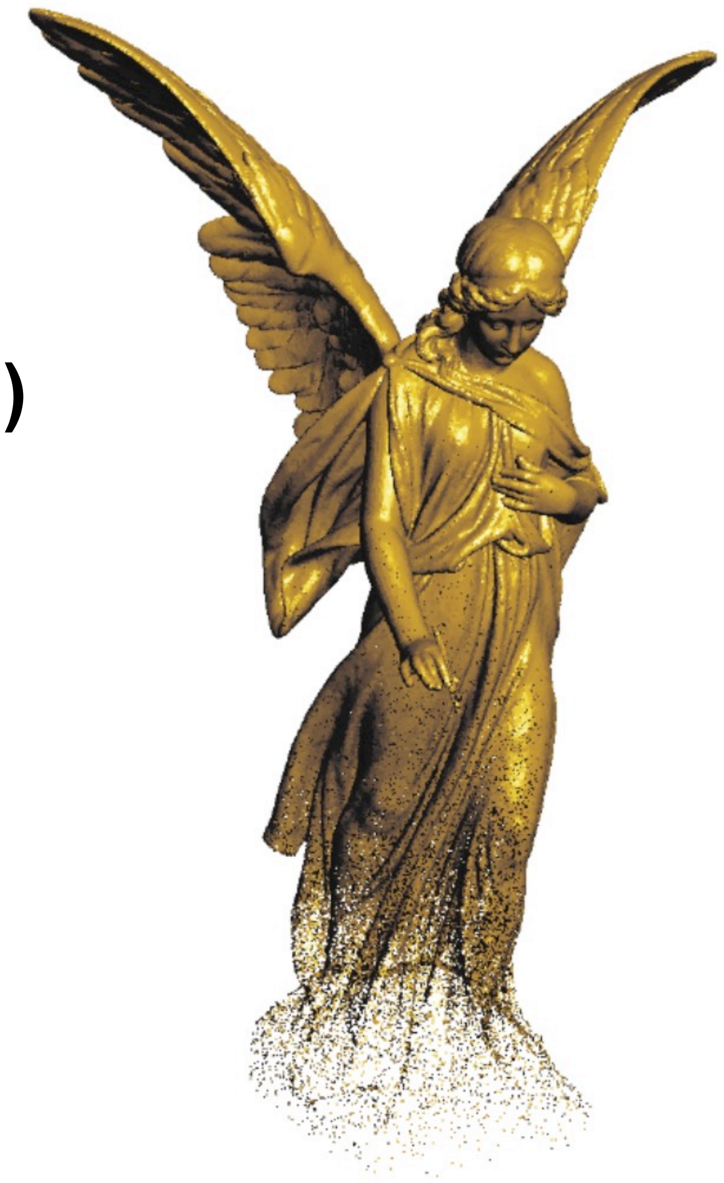
180 frames per second (From David Gu)



What about explicit representations?

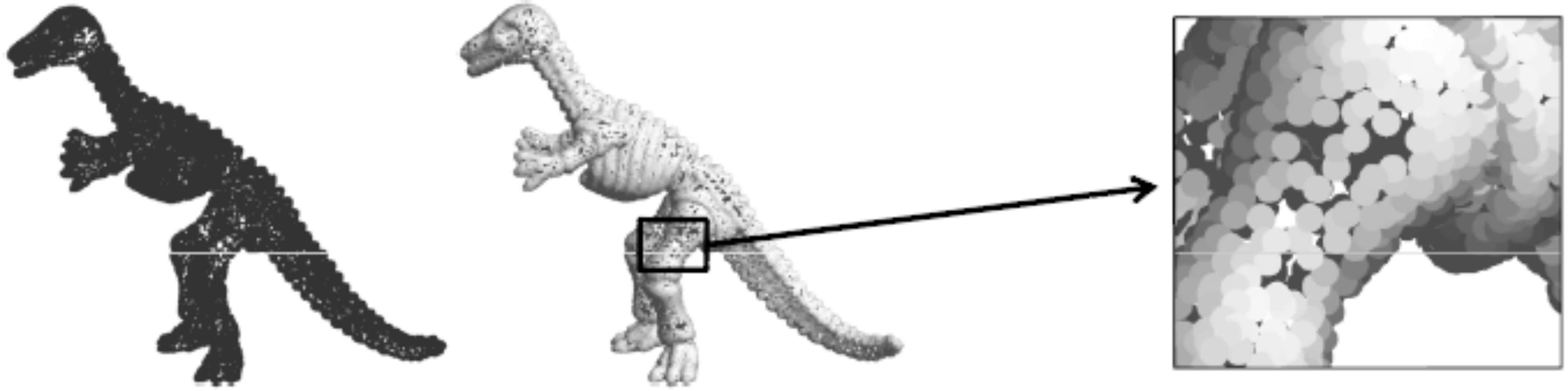
- Easiest representation: list of points (x,y,z)
- Often augmented with *normals*
- Easily represent any kind of geometry
- Useful for LARGE datasets (>>1 point/pixel)
- Difficult to draw in undersampled regions
- Hard to do processing / simulation





Points: Neighborhood information

- Why do we need neighbors?

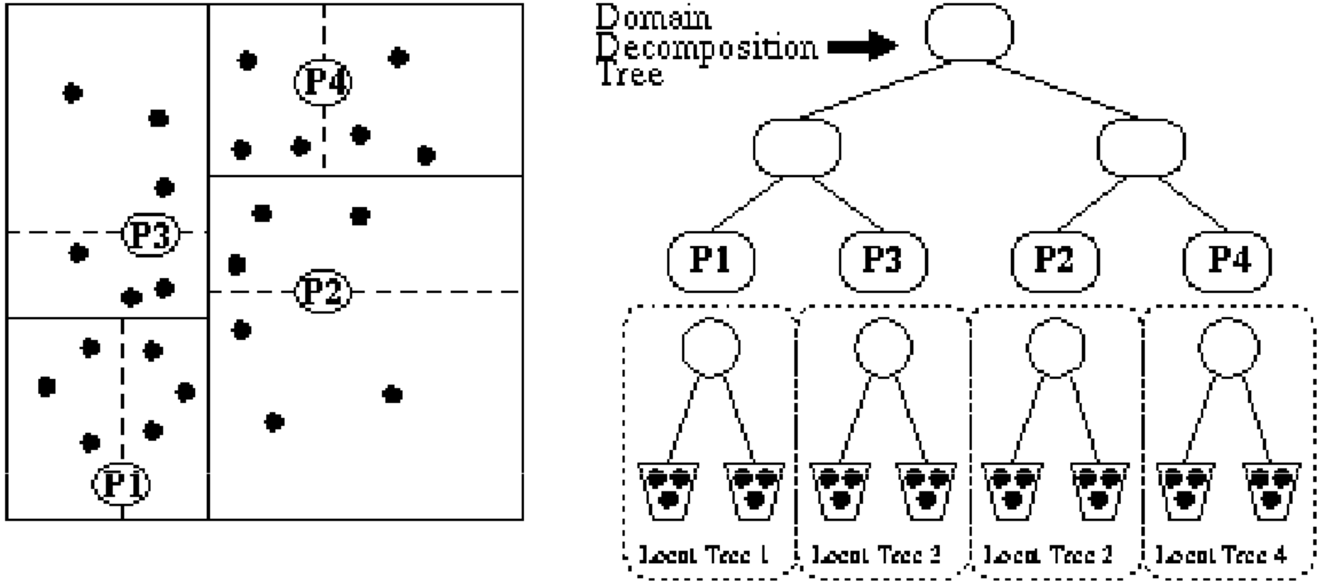


need normals (for shading) upsampling – need to count density

- Need sub-linear implementations of
  - K-nearest neighbors to point x (knn)
  - In radius search
- Efficient point processing & modeling requires a spatial partitioning data structure

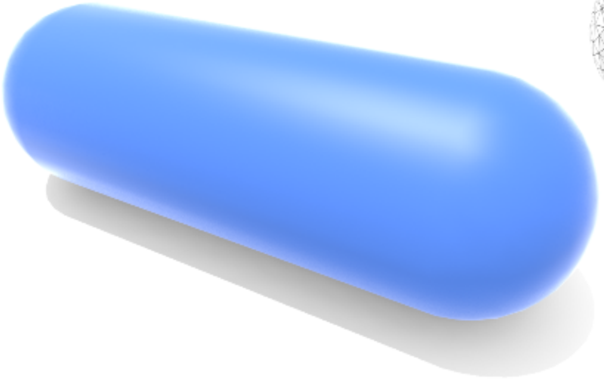
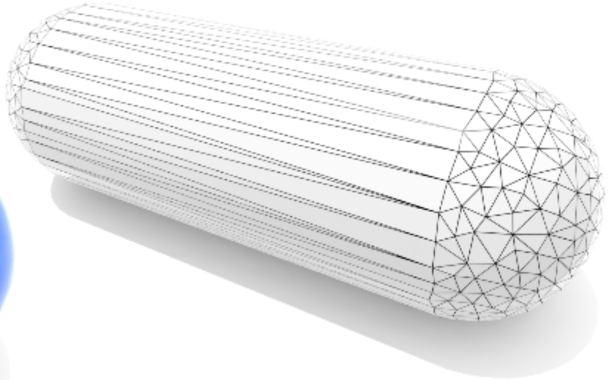
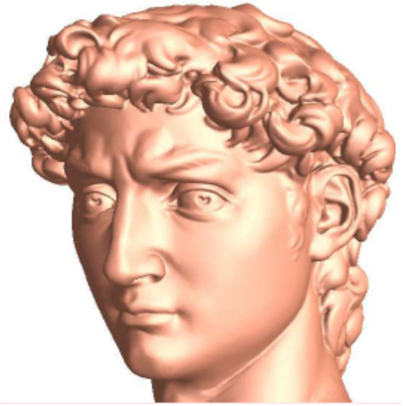
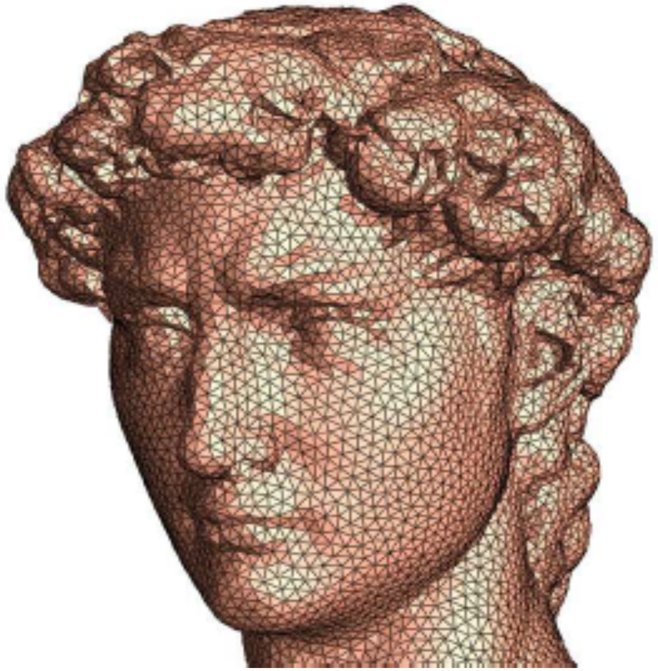
Kd-Tree

- Each cell is individually split along the median into two cells
- Same amount of points in cells
- Perfectly balanced tree
- Proximity search similar to the recursive search in an Octree.
- More data storage required for inhomogeneous cell dimensions



Polygon Mesh (Explicit)

- Store vertices *and* polygons (most often triangles or quads)
- Easier to do processing/simulation, adaptive sampling
- More complicated data structures
- **Perhaps most common representation in graphics**

Parametric Representation

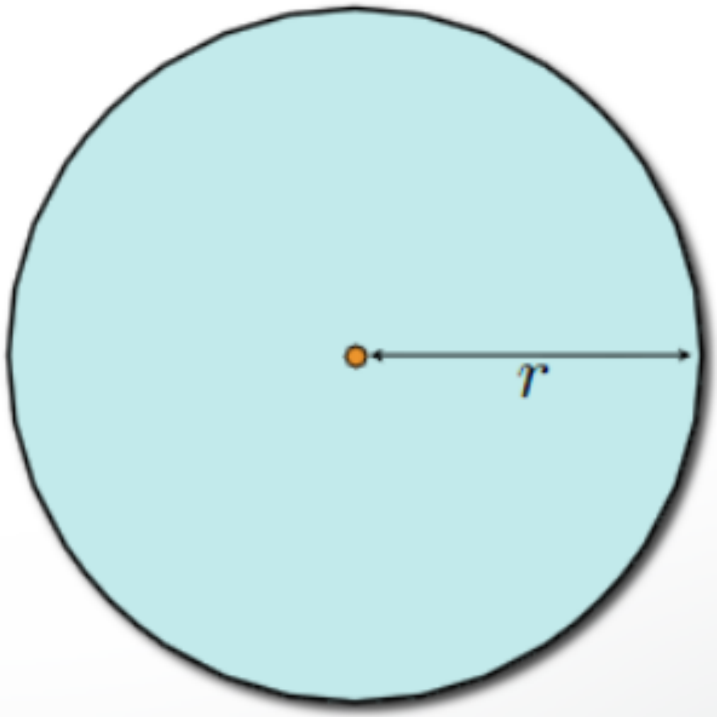
– Surface is the range of a function

$$\mathbf{f} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3, \quad \mathcal{S}_\Omega = \mathbf{f}(\Omega)$$

2D example: A Circle

$$\mathbf{f} : [0, 2\pi] \to \mathbb{R}^2$$

$$\mathbf{f}(t) = \begin{pmatrix} r\cos(t) \\ r\sin(t) \end{pmatrix}$$



Parametric Representation

Polynomials are computable functions

$$f(t) \;=\; \sum_{i=0}^{p} c_i\, t^i \;=\; \sum_{i=0}^{p} \tilde{c}_i\, \phi_i(t)$$

Taylor expansion up to degree p

$$g(h) \;=\; \sum_{i=0}^{p} \frac{1}{i!}\, g^{(i)}(0)\; h^i \;+\; O\big(h^{p+1}\big)$$

Error for approximation g by polynomial f

$$f(t_i) = g(t_i)\,, \quad 0 \le t_0 < \cdots < t_p \le h$$

$$|f(t) - g(t)| \;\le\; \frac{1}{(p+1)!}\, \max f^{(p+1)} \prod_{i=0}^{p} (t - t_i) \;=\; O\Big(h^{(p+1)}\Big)$$

Parametric Representation

Approximation error is $O(h^{p+1})$

Improve approximation quality by

- increasing p... higher order polynomials
- decreasing h ... shorter / more segments

Issues

- smoothness of the target data ($\max_t f^{(p+1)}(t)$)
- smoothness condition between segments

Parametric Representation

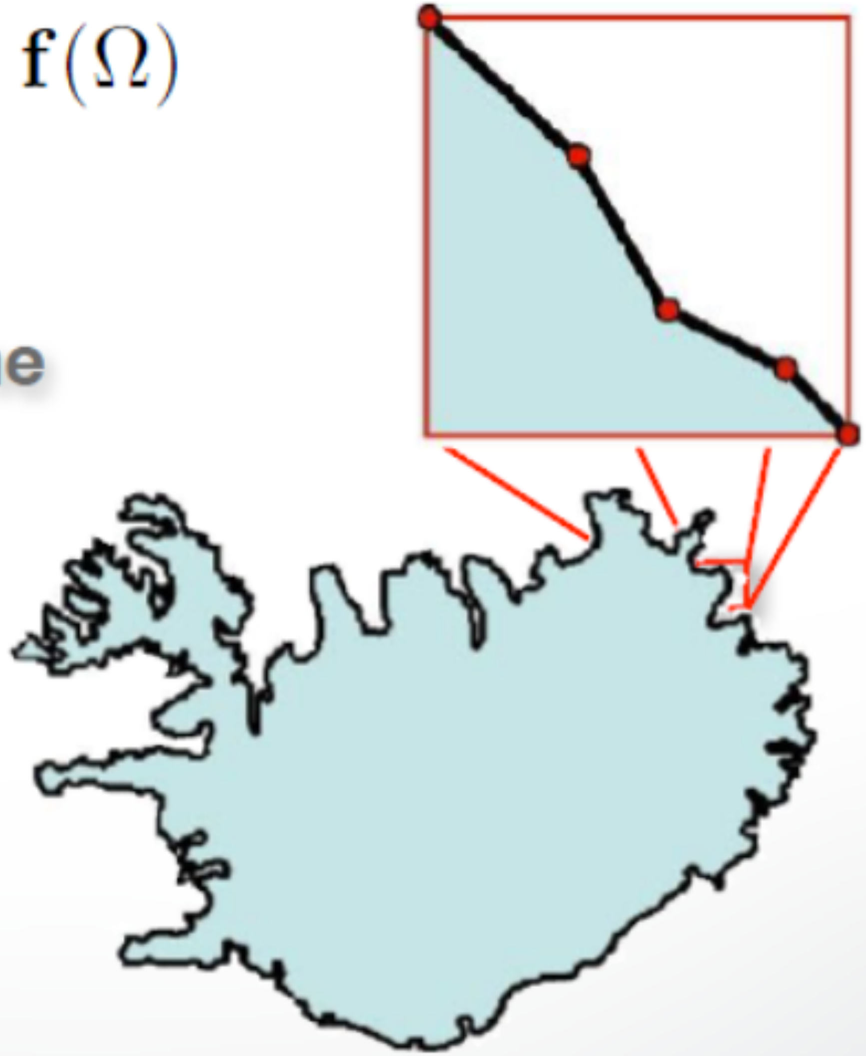
Surface is the range of a function

$$\mathbf{f} : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad \mathcal{S}_\Omega = \mathbf{f}(\Omega)$$

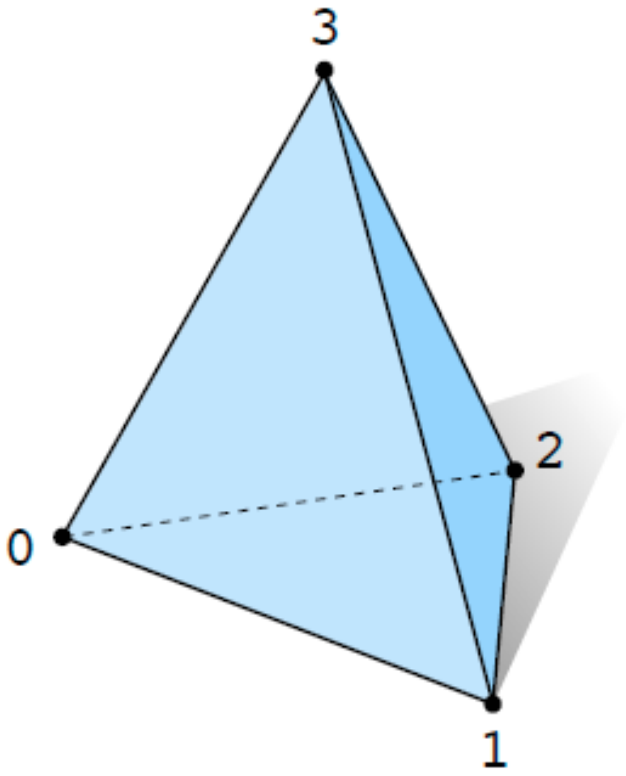
2D example: Island coast line

$$\mathbf{f} : [0, 2\pi] \rightarrow \mathbb{R}^2$$

$$\mathbf{f}(t) = \begin{pmatrix} ? \\ ? \end{pmatrix}$$



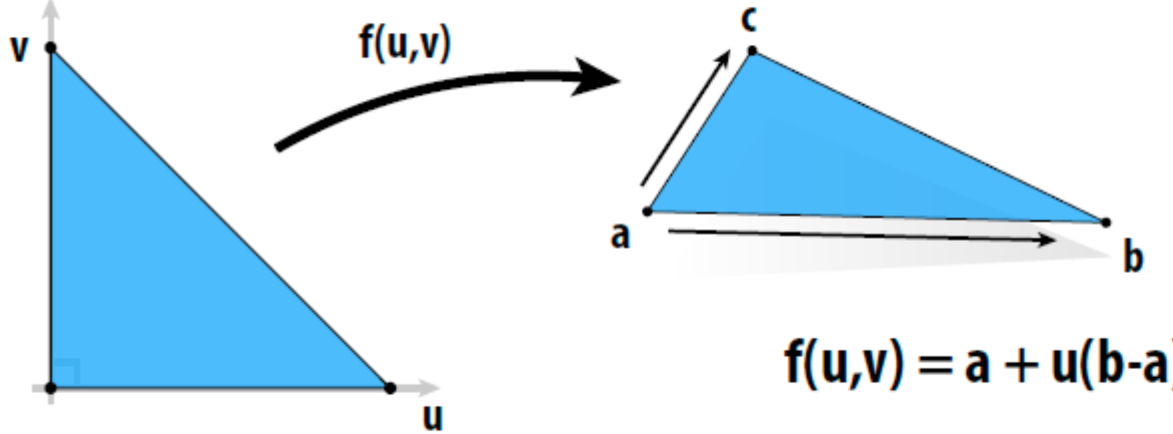
Triangle Mesh (Explicit)



- **Store vertices as triples of coordinates (x,y,z)**
- **Store triangles as triples of indices (i,j,k)**
- **E.g., tetrahedron:**

	VERTICES			TRIANGLES		
	x	y	z	i	j	k
0:	−1	−1	−1	0	2	1
1:	1	−1	1	0	3	2
2:	1	1	−1	3	0	1
3:	−1	1	1	3	1	2

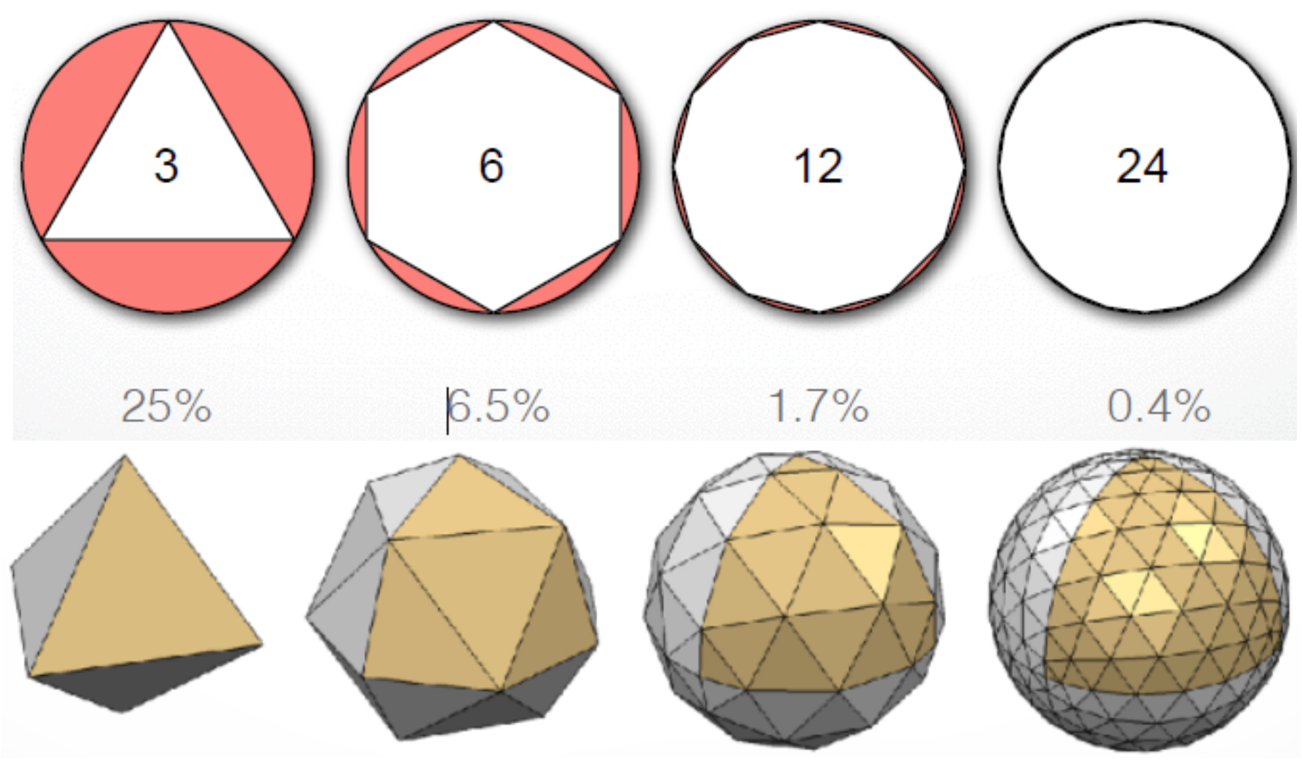
- **Can think of triangle as *affine* map from plane into space:**

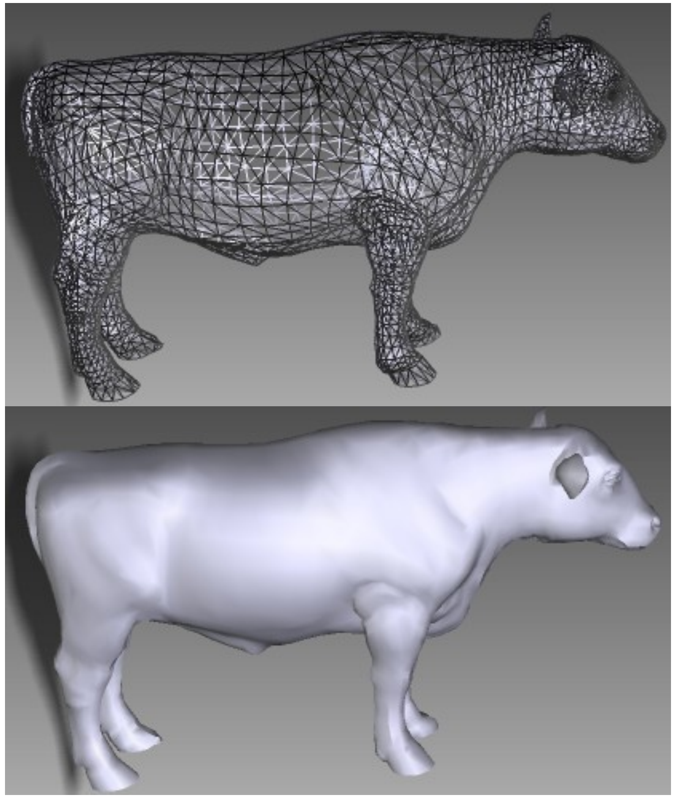


$$f(u,v) = a + u(b\text{-}a) + v(c\text{-}a)$$

Polygonal meshes are a good compromise

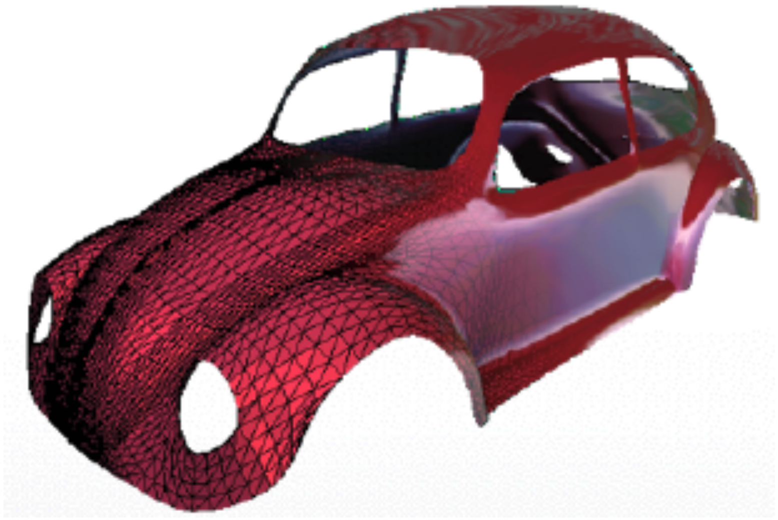
- **Theorem** Given a smooth surface S and a given error $\varepsilon > 0$, there exists a piecewise linear surface (mesh) M, such that $|\mathrm{M} - S| < \varepsilon$ for all points of M.
  - Piecewise linear approximation → error is O(h^2)
  - Error inversely proportional to #faces





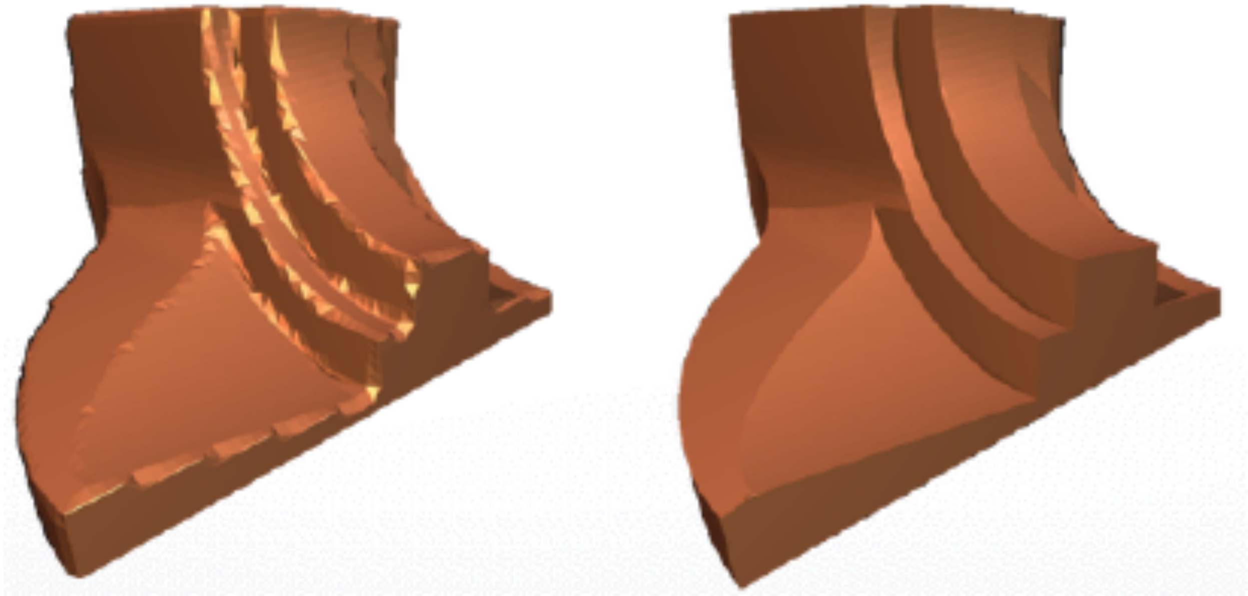
Polygonal meshes are a good compromise

- Piecewise linear approximation → error is $O(h^2)$
- Arbitrary topology surfaces



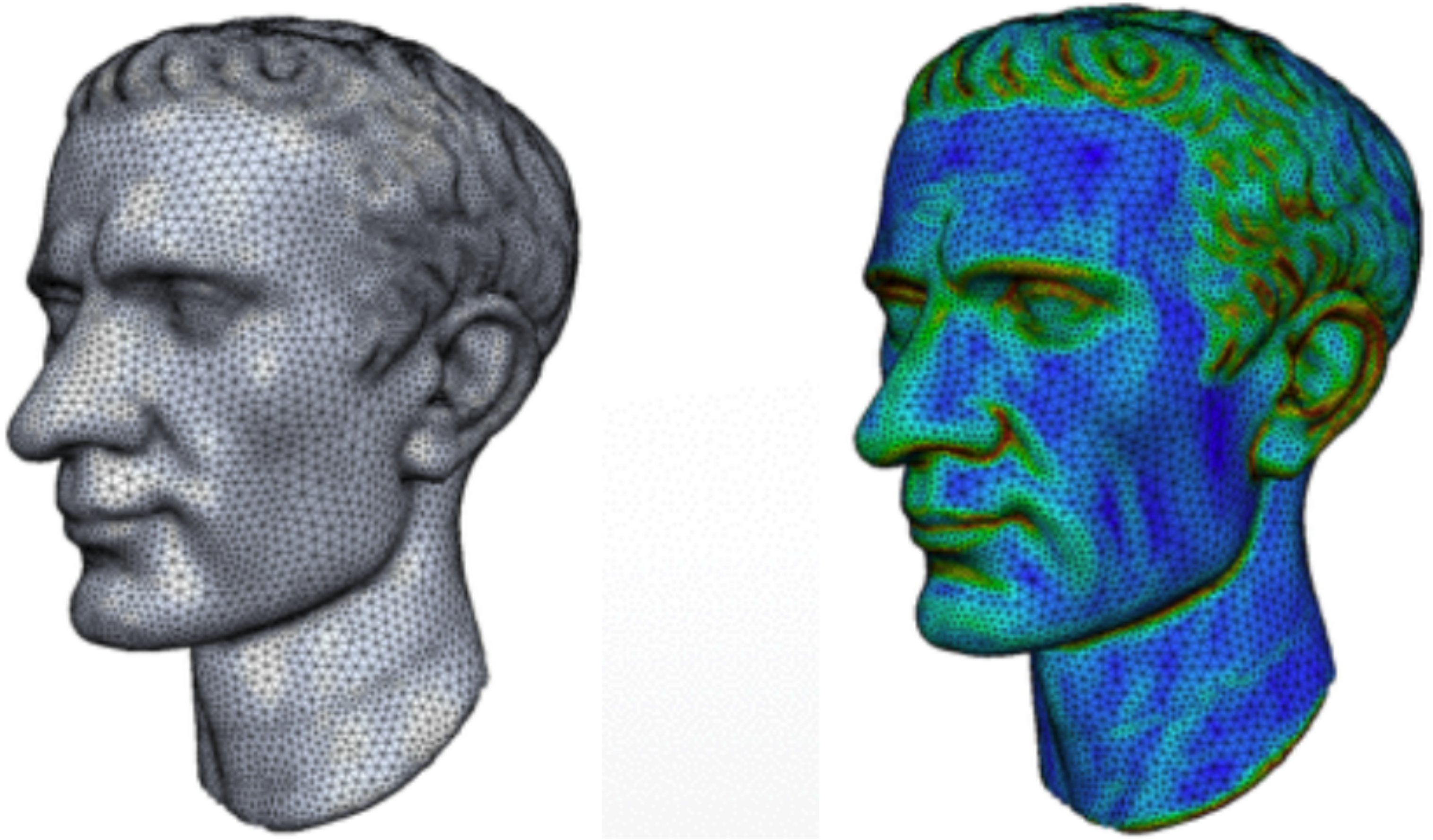
Polygonal meshes are a good compromise

- Piecewise linear approximation → error is $O(h^2)$
- Arbitrary topology surfaces
- Piecewise smooth surfaces



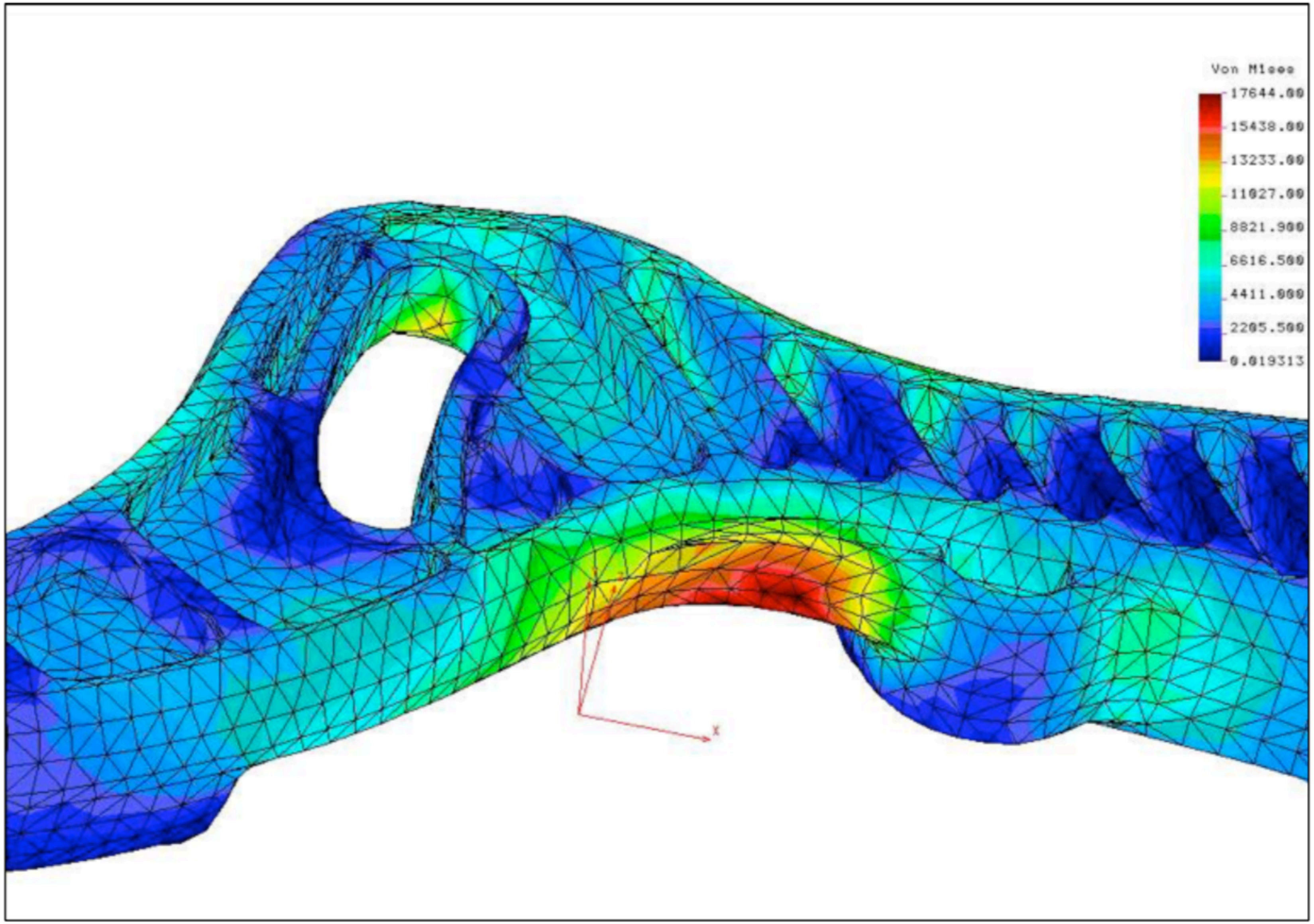
Polygonal meshes are a good compromise

- Piecewise linear approximation → error is $O(h^2)$
- Arbitrary topology surfaces
- Piecewise smooth surfaces
- Adaptive sampling



Polygonal meshes are a good compromise

- Piecewise linear approximation → error is $O(h^2)$
- Arbitrary topology surfaces
- Piecewise smooth surfaces
- Adaptive sampling
- Efficient GPU-based rendering/processing



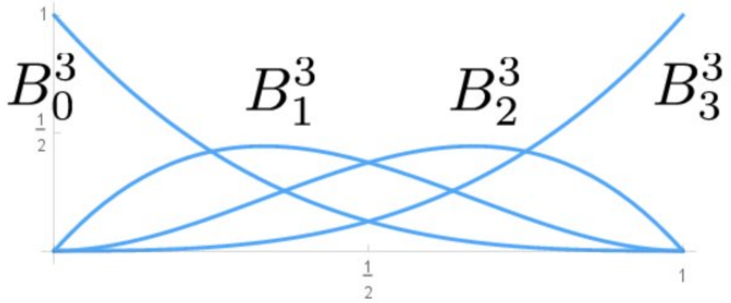
PATRIOT Engineering

finite element analysis

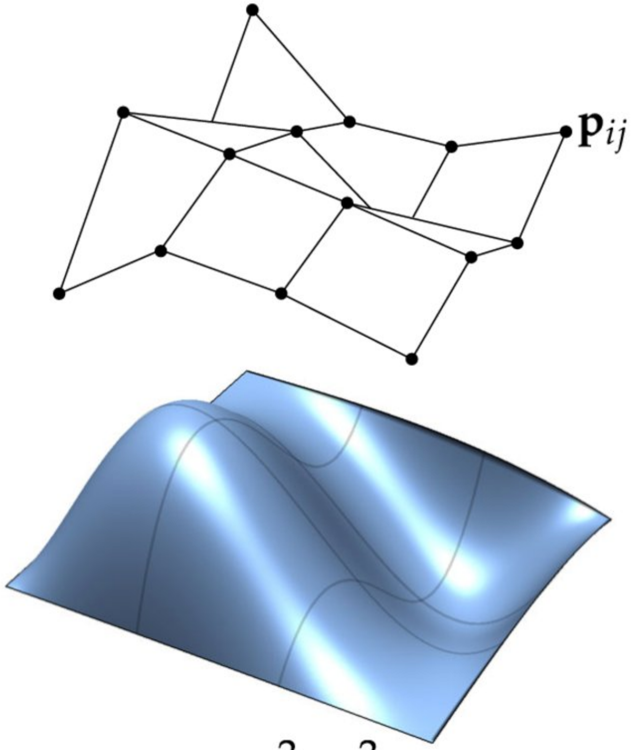
Spline & NURBS

- Extract analytical rep.
- Support interactive shape editing
- Compact rep.

- Major modeling techniques in CAD



$$B^3_{i,j}(u,v) := B^3_i(u)B^3_j(v)$$



$$S(u,v) := \sum_{i=0}^{3}\sum_{j=0}^{3} B^3_{i,j}(u,v)\mathbf{p}_{ij}$$

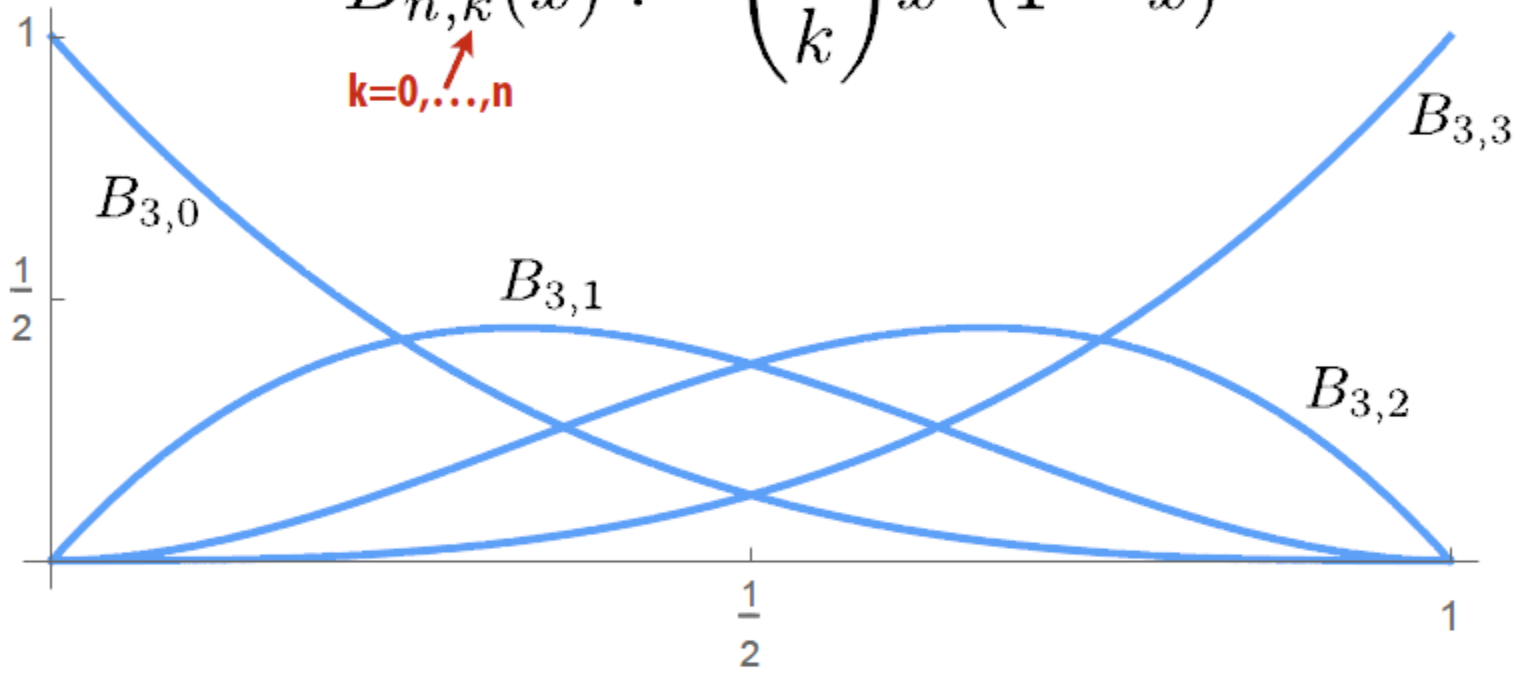
Bernstein Basis

- Why limit ourselves to just affine functions?
- More flexibility by using higher-order polynomials
- Instead of usual basis (1, x, x2, x3, ...), use Bernstein basis:

degree $0 \leq x \leq 1$ "n choose k"

$$B_{n,k}(x) := \binom{n}{k} x^k (1-x)^{n-k}$$

k=0,…,n



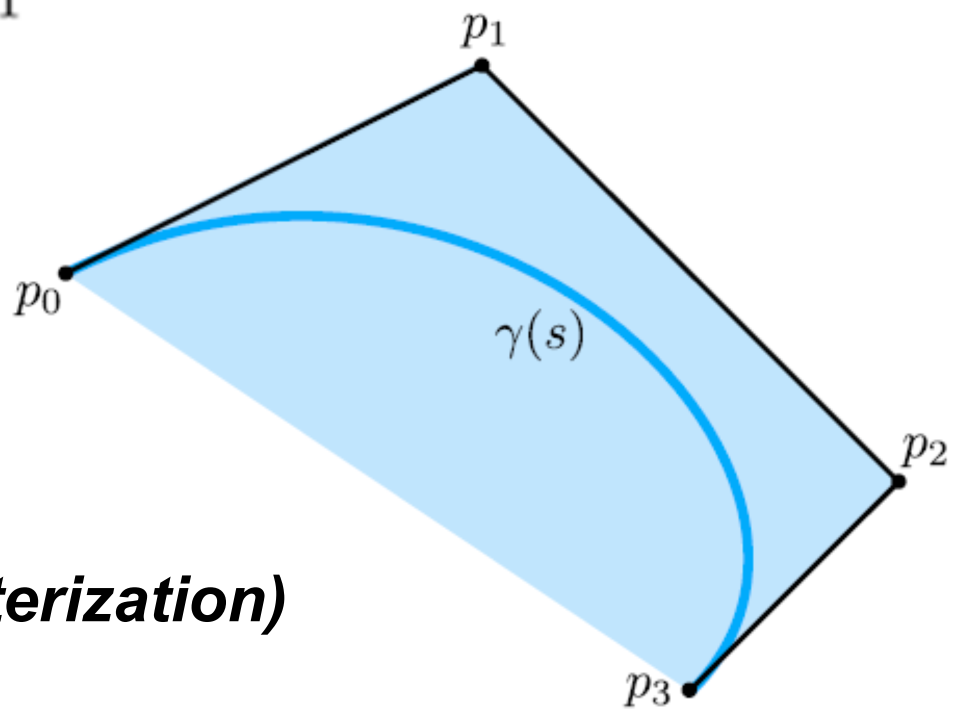
Bézier Curves (Explicit)

- A Bézier curve is a curve expressed in the Bernstein basis:

$$\gamma(s) := \sum_{k=0}^{n} B_{n,k}(s)p_k$$

control points

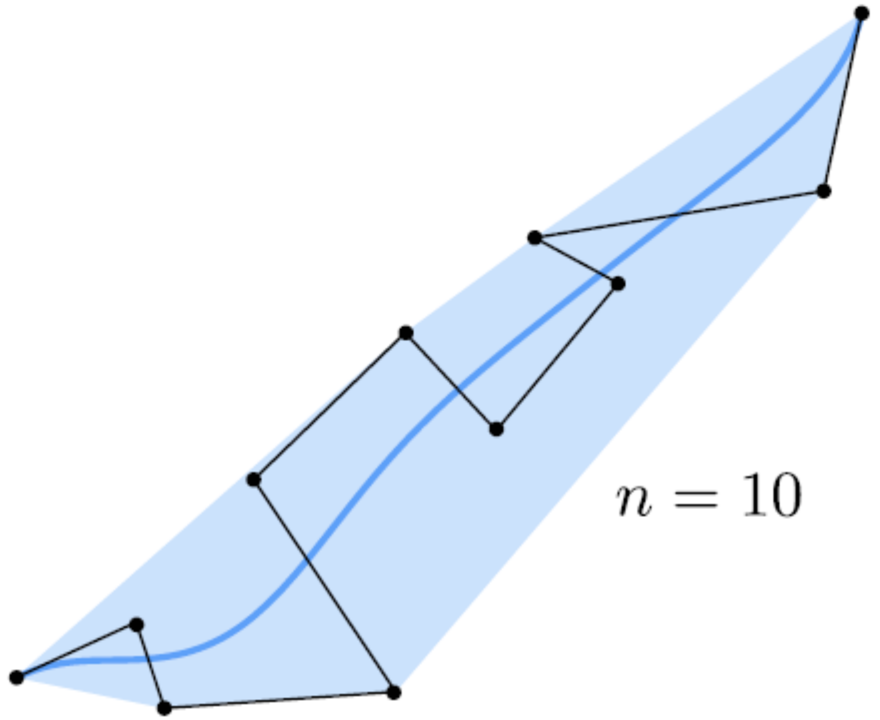
p_1



- For n=1, just get a line segment!
- For n=3, get “cubic Bézier”:
- Important features:
  1. interpolates endpoints
  2. tangent to end segments
  3. contained in convex hull *(nice for rasterization)*

Higher-order Bézier Curves?

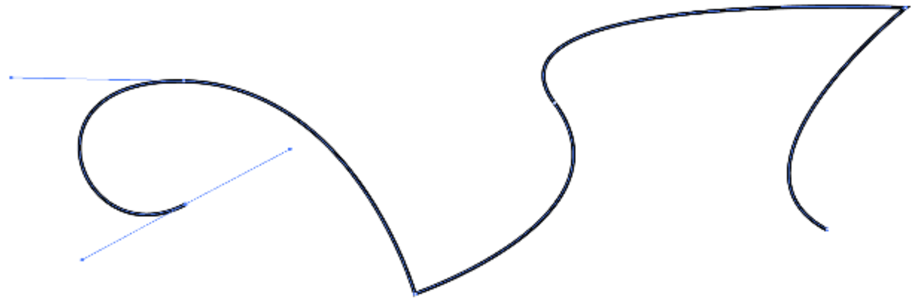
- **What if we want a more interesting curve?**
- **High-degree Bernstein polynomials don't interpolate well:**

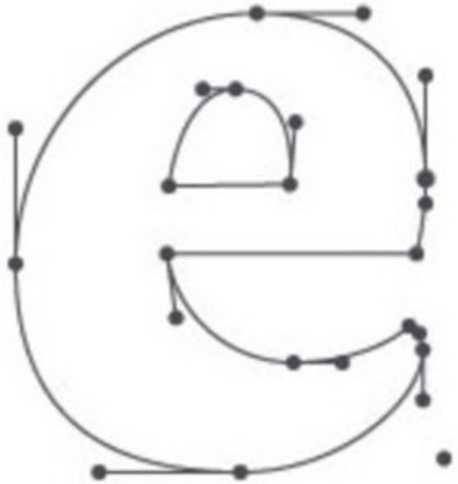


Very hard to control!

B-Spline Curves (Explicit)

- Instead, use many low-order Bézier curve (B-spline)
- Widely-used technique in software (Illustrator, Inkscape, etc.)





- Formally, piecewise Bézier curve:

B-spline

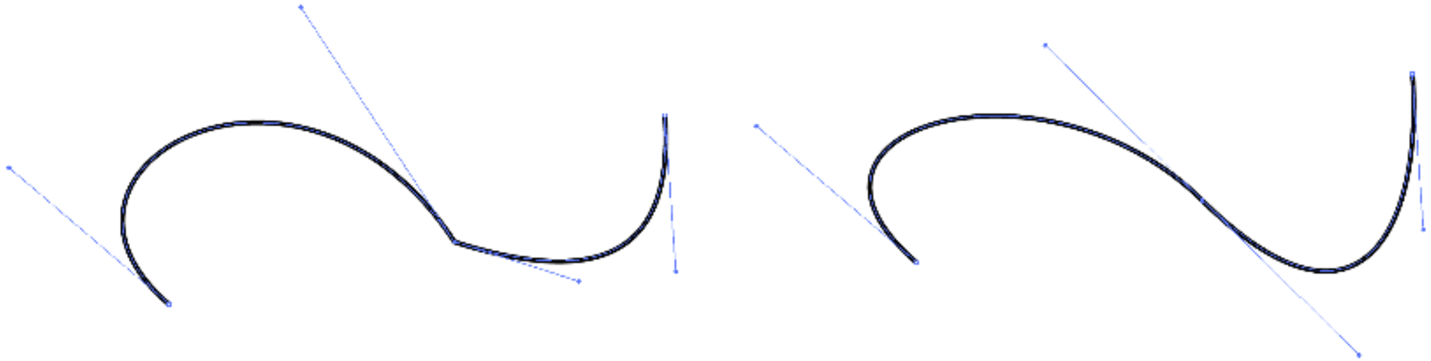
$$\gamma(u) := \gamma_i\left(\frac{u - u_i}{u_{i+1} - u_i}\right), \qquad u_i \leq u < u_{i+1}$$

Bézier

- Location of ui parameters are called "knots"

B-Splines — tangent continuity

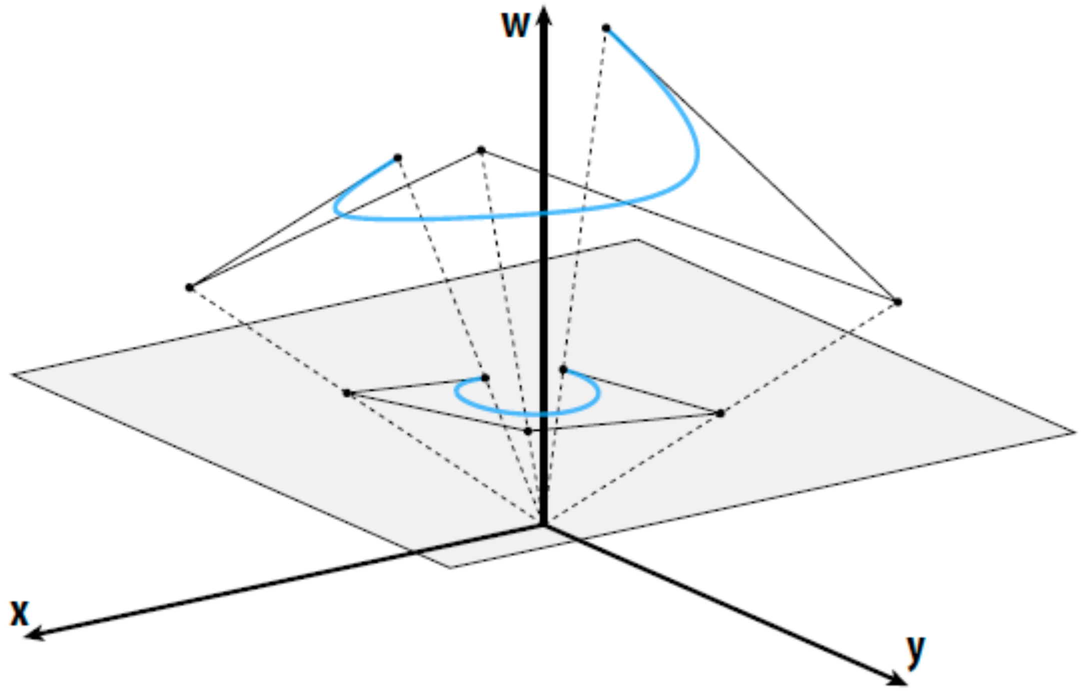
- To get “seamless” curves, want *tangents* to line up:



- Ok, but how?
- Each curve is cubic: u3p0 + 3u2(1-u)p1 + 3u(1-u)2p2 + (1-u)3p3
- Q: How many degrees of freedom in a single cubic Bézier
- Tangents are difference between first two & last two points
- Q: How many degrees of freedom per B-spline segment?
- Q: Could you do this with quadratic Bézier? Linear Bézier?

Rational B-Splines (Explicit)

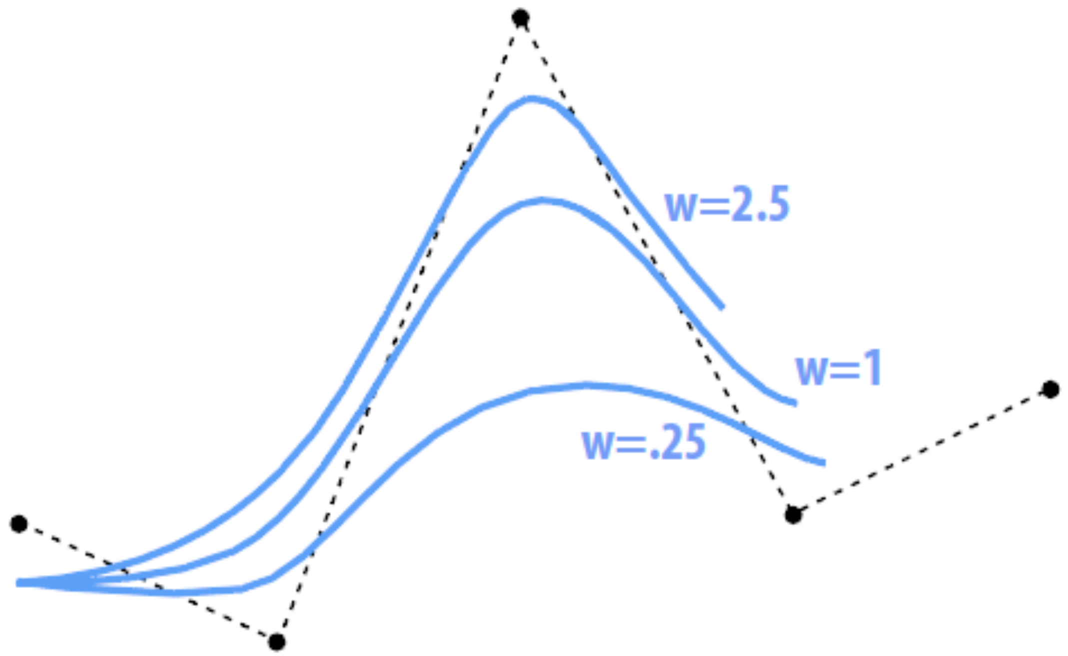
- B-Splines can't exactly represent *conics*—not even the circle!
- Solution: interpolate in homogeneous coordinates, then
- project back to the plane:



Result is called a *rational* B-spline.

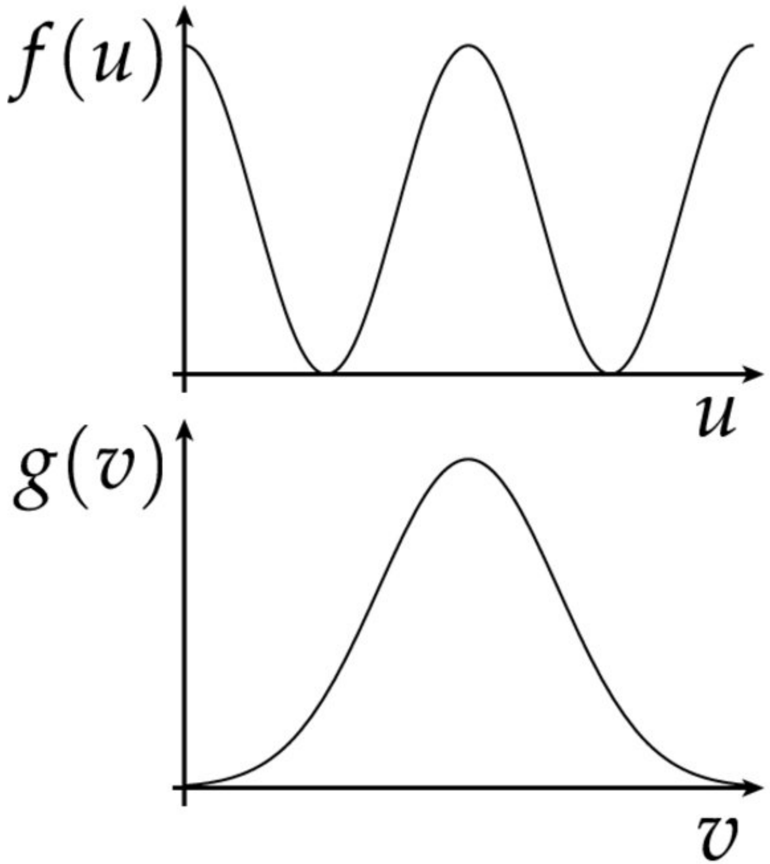
NURBS (Explicit)

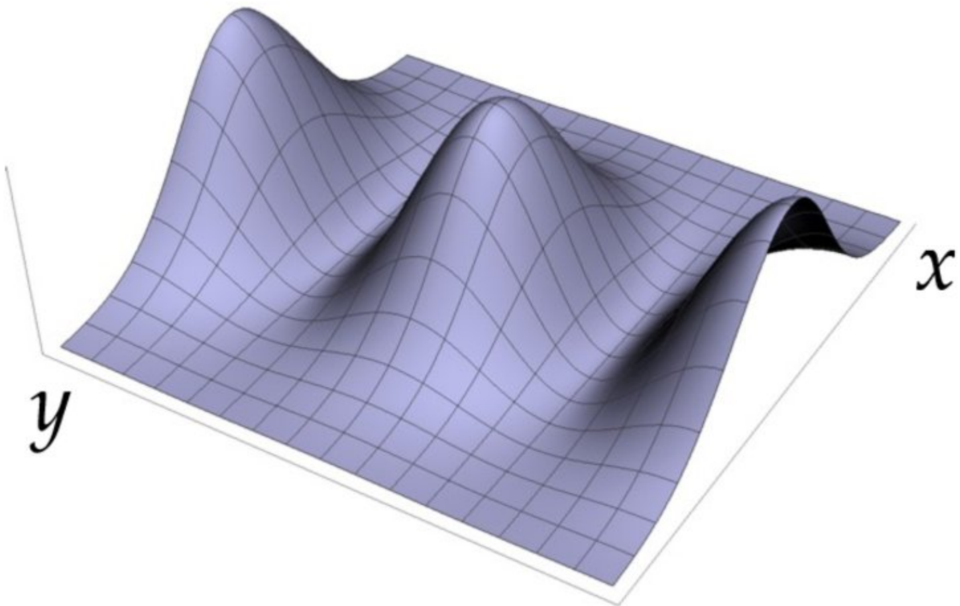
- (N)on-(U)niform (R)ational (B)-(S)pline
  - knots at arbitrary locations (non-uniform)
  - expressed in homogeneous coordinates (rational)
  - piecewise polynomial curve (B-Spline)
- Homogeneous coordinate w controls “strength” of a vertex:



Tensor product

- Use a pair of curves to get a surface
- Value at any point (u,v) given by product of a curve f at u and a curve g at v

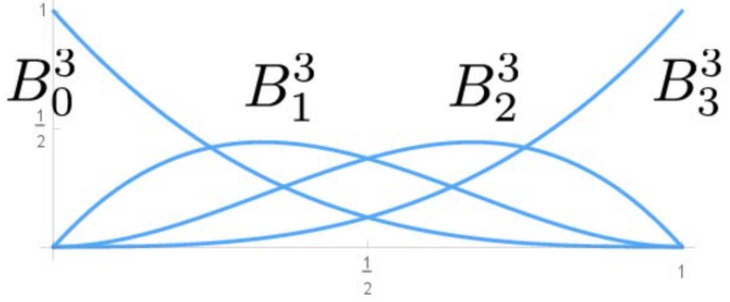




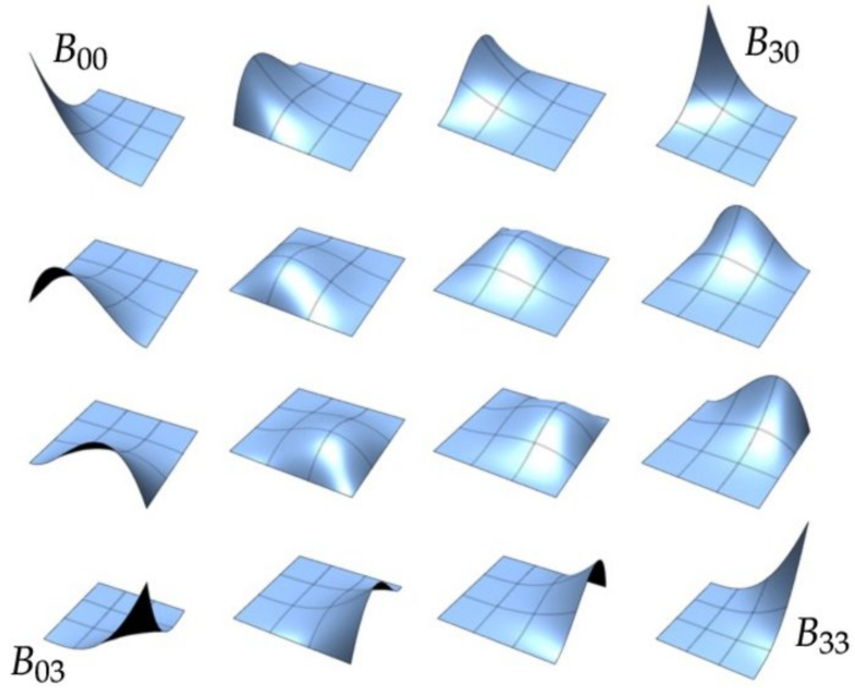
$$(f \otimes g)(u,v) := f(u)g(v)$$

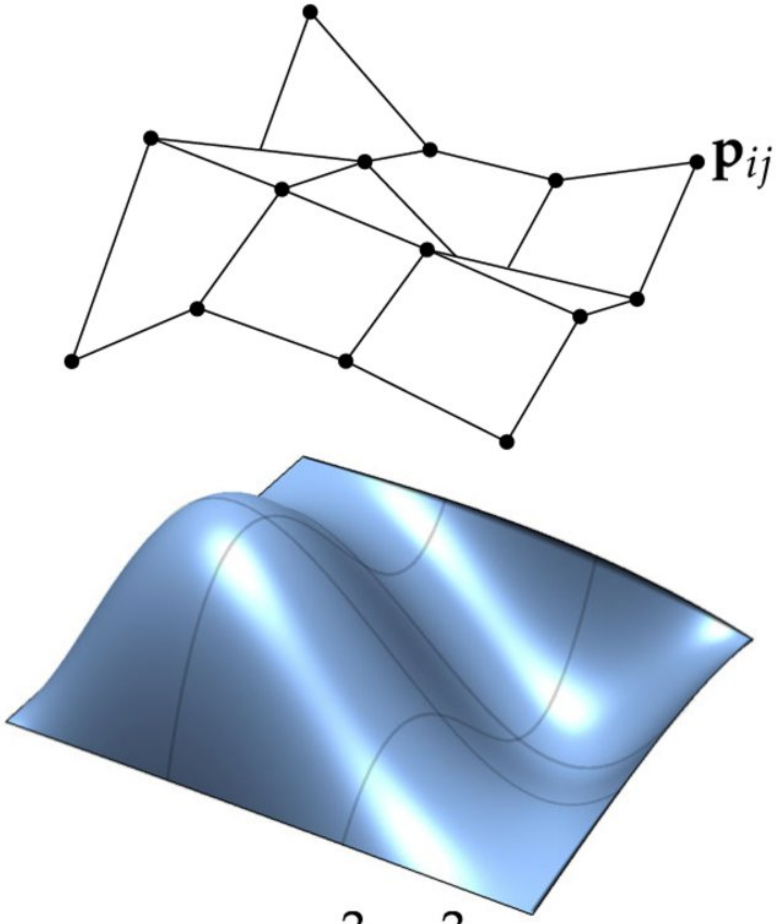
Bezier patches

- Bezier patch is a sum of (tensor) product of Bernstein bases.



$$B_{i,j}^3(u,v) := B_i^3(u)B_j^3(v)$$

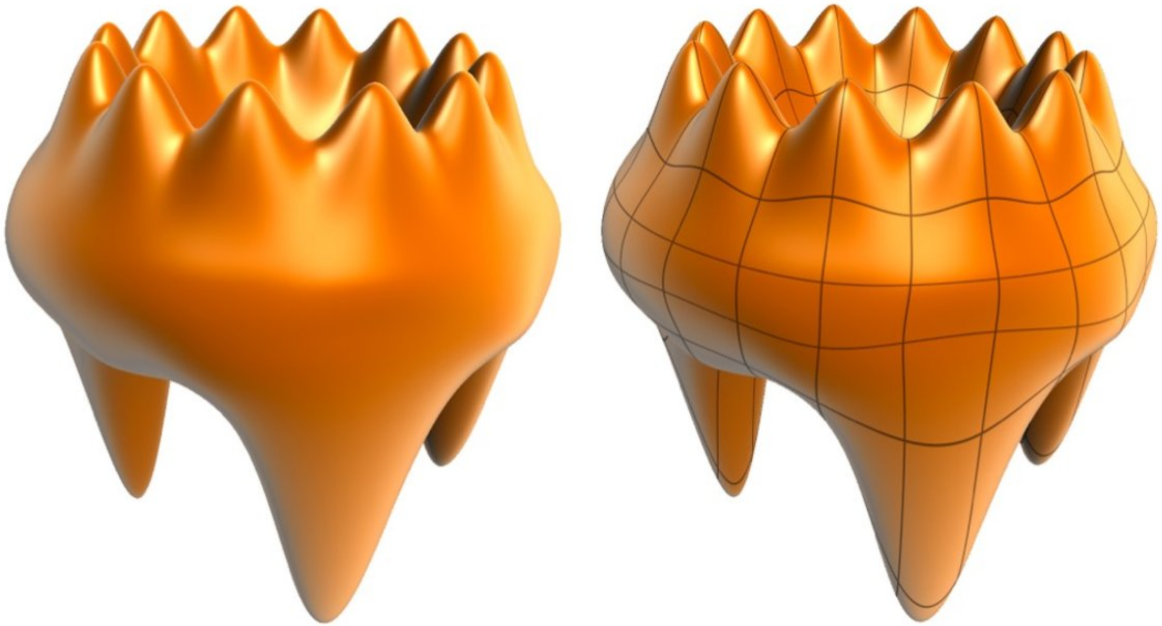




$$S(u,v) := \sum_{i=0}^{3}\sum_{j=0}^{3} B_{i,j}^3(u,v)\mathbf{p}_{ij}$$

Bezier surface

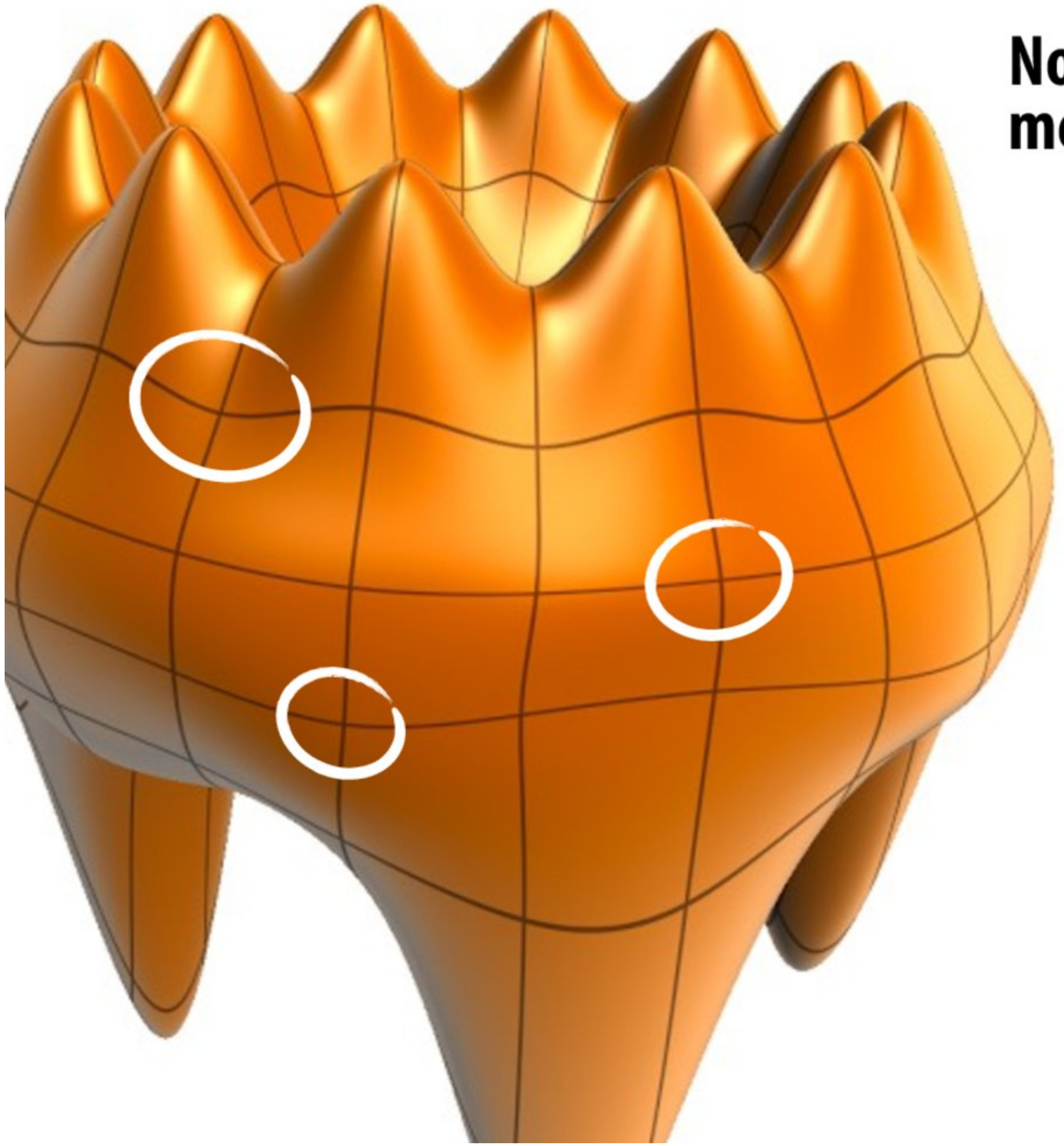
- Just as we connect Bezier curves, can connect Bezier patches to get a surface



- Very easy to draw: just dice each patch into a regular (u,v) grid!
- Q: Can we always get tangent continuity?

(Think: How many constraints? How many degrees of freedom?)

Bezier patches are too simple



Notice that exactly four patches meet around *every* vertex!

In practice, far too constrained.

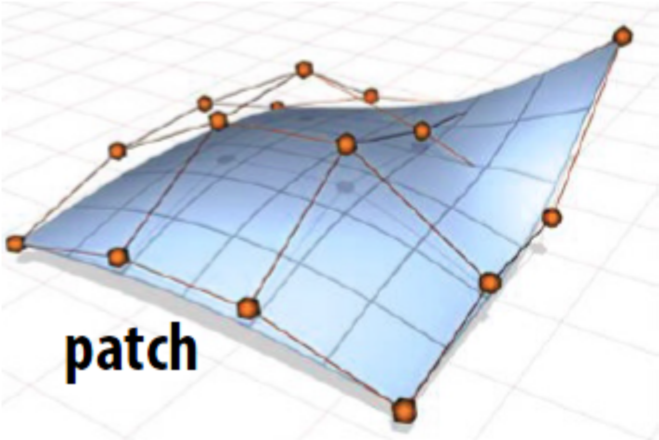
To make interesting shapes (with good continuity), we need patches that allow more interesting connectivity...

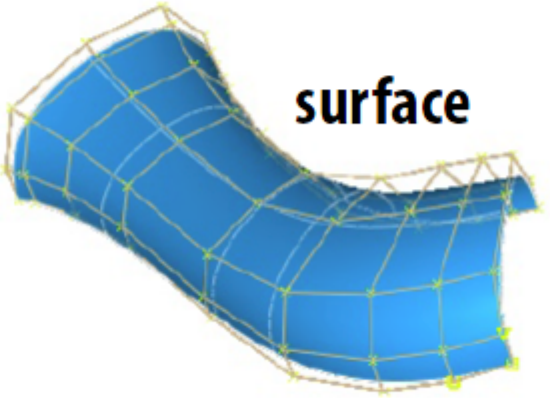
NURBS Surface (Explicit)

- Much more common than using NURBS for curves
- Use *tensor product* of scalar NURBS curves to get a patch:

$$S(u, v) := N_i(u) N_j(v) p_{ij}$$

- Multiple NURBS patches form a surface

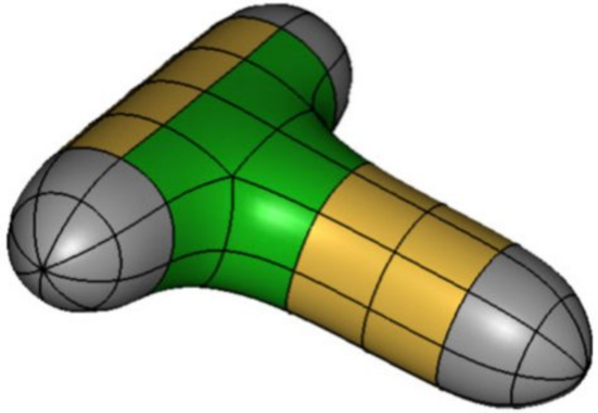




- Pros: easy to evaluate, exact conics, high degree of continuity
- Cons: Hard to piece together patches, hard to edit (many DOFs)

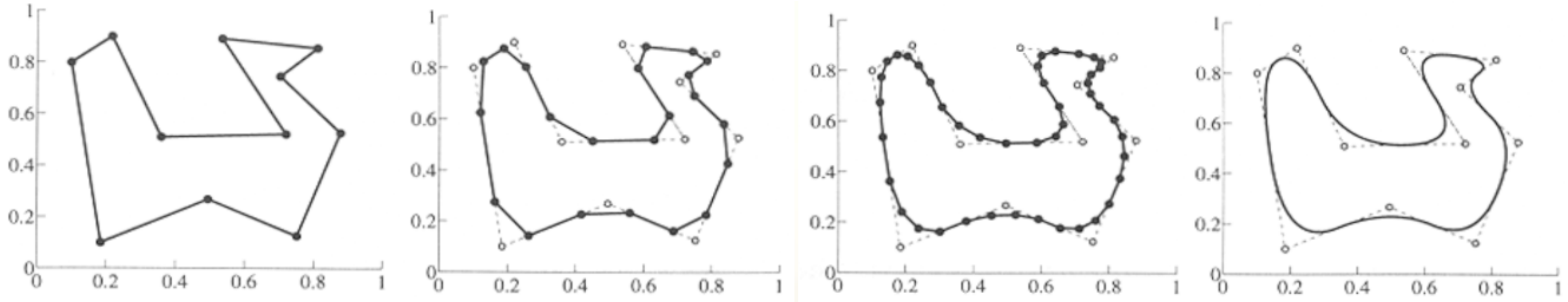
Spline patch schemes

- There are many alternatives
  - NURBS, Gregory, Pm, polar …
- Tradeoffs:
  - Degree of freedoms
  - Continuity
  - Difficulty of editing
  - Cost of evaluation
  - Generality
  - …
- As usual: pick the right tool for the job



Subdivision (Explicit)

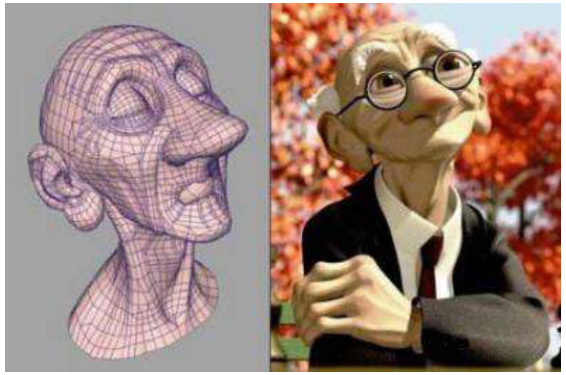
- Alternative starting point for curves: ***subdivision***
- **Start with control curve**
- **Insert new vertex at each edge midpoint**
- **Update vertex positions according to fixed rule**
- **For careful choice of averaging rule, yields smooth curve**
  - **Some subdivision schemes correspond to well-known spline schemes!**



Subdivision Surfaces (Explicit)

- Start with coarse polygon mesh (“control cage”)
- Subdivide each element
- Update vertices via local averaging
- Many possible rule:
  - Catmull-Clark (quads)
  - Loop (triangles)
  - ...

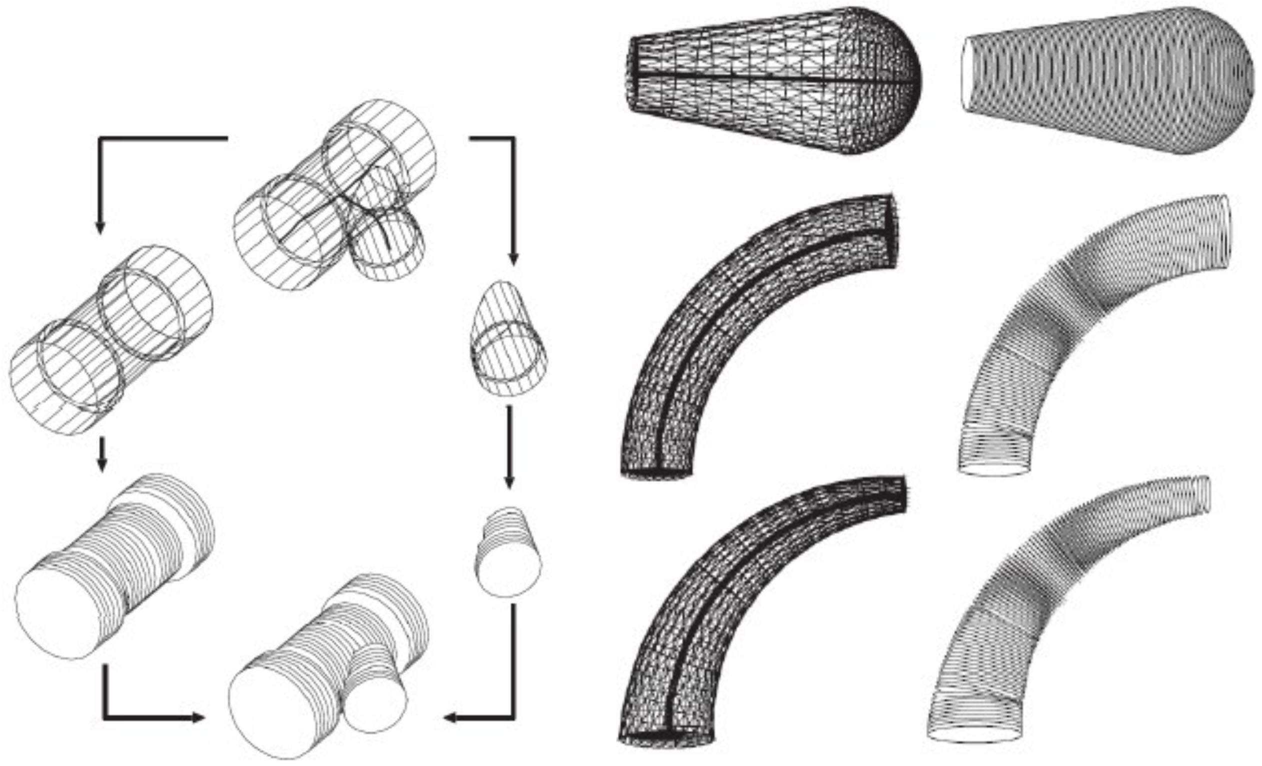
- Common issues:
  - interpolating or approximating?
  - continuity at vertices?
- Easier than splines for modeling



Other representations

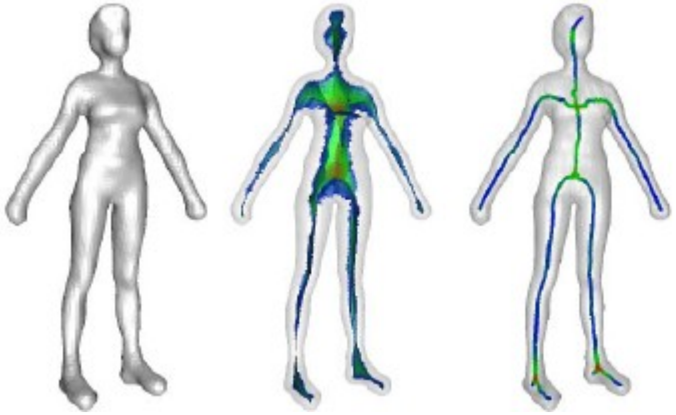
Generalized cylinder rep.

- A shape = {axis, a cross-section curve, a scaling function}
- Good for symmetric shapes with few local details and with clear skeletal structure
- Widely used in vision community for shape recognition, and shape recover

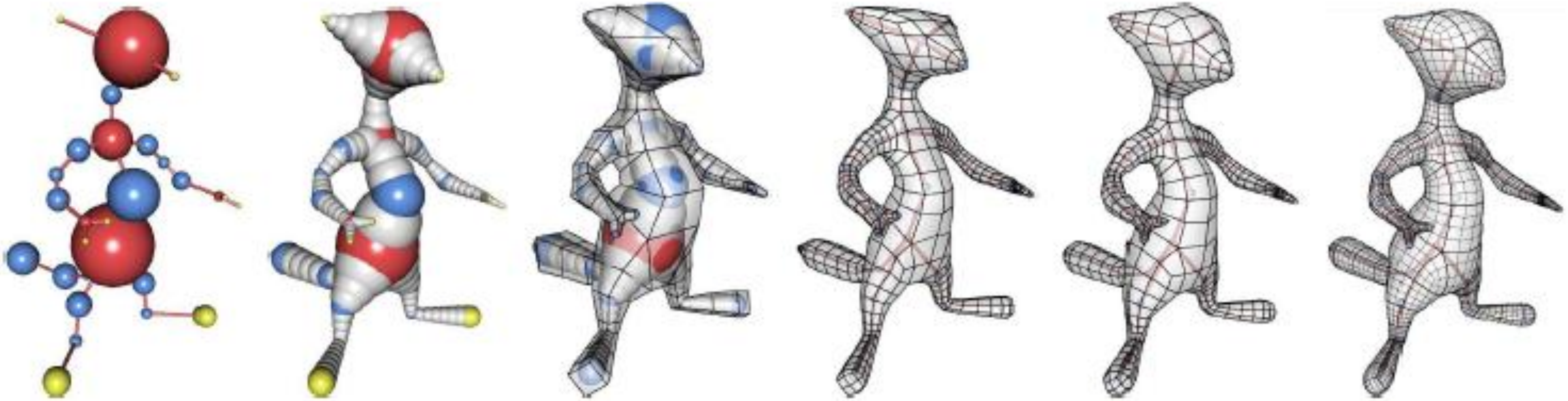


Skeleton Rep.

- A thin 1D or 2D representation of 2D/3D objects
- A (hierarchical) set of bones + attached skins
- Widely used in animation, matching, object recognition



pg10_B-Mesh: A Fast Modeling System for Base Meshes of 3D Articulated Shapes



Overview of our B-Mesh modeling approach. (a) Specifying the skeleton and key-balls at the nodes by users; (b) creating inbetween-balls (in gray) by interpolating the key-balls; (c) generating an initial mesh; (d) subdividing the mesh (c); (e) evolving the mesh (d); (f) obtaining the final mesh with more subdivision and evolution.

Gm13_Geometry Curves: A Compact Representation for 3D Shapes