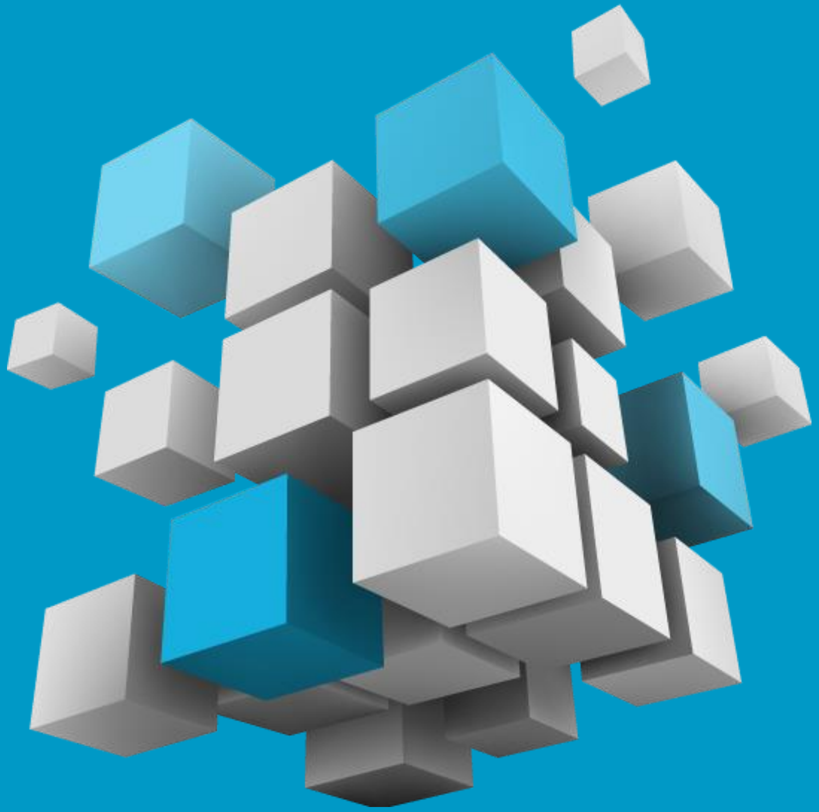
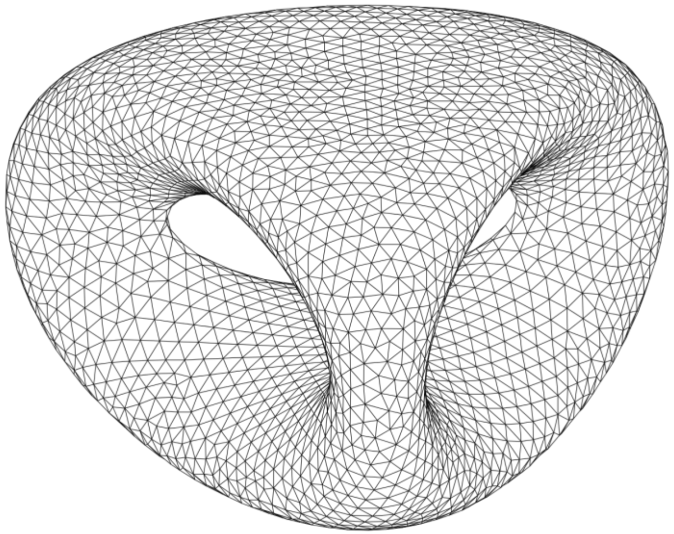
Modeling



What *surface representations* do you know about?

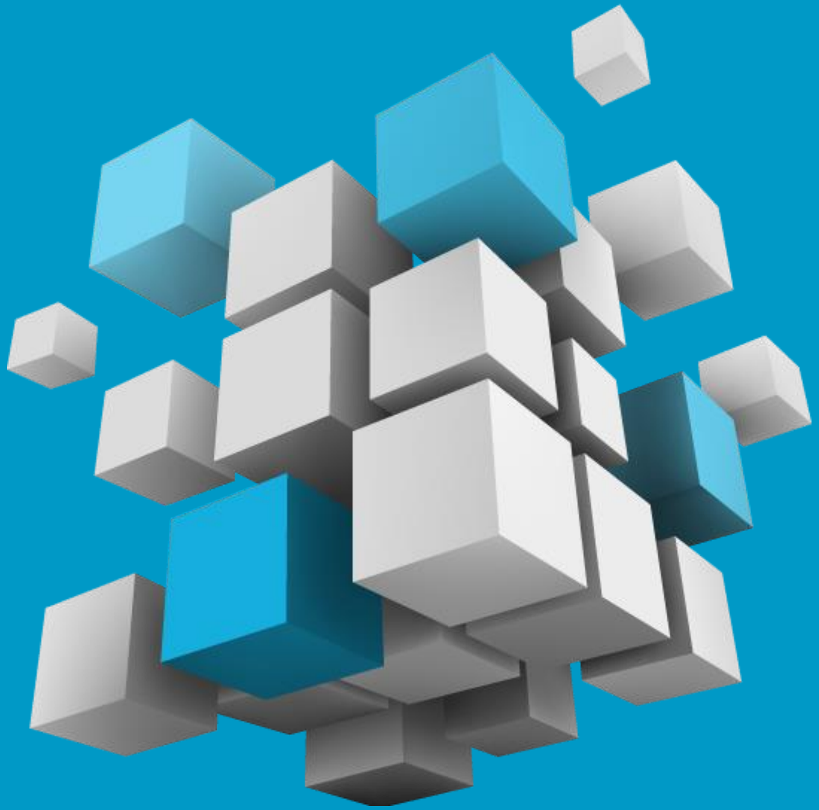
"**surfaces** are one way of representing objects. The other ways are wireframe (lines and curves) and solids."

[wikipedia]



Modeling

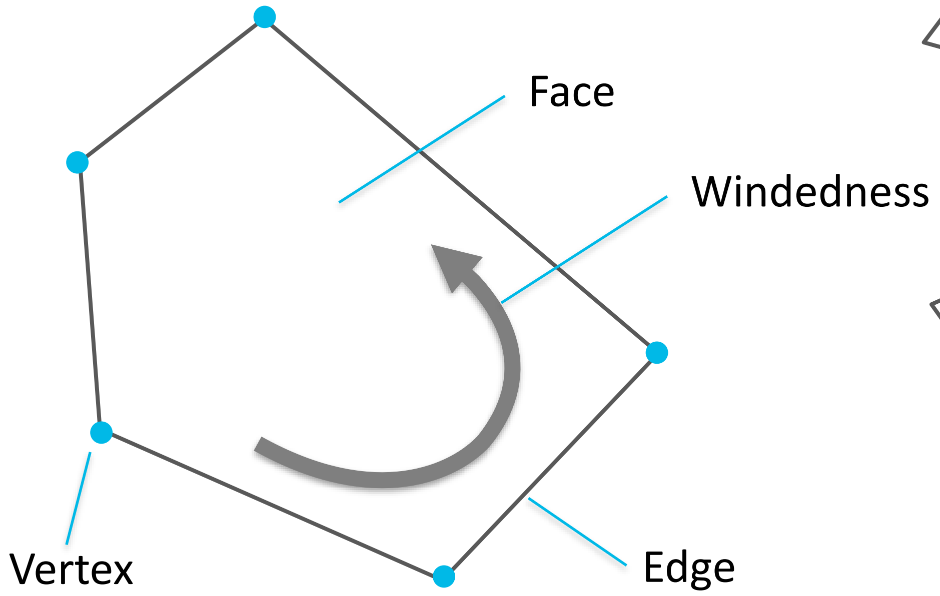
Polygon meshes

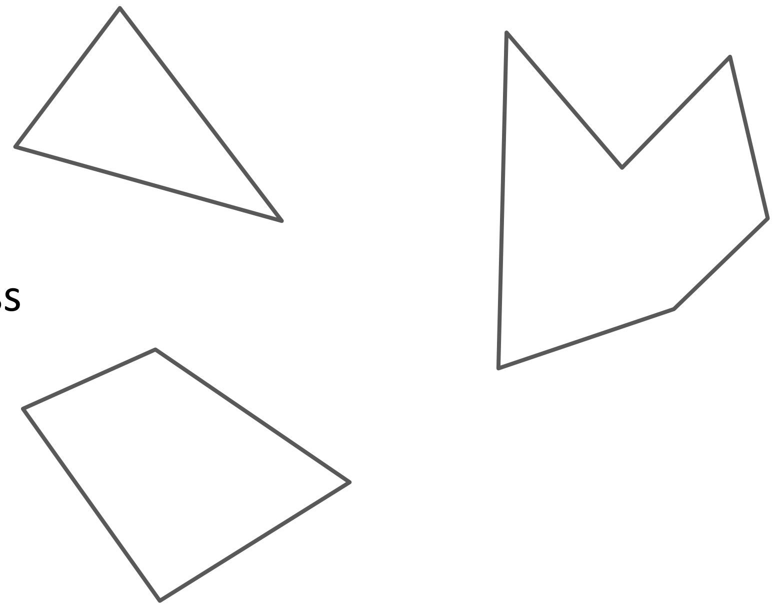


Polygons

Planar figures described by straight line segments

- Vertices
- Edges
- Face

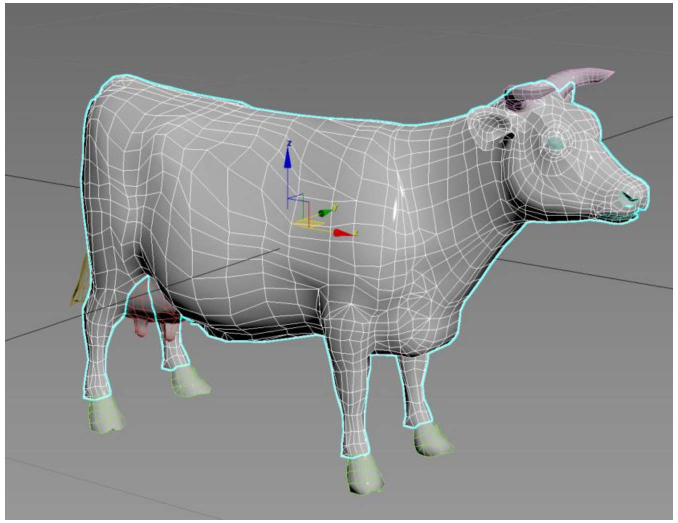




Windedness describes polygon orientation

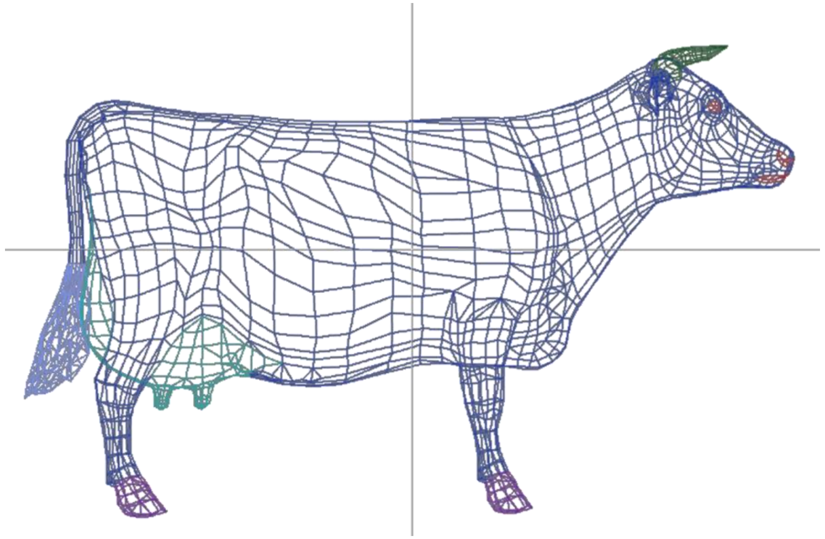
- Front side if **c**ounter-**c**lock**w**ise (ccw)

Polygon Meshes



- Consist of one or more polygons
  - Special case: triangle meshes
  - Quads preferred for modeling

- Typically stored in index-based lists
  - Vertices
  - Normals
  - Faces



- Rendering in OpenGL
  - Lines or triangles
  - Index list of vertices

Polygon Meshes – Normals

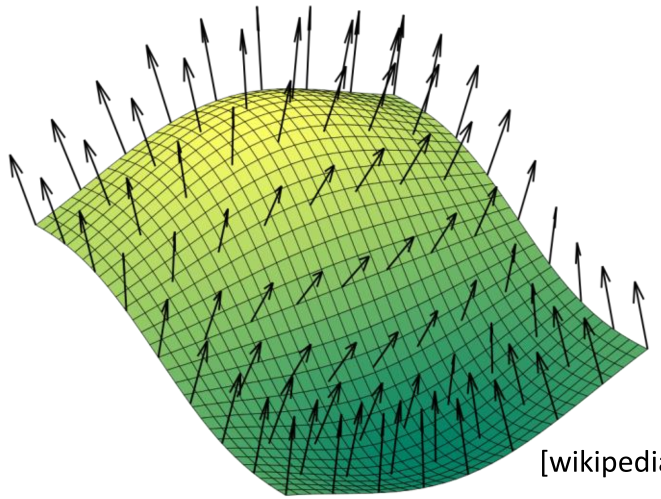
Affect surface appearance (shading)

Different types of normals

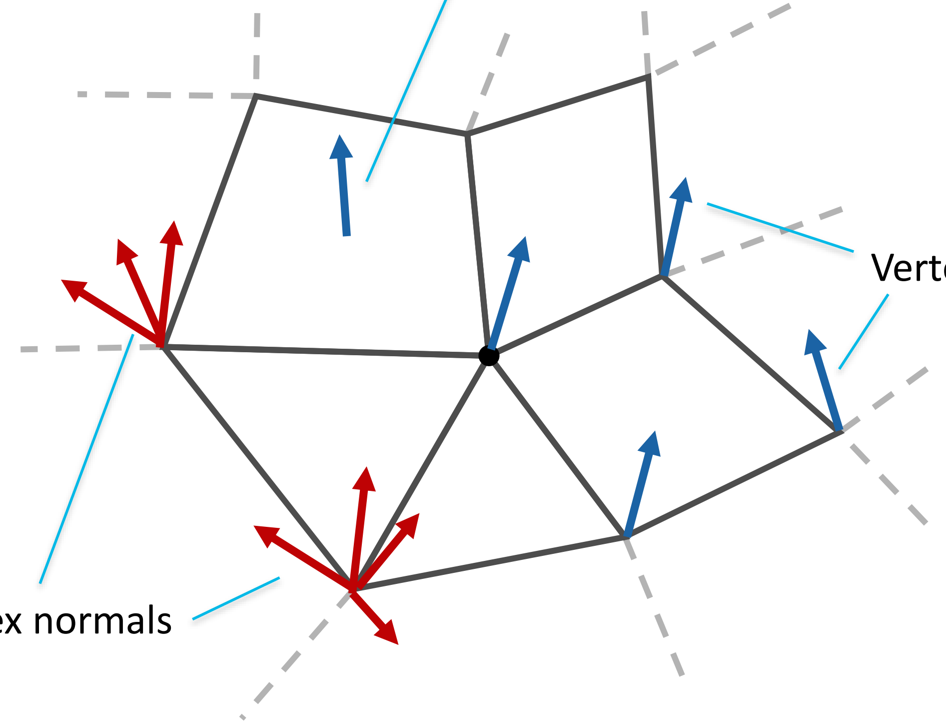
- Vertex normals
- Face normals

Vertex normals result in smooth surface

- Split vertex normals for hard edges
  (one for each face)



[wikipedia]



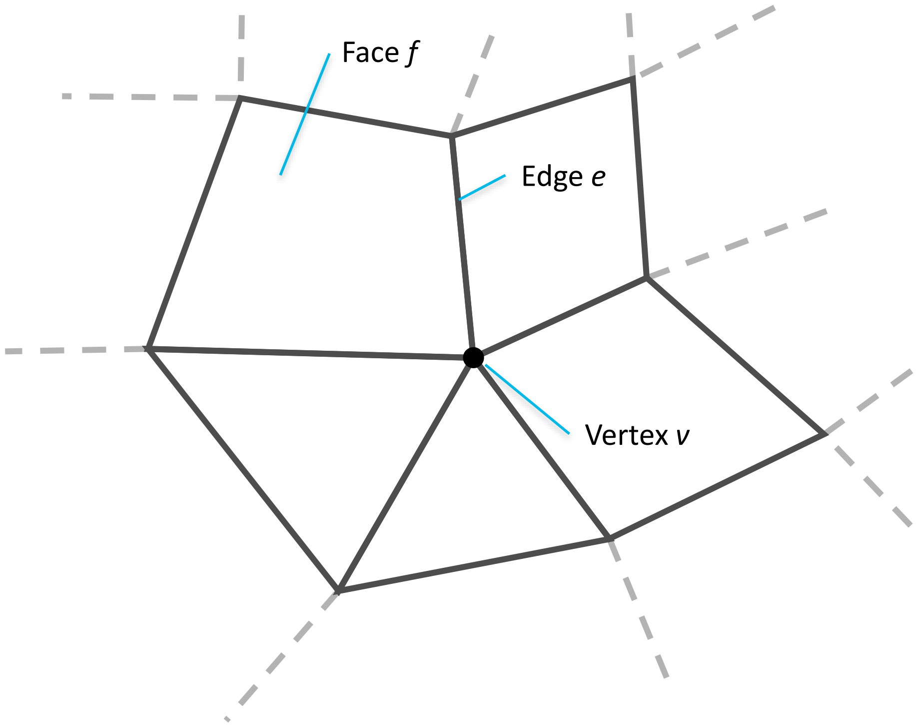
Polygon Meshes – Adjacency

Modeling and manipulating meshes requires information on adjacency

- Neighborhood information

Typical queries

- Adjacent faces of face *f*
- Faces using vertex *v*
- Edges connected to vertex *v*
- Faces belonging to edge *e*
- All edges of face *f*



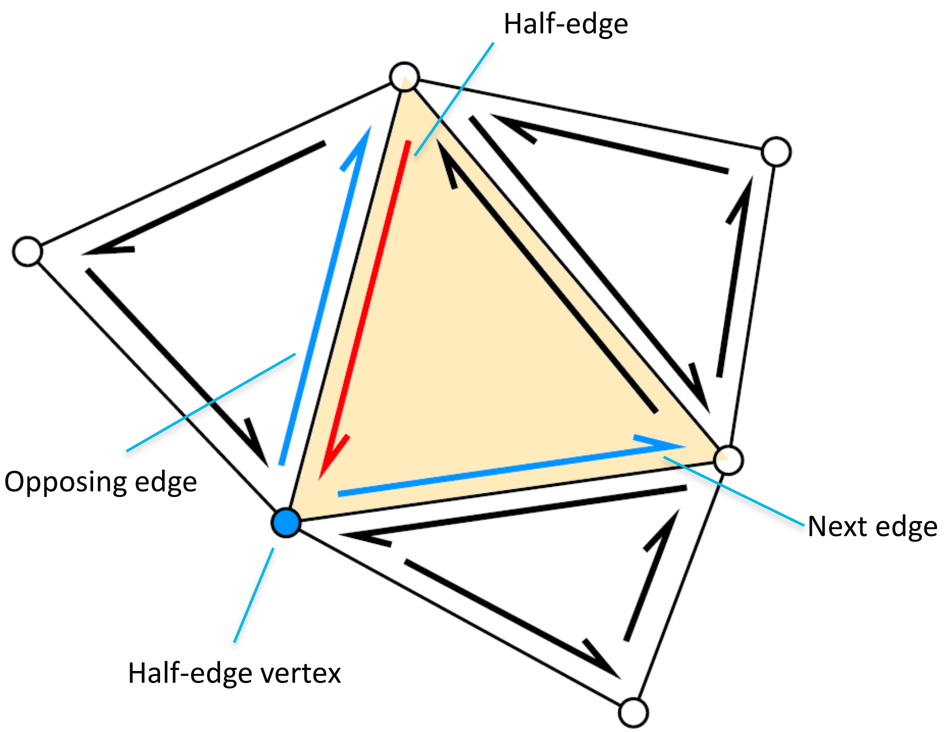
Half-edge Mesh Representation

Edges are split in half

- Each half belongs to one face
- Direction follows winding order (ccw)

Combines face and neighbor information

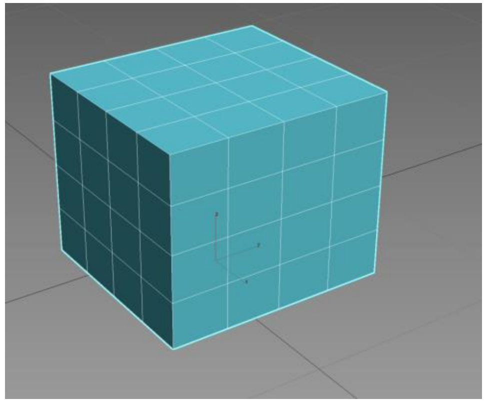
Sometimes previous edge is also stored

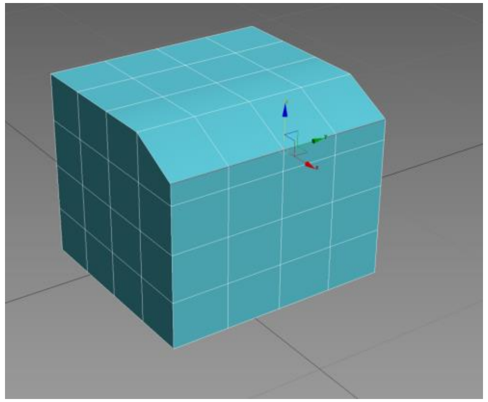



```
struct Halfedge;
struct Vertex {
    float x, y, z;
    Halfedge* edge;
};
struct Face {
    Halfedge* edge;
};
```

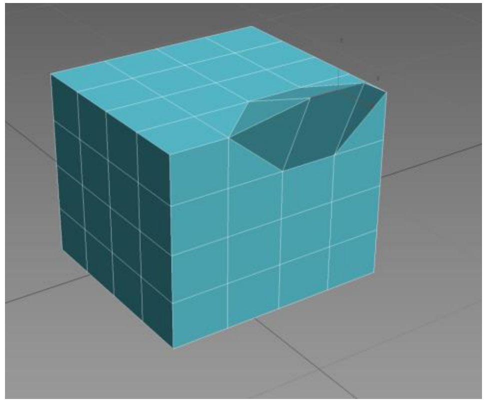
```
struct Halfedge {
    Vertex* vert;
    Halfedge* next;
    Halfedge* opposing;
    Face* left;
};
```

Typical Modeling Operations

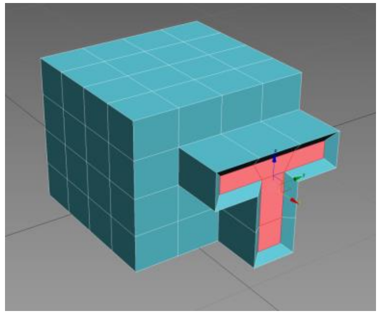




Edit edges

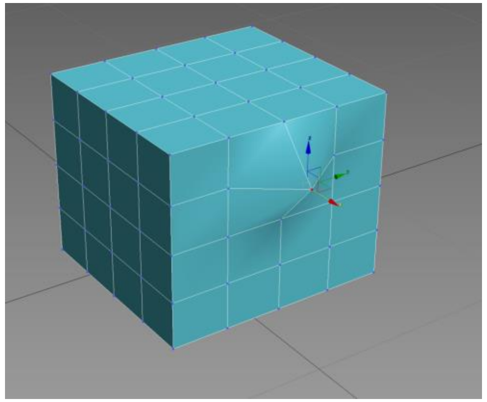


Extrude edges

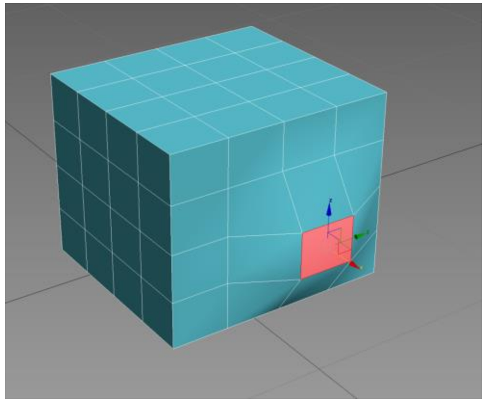


Insets

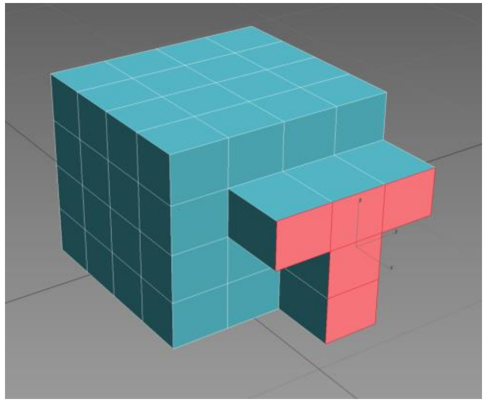
…



Edit vertices



Edit faces



Extrude faces

Subdivision Surfaces

Structured polygon meshes are easy to subdivide

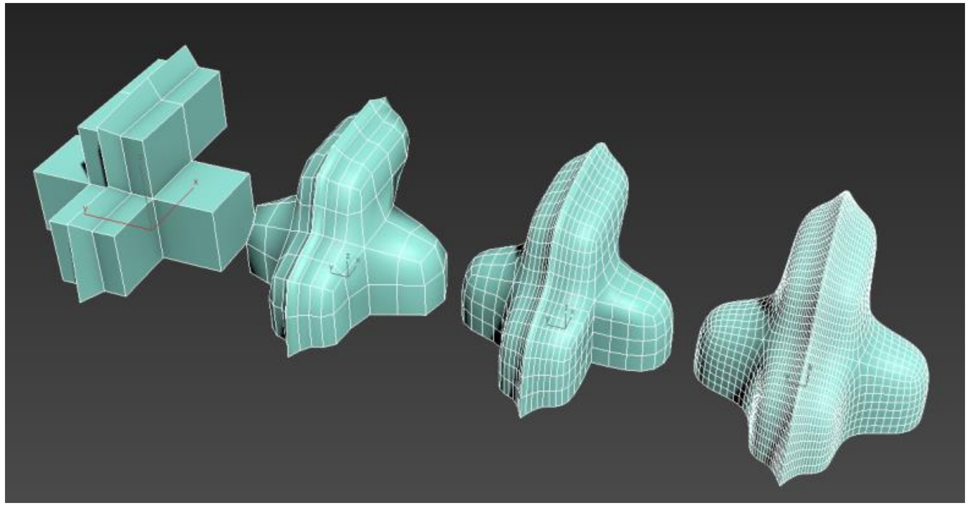
- Completely
- Or partially

Recursive subdivision of faces and edges

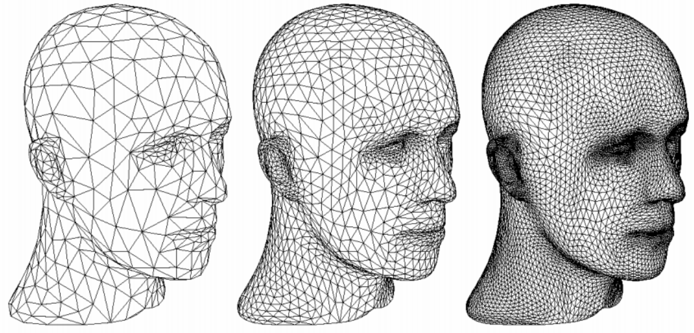
Results in very smooth meshes

Hierarchical subdivision surfaces (HSDS)

- Keep track of intermediate meshes



[autodesk.com]

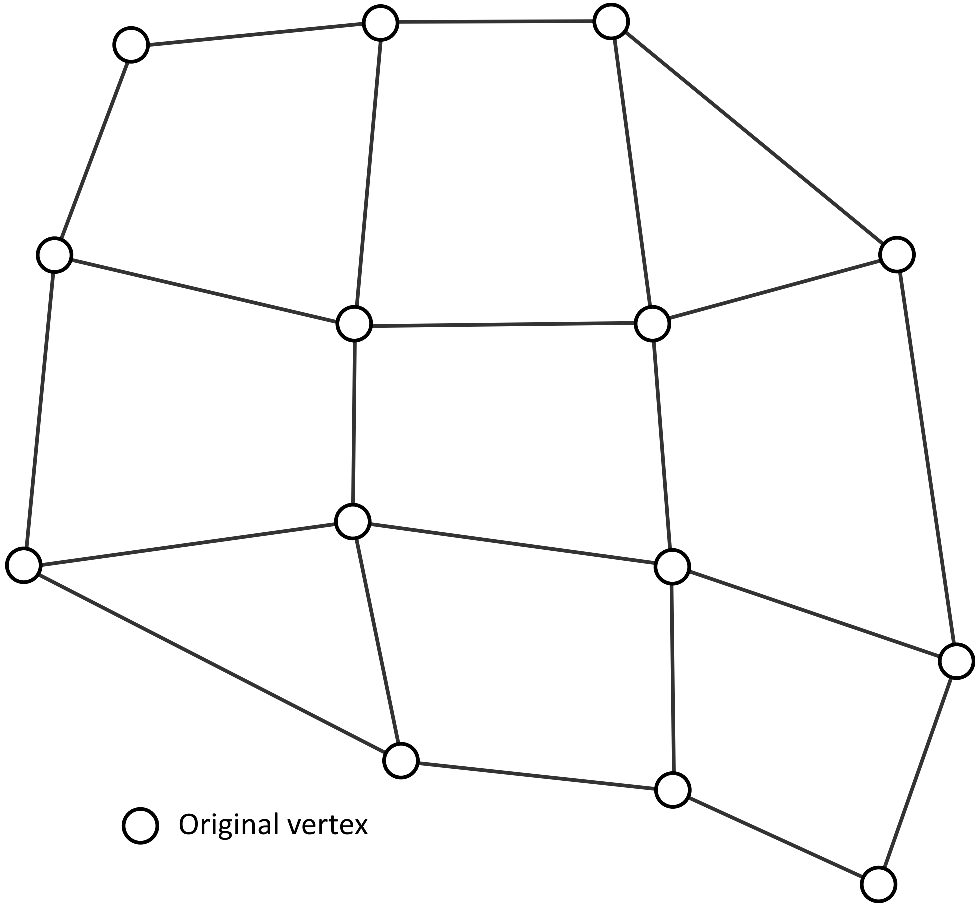


Catmull-Clark Subdivision

First subdivision method by
Edwin Catmull and Jim Clark, 1978

Algorithm

1. Create new *face vertex* in the center of the face
2. Create new edge vertex for each edge
3. Adjust original vertices (weighted average)
4. Add edges between new vertices
5. Create *n* new faces for a face with *n* vertices

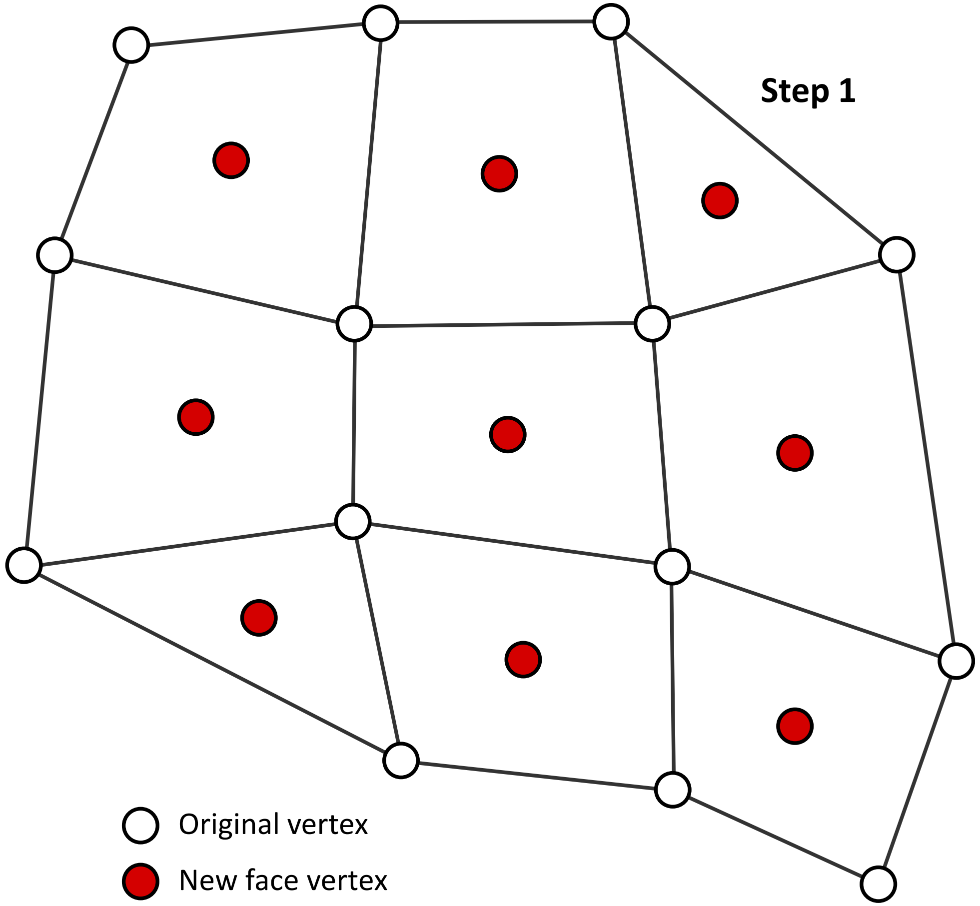


Catmull-Clark Subdivision

First subdivision method by
Edwin Catmull and Jim Clark, 1978

Algorithm

1. Create new *face vertex* in the center of the face
2. Create new edge vertex for each edge
3. Adjust original vertices (weighted average)
4. Add edges between new vertices
5. Create *n* new faces for a face with *n* vertices

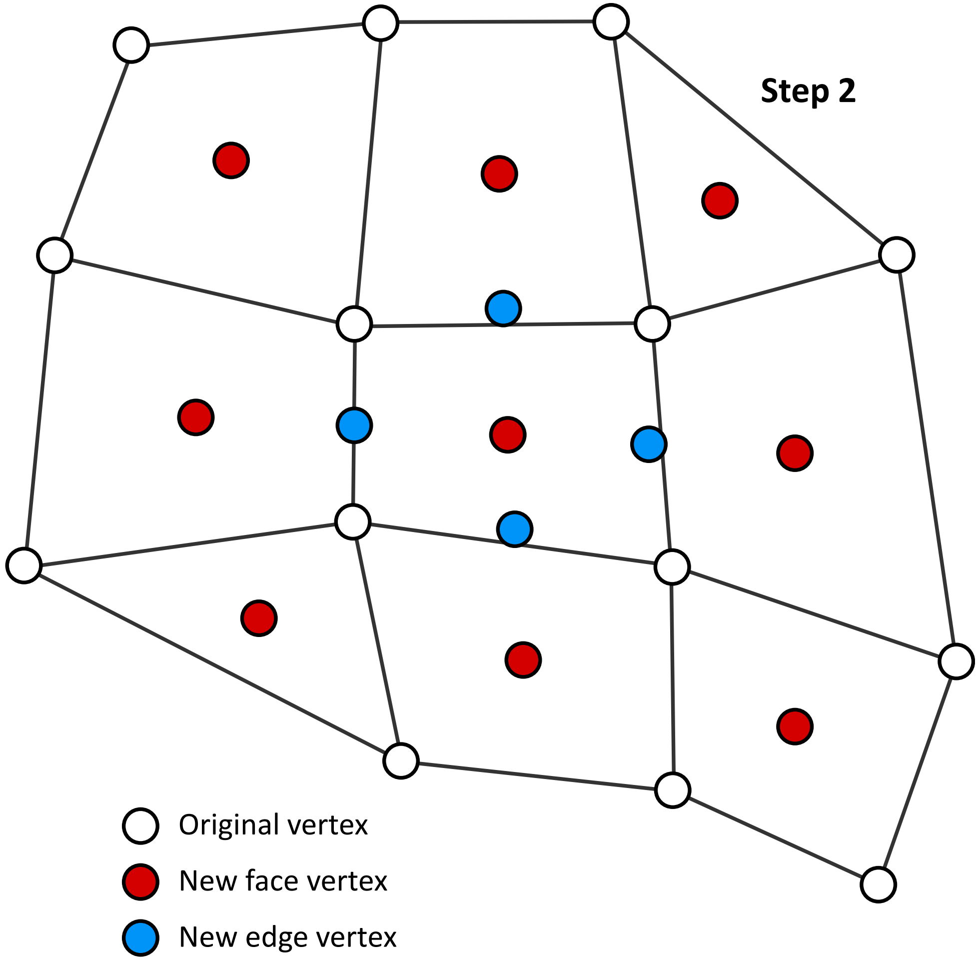


Catmull-Clark Subdivision

First subdivision method by
Edwin Catmull and Jim Clark, 1978

Algorithm

1. Create new *face vertex* in the center of the face
2. Create new edge vertex for each edge
3. Adjust original vertices (weighted average)
4. Add edges between new vertices
5. Create *n* new faces for a face with *n* vertices

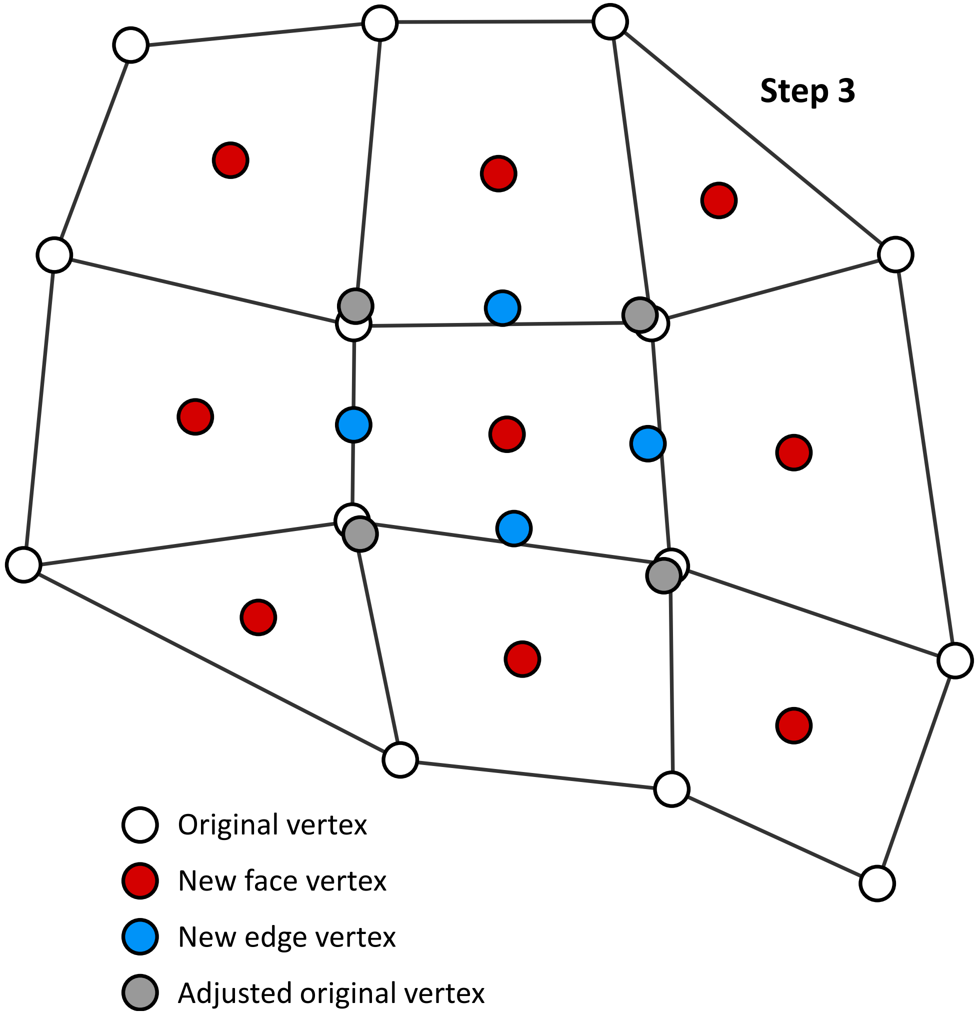


Catmull-Clark Subdivision

First subdivision method by
Edwin Catmull and Jim Clark, 1978

Algorithm

1. Create new *face vertex* in the center of the face
2. Create new edge vertex for each edge
3. Adjust original vertices (weighted average)
4. Add edges between new vertices
5. Create *n* new faces for a face with *n* vertices

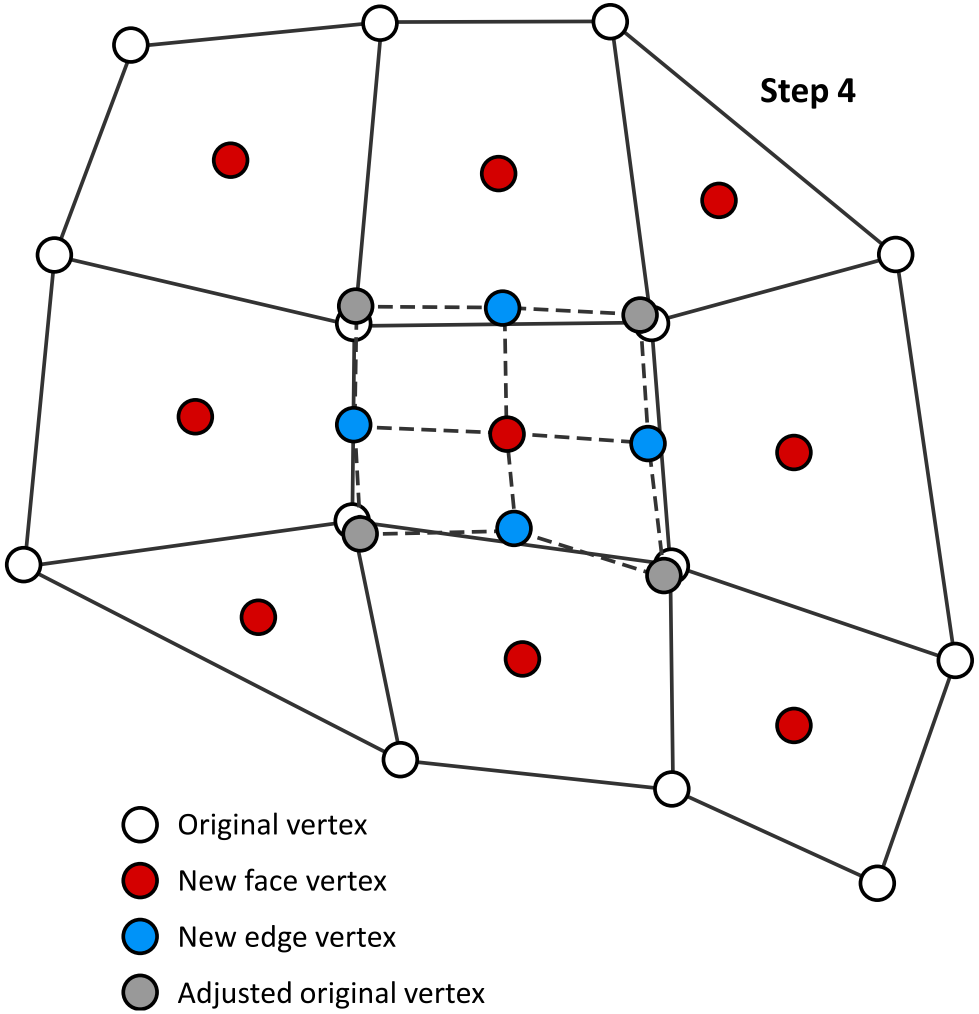


Catmull-Clark Subdivision

First subdivision method by
Edwin Catmull and Jim Clark, 1978

Algorithm

1. Create new *face vertex* in the center of the face
2. Create new edge vertex for each edge
3. Adjust original vertices (weighted average)
4. Add edges between new vertices
5. Create *n* new faces for a face with *n* vertices

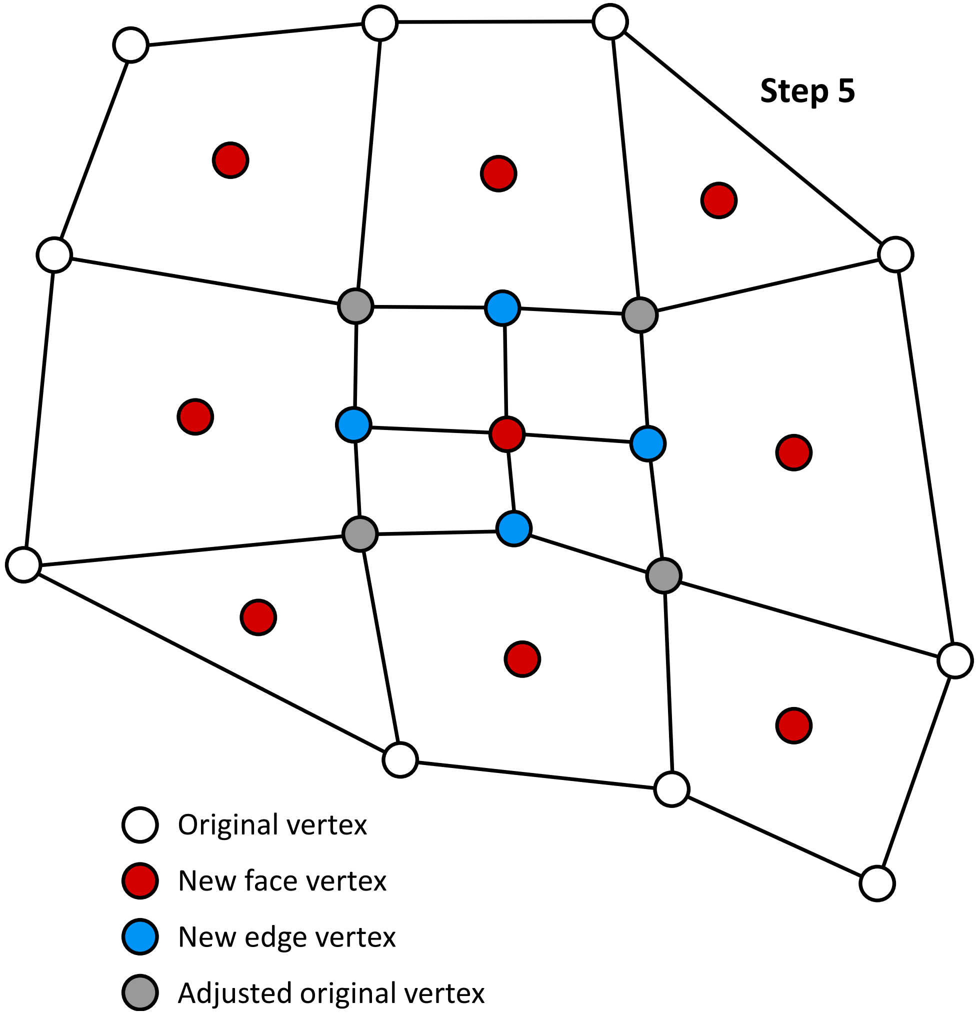


Catmull-Clark Subdivision

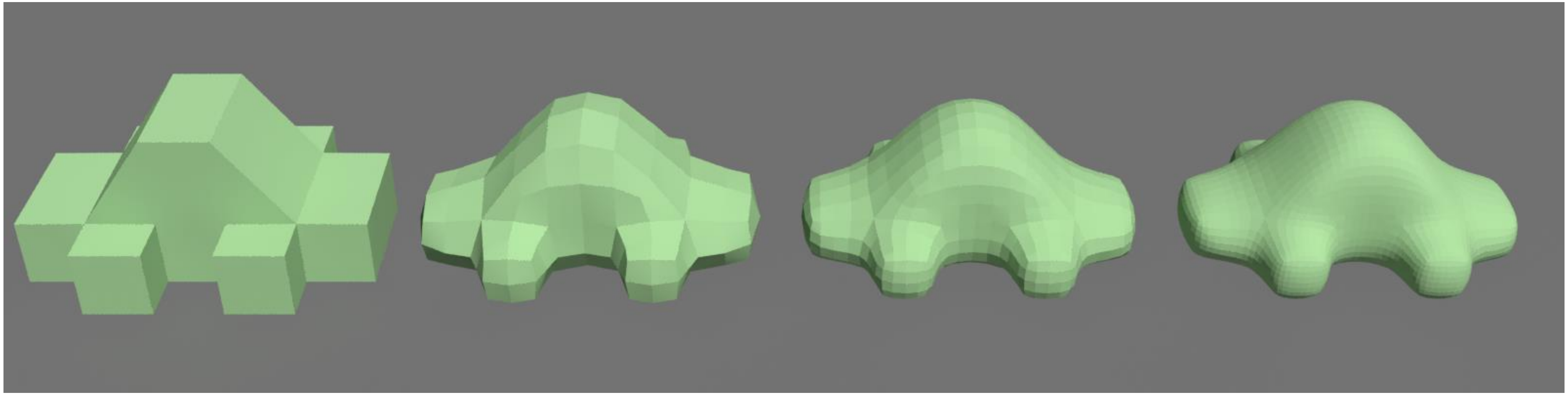
First subdivision method by
Edwin Catmull and Jim Clark, 1978

Algorithm

1. Create new *face vertex* in the center of the face
2. Create new edge vertex for each edge
3. Adjust original vertices (weighted average)
4. Add edges between new vertices
5. Create *n* new faces for a face with *n* vertices



Catmull-Clark Subdivision (cont.)

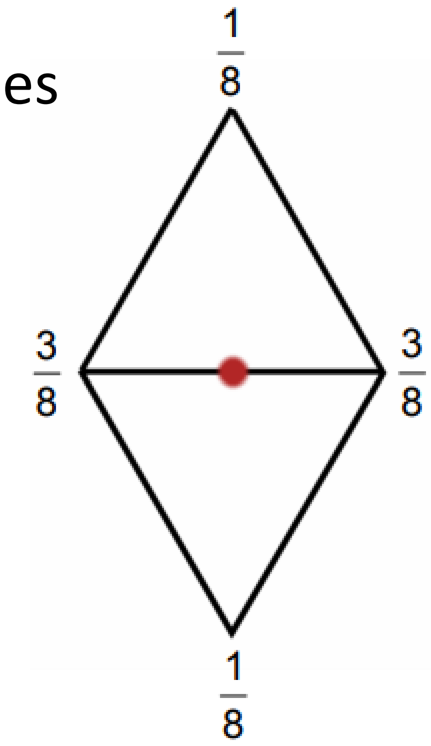


Original Mesh First Iteration Second Iteration Third Iteration

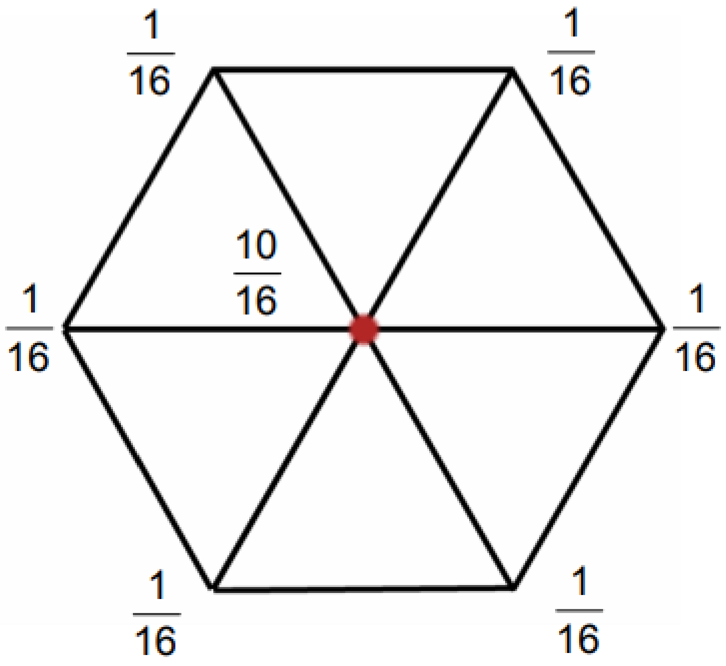
Subdivision Surfaces – Loop Subdivision

Applied to triangle meshes

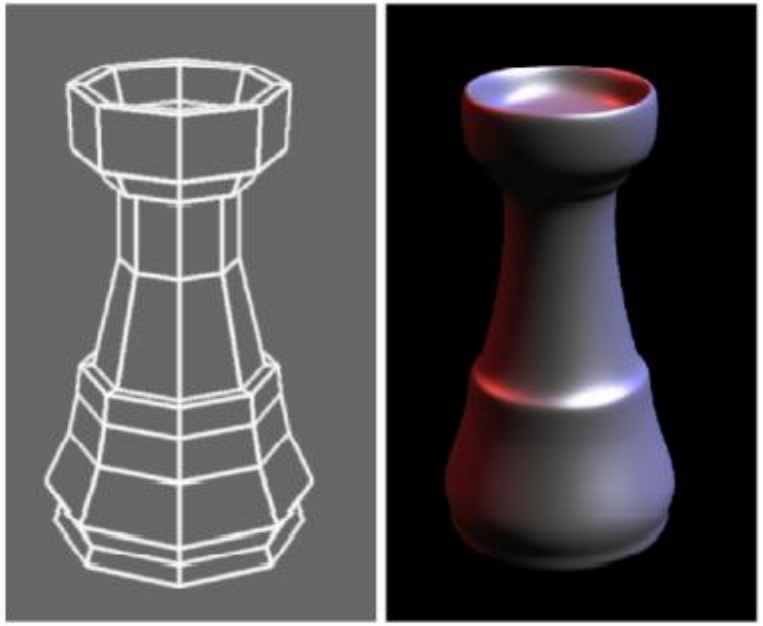
Rules



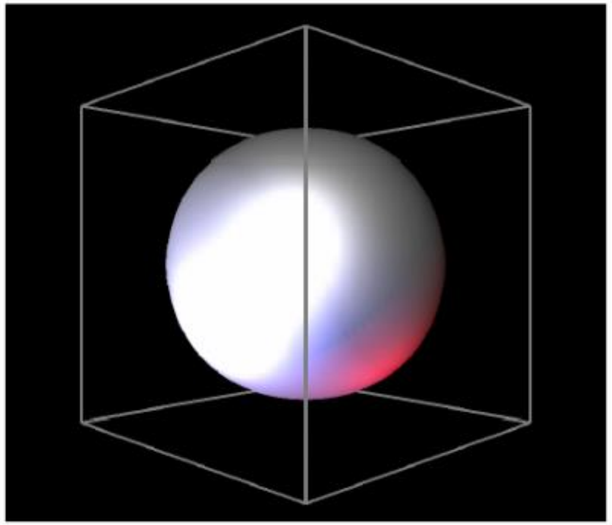
Add new point



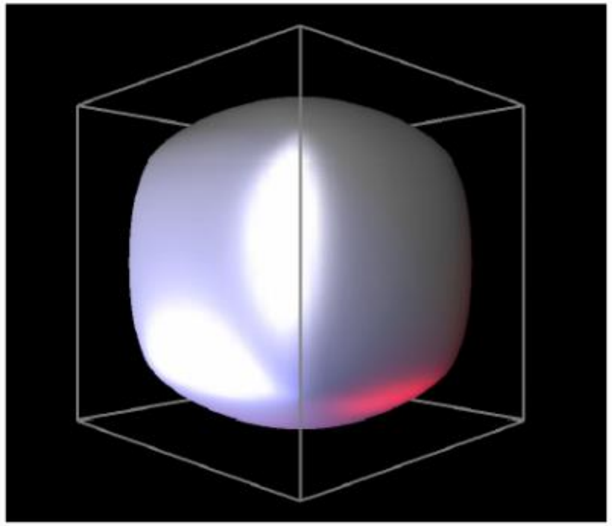
Adjust old point



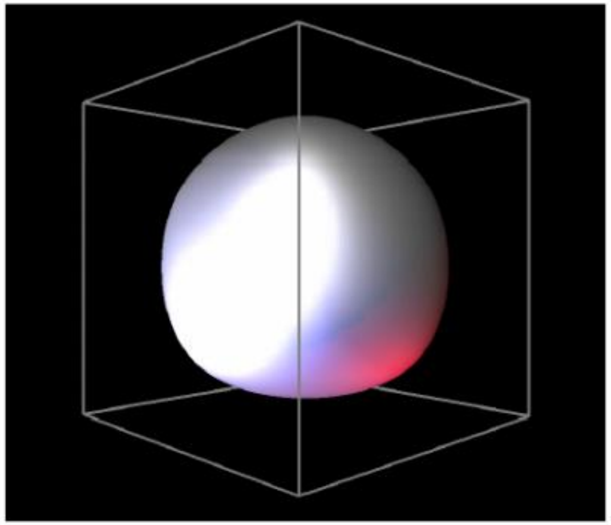
Comparison of Different Schemes



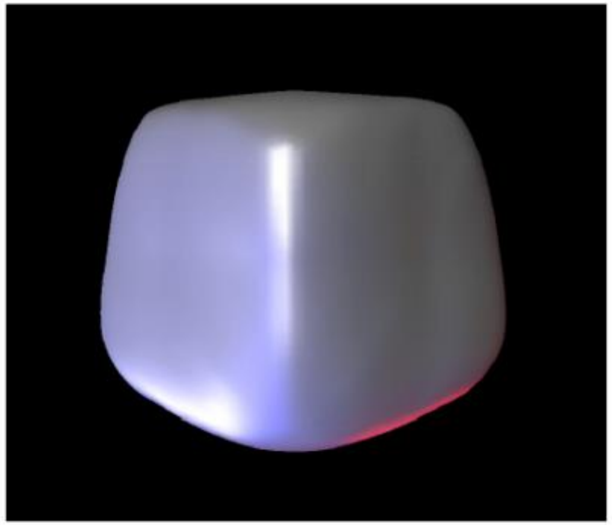
Catmull-Clark



Doo-Sabin



Loop



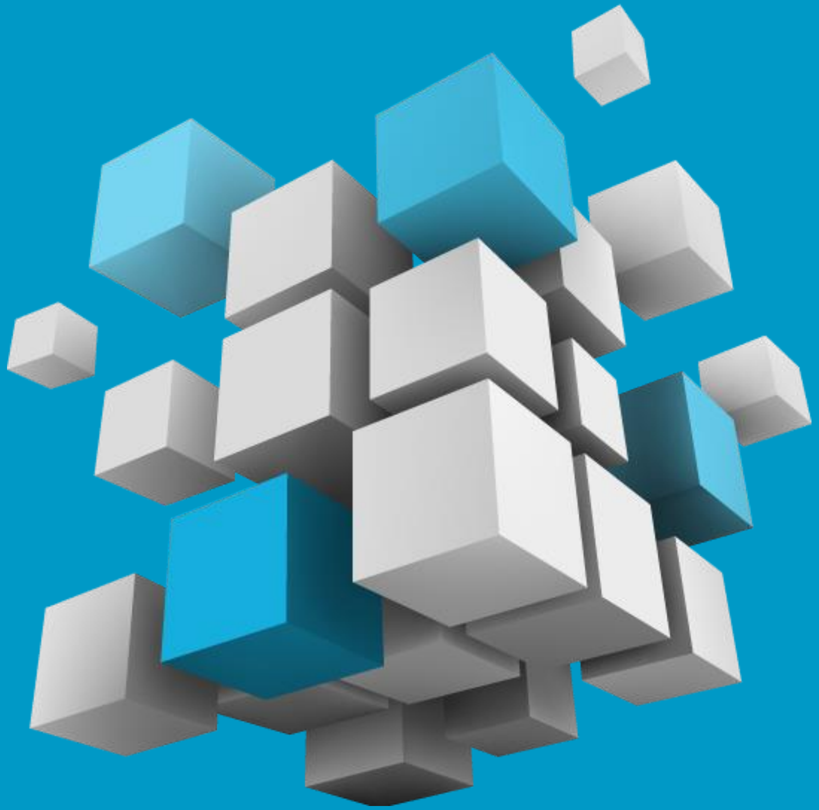
Butterfly

Other subdivision schemes

- Butterfly, Doo-Sabin, Midedge, Biquartic, Kobbelt, $\sqrt{3}$

Modeling

Parametric curves & surfaces



Brief History

[daimler.com]

1940s

Design using a 1:1 model, measurements at model

First mathematical model descriptions, splines by Schönberg

[formtrends.com]

1960s

de Casteljau (at Citroën)

non-rational curves (Ferguson at Boing)

free-form surfaces, Coons patch (Coons)

1970s

Bézier (at Renault)

NURBS (de Boor)

B-Splines (Riesenfeld)

Parametric Curves

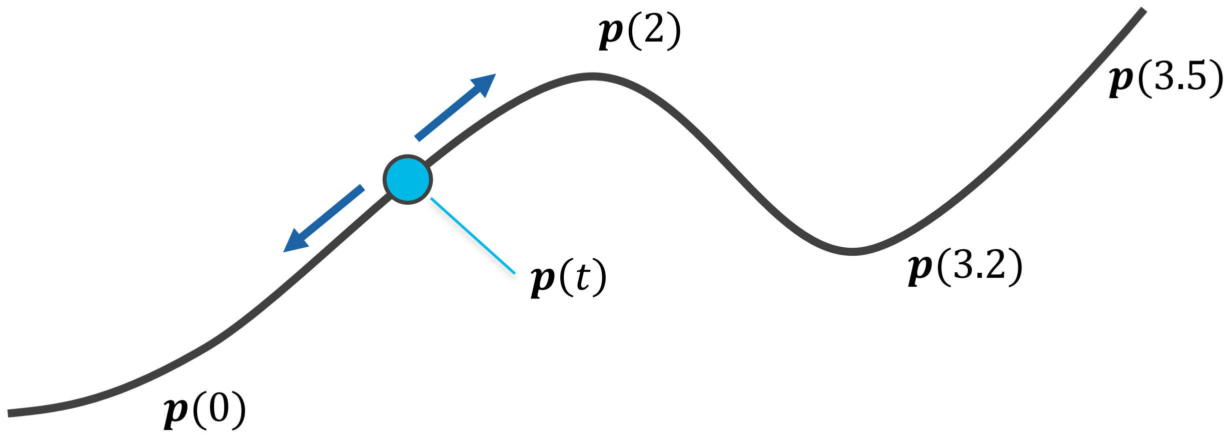
A parametric curve is controlled by a single parameter *t*

- *t* moves continuously along the curve

$$\boldsymbol{p}(t) = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} g(t) \\ h(t) \end{pmatrix}$$

Point on curve

Curve parameter *t*



A parametric circle

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_c + r\cos t \\ y_c + r\sin t \end{pmatrix} \quad \text{with} \quad t \in [0, 2\pi)$$

Range guarantees that t is unique for every point

Extends easily to 3D

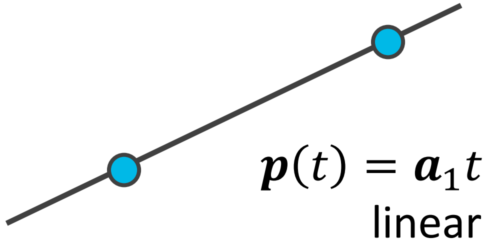
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} r\cos t \\ r\sin t \\ t \end{pmatrix}$$

A spiral around the z axis

Parametric Curves (cont.)

Well-behaved functions can be approximated with polynomials

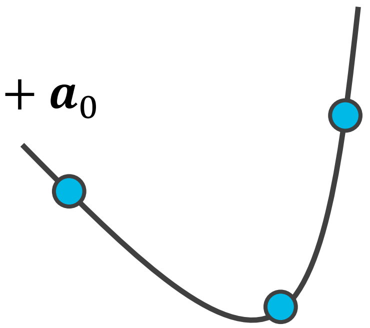
- Often piecewise and with low degree (usually cubic)



$\boldsymbol{p}(t) = \boldsymbol{a}_1 t + \boldsymbol{a}_0$
linear

Polynomial

$$f(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + \cdots + a_n t^n$$



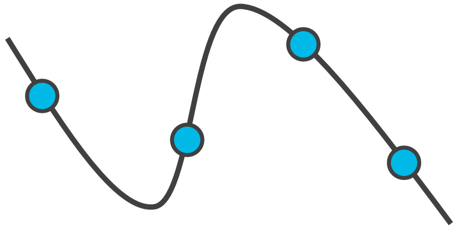
$\boldsymbol{p}(t) = \boldsymbol{a}_2 t^2 + \boldsymbol{a}_1 t + \boldsymbol{a}_0$
quadratic

And in canonical form

Degree n of the polynomial

$$\boldsymbol{f}(t) = \sum_{i=0}^{n} \boldsymbol{a}_i t^i$$

nD coefficients $\boldsymbol{a}_i$



$\boldsymbol{p}(t) = \boldsymbol{a}_3 t^3 + \boldsymbol{a}_2 t^2 + \boldsymbol{a}_1 t + \boldsymbol{a}_0$
cubic

Bézier Curves

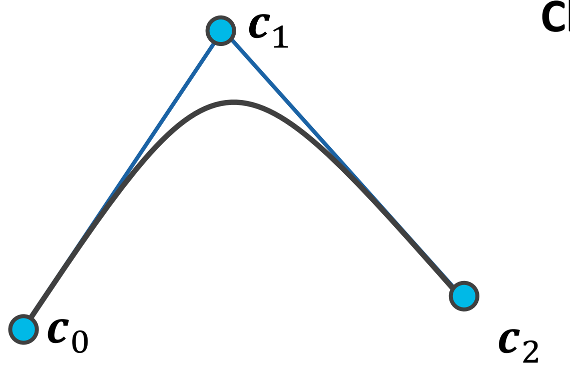
Curve of degree *d* is controlled by *d*+1 control points

- Specify endpoints and their tangents

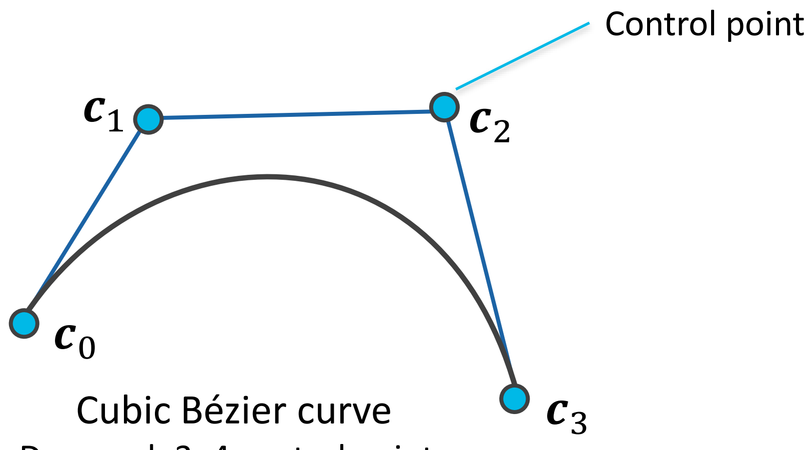
$$\boldsymbol{p}(t) = \sum_{i=0}^{n} B_i^n(t)\boldsymbol{c}_i$$

Control points

Bernstein polynomials



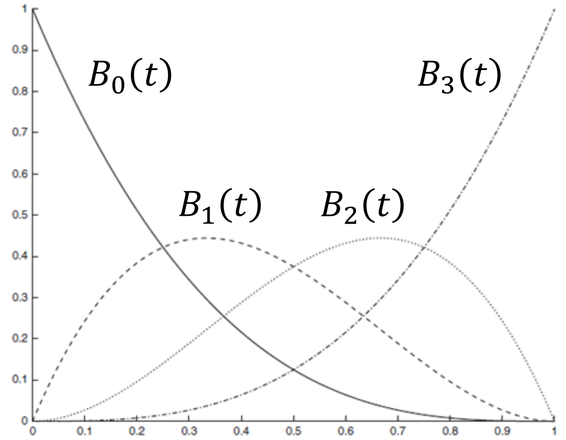
Quadratic Bézier curve
Degree d=2, 3 control points



Cubic Bézier curve
Degree d=3, 4 control points

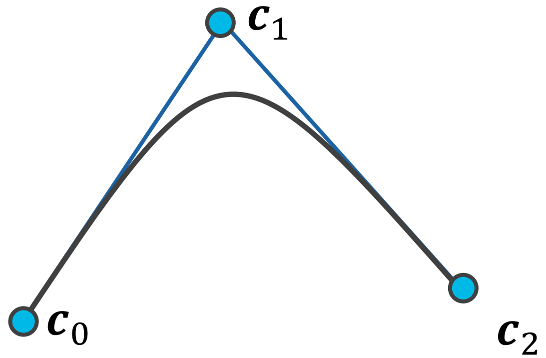
Cubic Bézier curve, *d*=3

$$B_0(t) = (1-t)^3$$
$$B_1(t) = 3t(1-t)^2$$
$$B_2(t) = 3t^2(1-t)$$
$$B_3(t) = t^3$$



Cubic Bernstein polynomials

Properties of Bézier Curves



Quadratic Bézier curve
Degree d=2, 3 control points

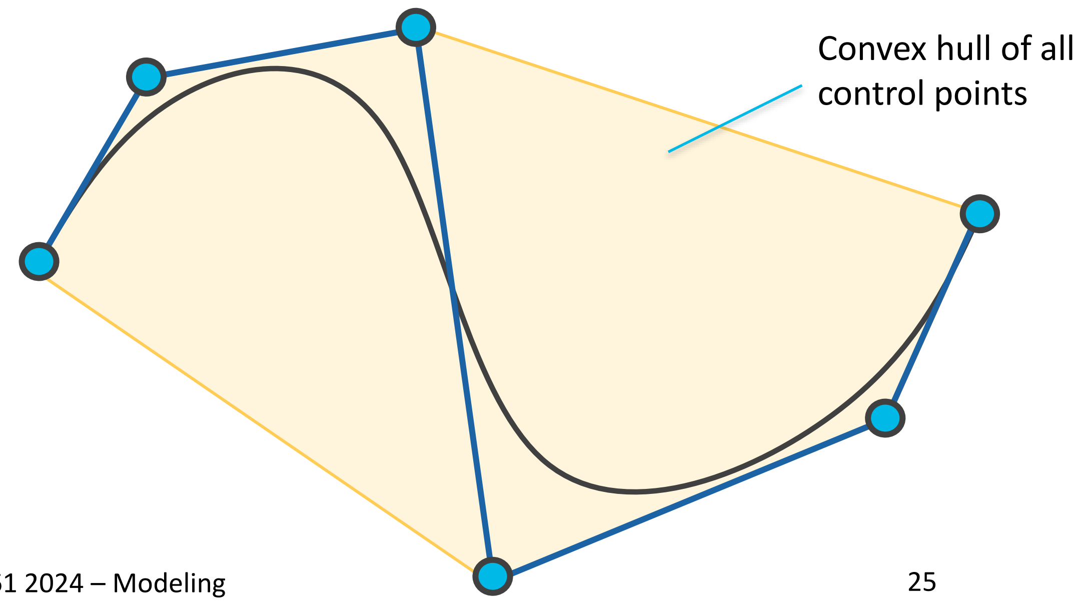
Curve of degree *d* is controlled by *d*+1 control points

Curve ends at first & last control point

Curve never leaves the convex hull of the control points

Endpoint tangents point to next/previous control point

- Curve follows the tangents



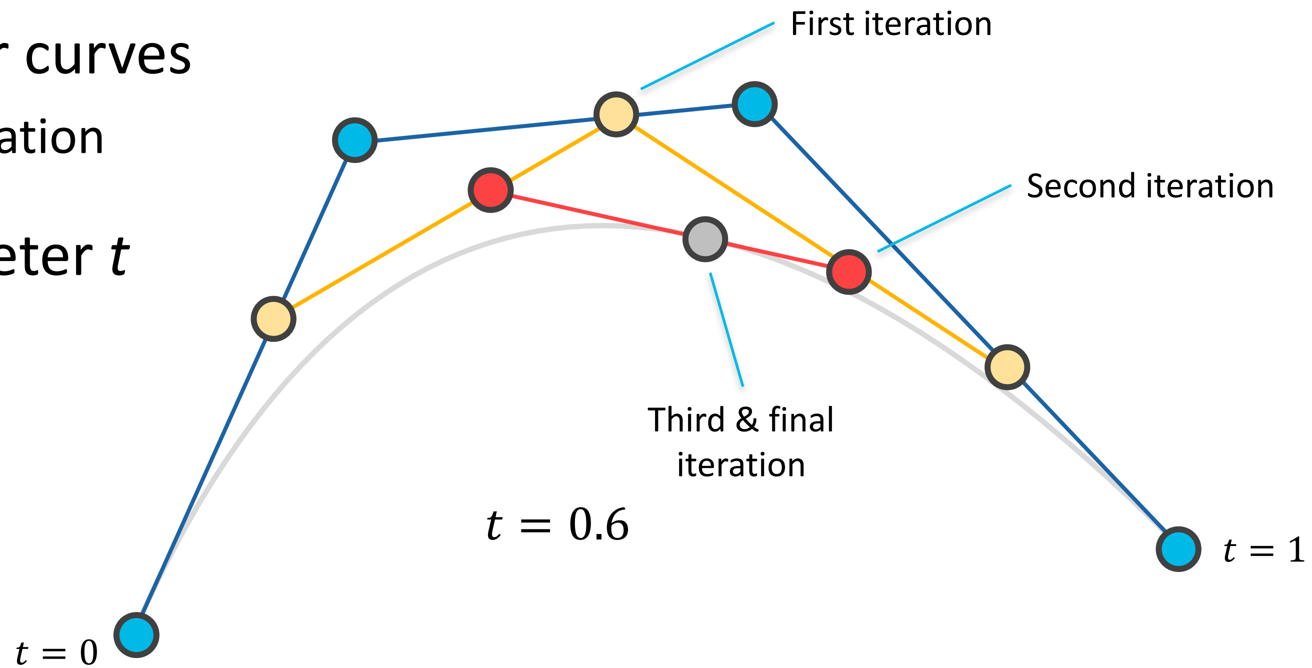
De Casteljau's Algorithm

Recursive algorithm to evaluate Bézier curves

- More numerically stable than direct evaluation

Determine position for a given parameter t

- Divide each line segment at t
- Connect new points with lines
- Continue until only 1 point is created



Parametric Surfaces

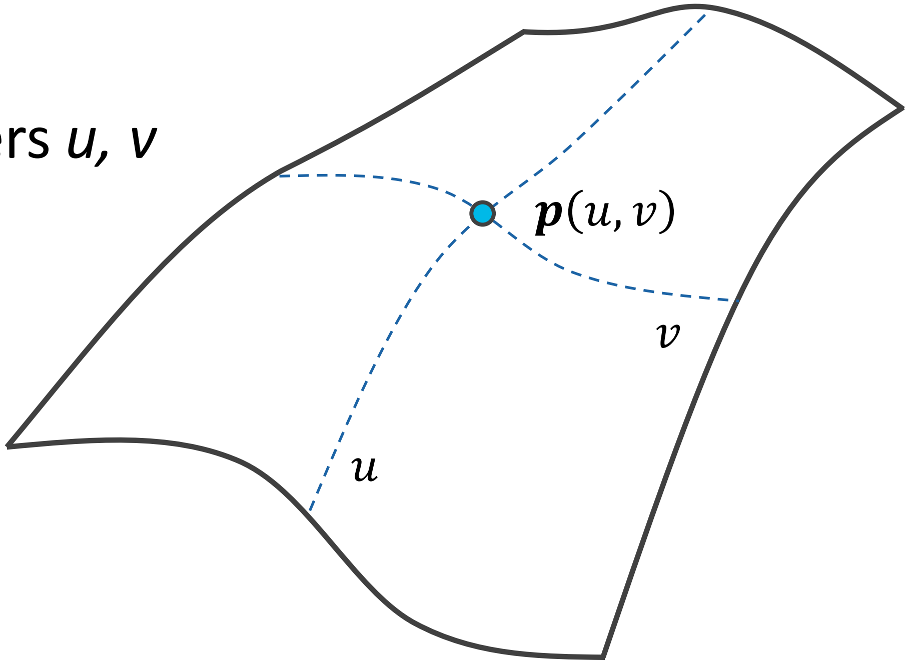
A parametric surface is controlled by two parameters *u, v*

- *u, v* move over surface

$$\boldsymbol{p}(u,v) = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f(u,v) \\ g(u,v) \\ h(u,v) \end{pmatrix}$$

Point on surface

Surface parameters *u, v*



Parametric sphere

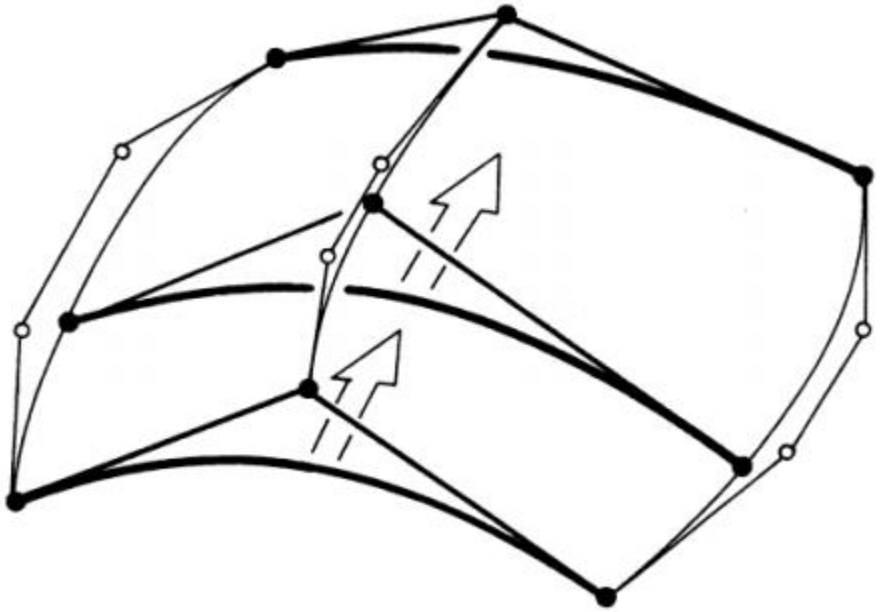
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} r\cos\phi\sin\theta \\ r\sin\phi\sin\theta \\ r\cos\theta \end{pmatrix}$$ with $\phi \in [0, 2\pi)$ and $\theta \in [-\pi, \pi)$

Bézier Patches

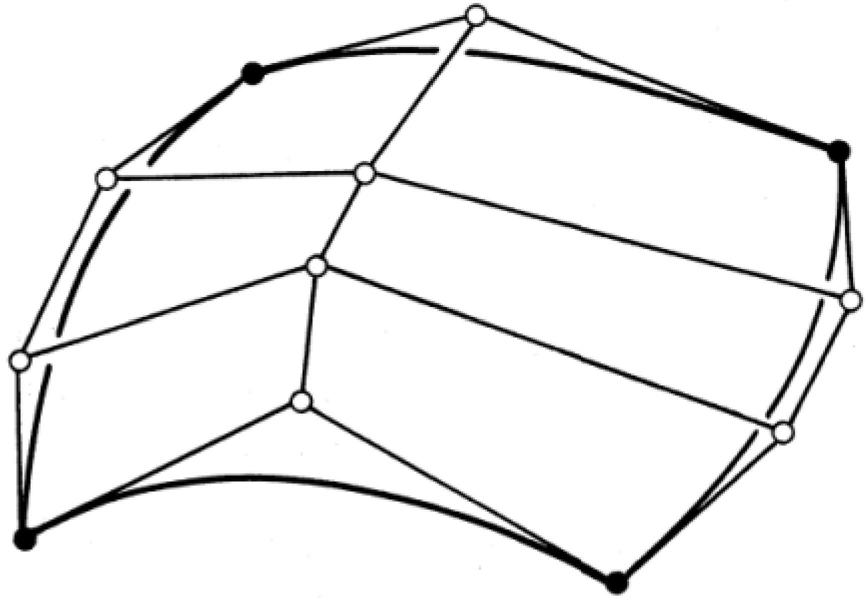
Surface is created by moving a Bézier spline through space

Curve degree can be different for *u* and *v*

$$\boldsymbol{p}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)\, B_j^m(v)\boldsymbol{c}_i$$



Surface construction

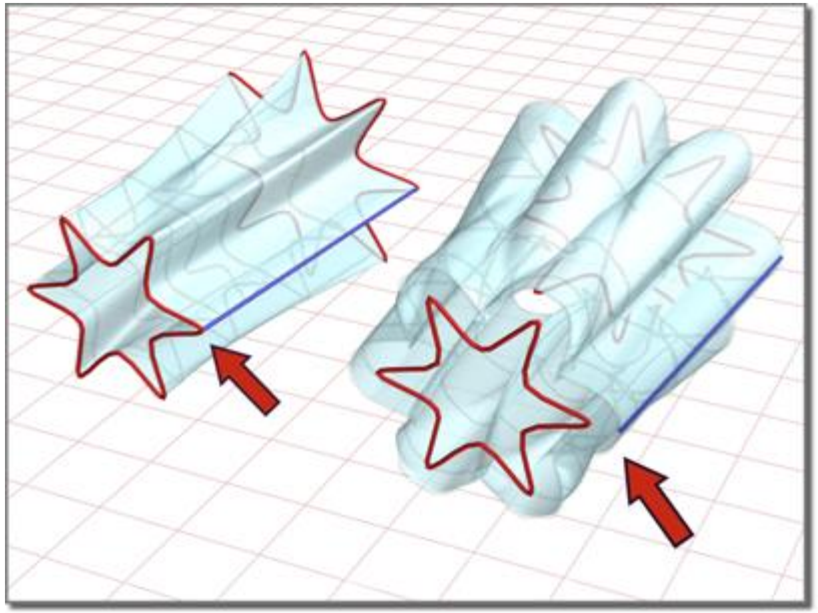


Surface with control mesh

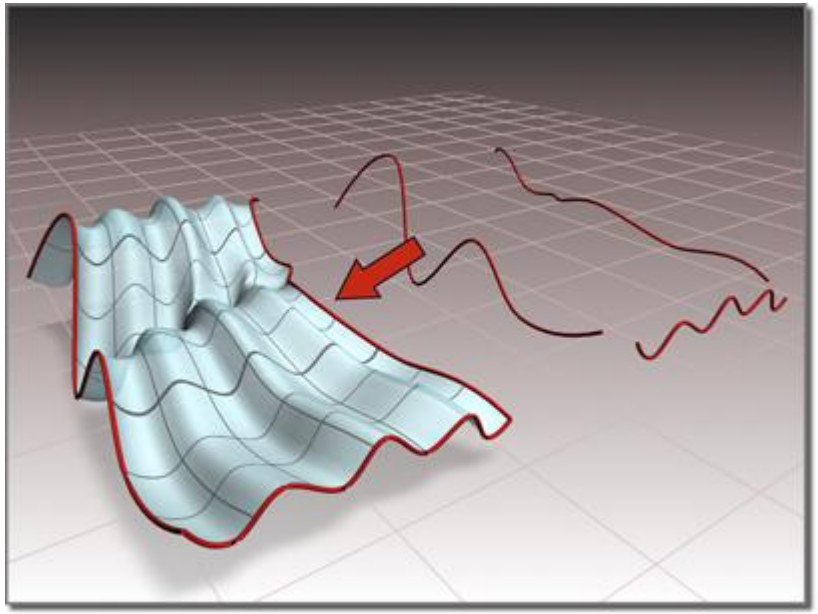
Modeling: Swept Surface

Move a curve along one or more curves

Distance to guide curve determines scaling



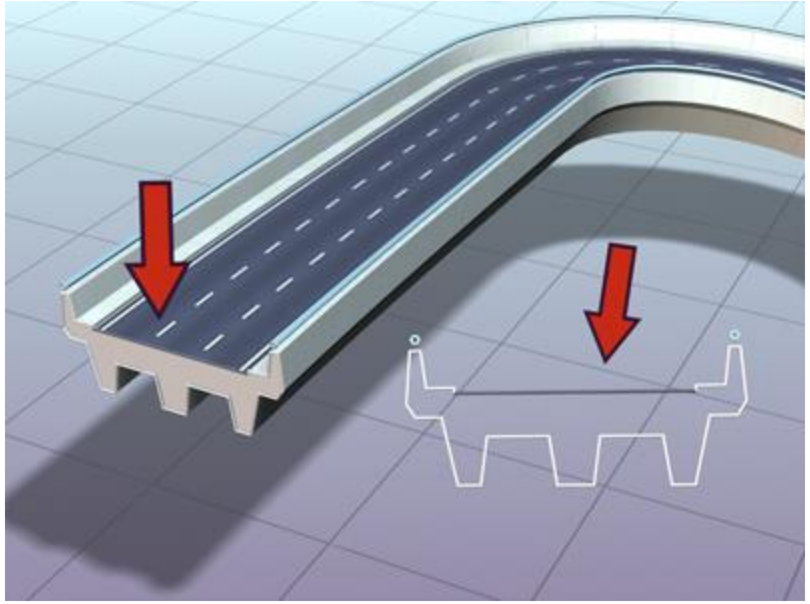
1-Rail Sweep Surface



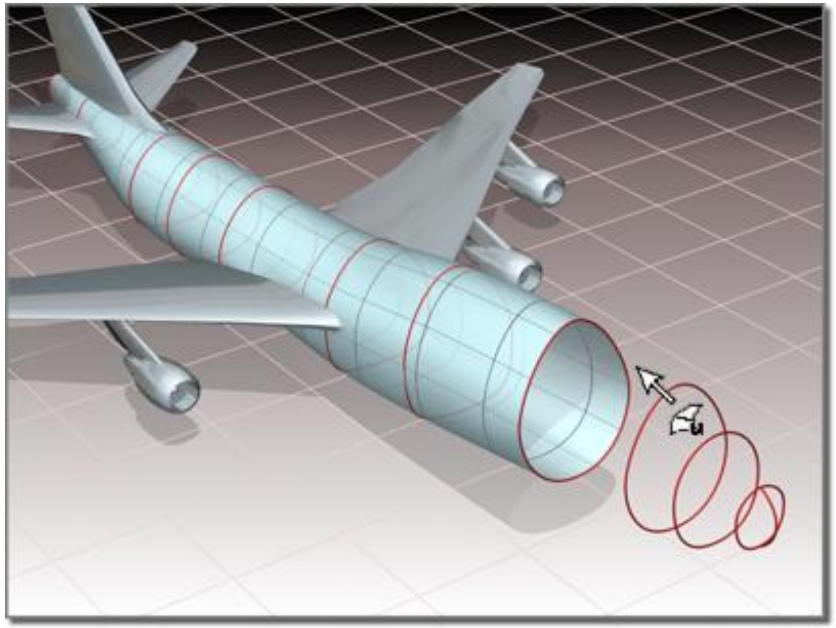
Sweep surface created with two rails

Modeling: Loft Surface

Move a 2D shape along a guide curve



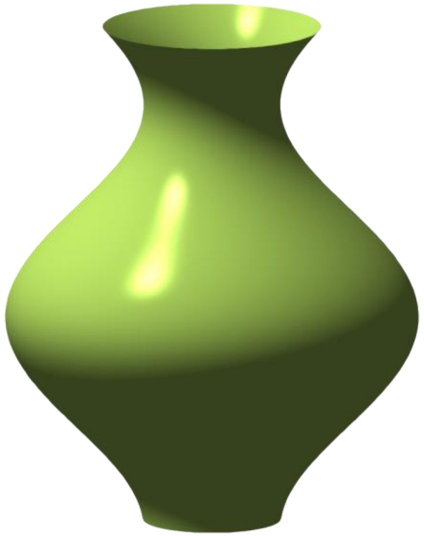
Loft with compound 2D shape

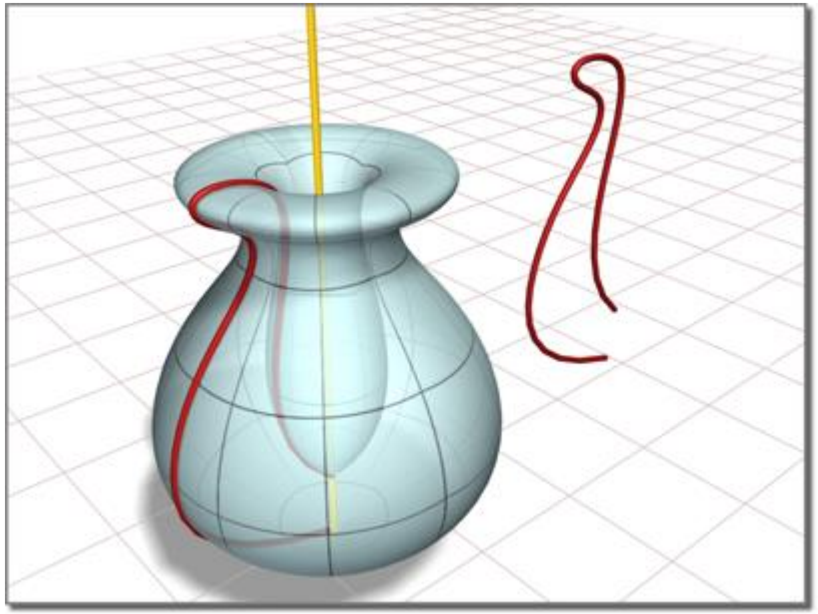


Loft with multiple cross sections

Revolution Surfaces

Generate a surface by rotating a curve around an axis

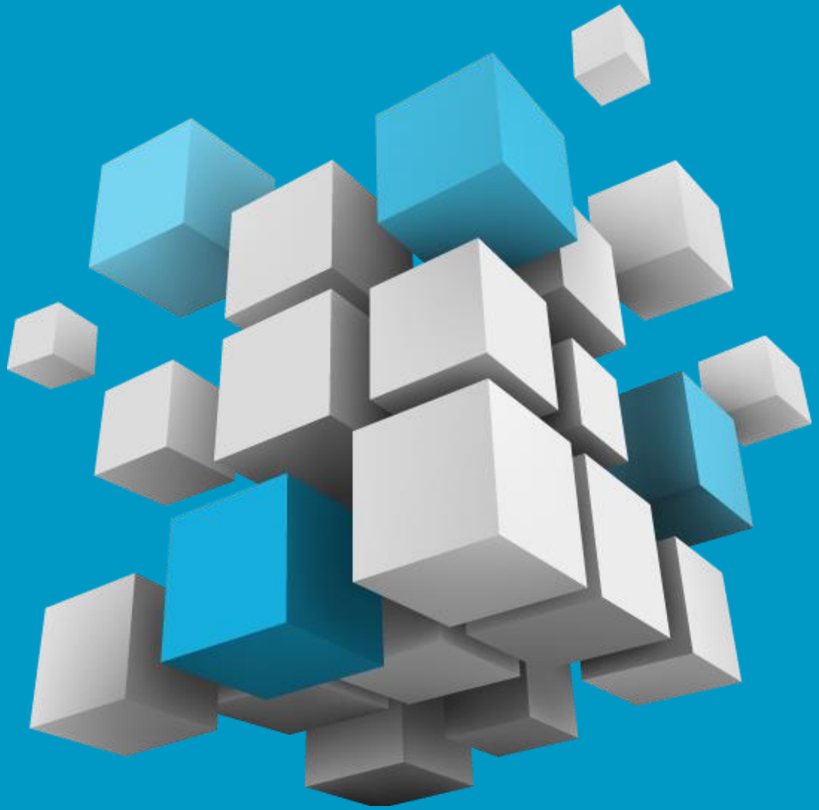




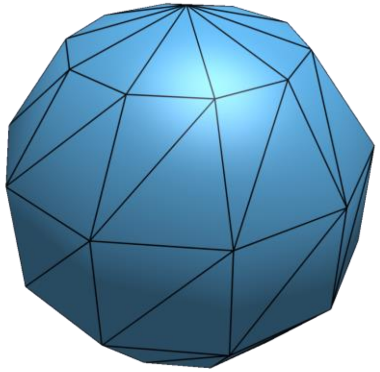
Surface created by rotating a curve

Modeling

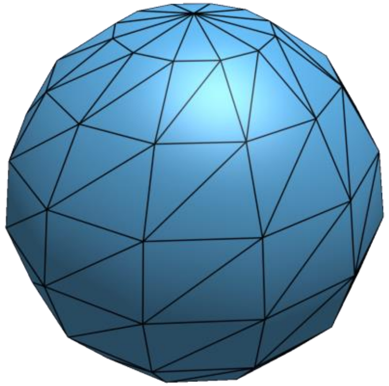
Implicit modeling & CSG



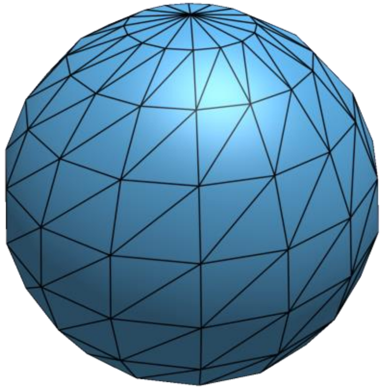
Example: Spheres



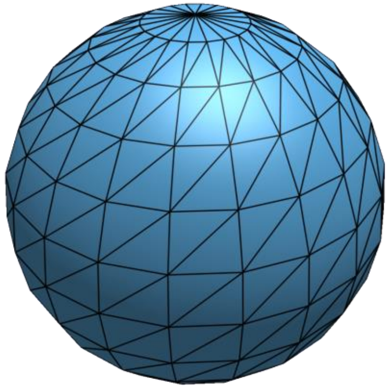
66 vertices



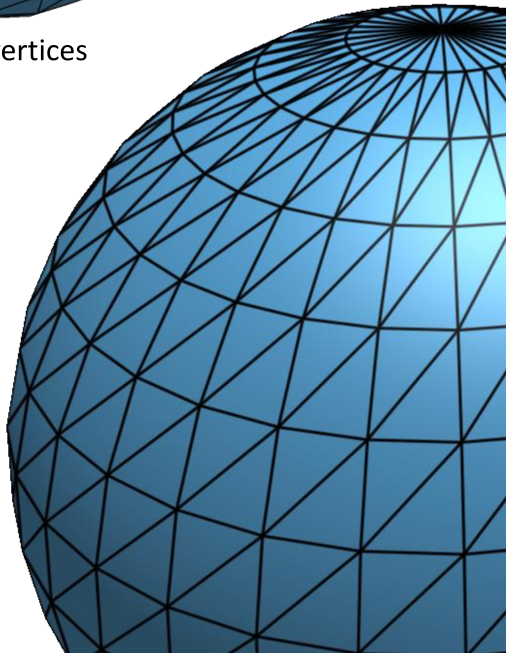
117 vertices



187 vertices



273 vertices



561 vertices

Polygonal meshes rough for low vertex counts

- Sufficient from far away, but not in close-ups

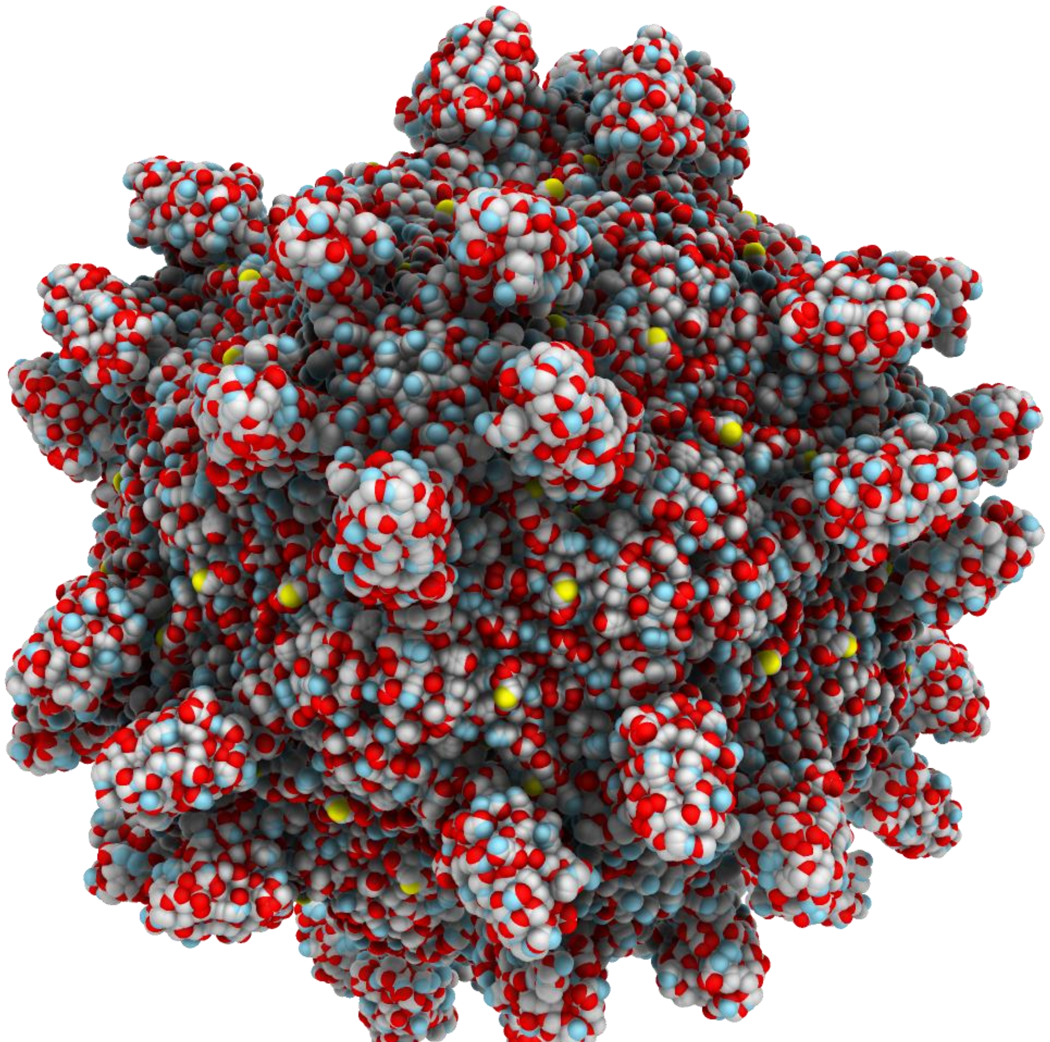
Does not scale for many spheres (data transfer, memory)

Ideally triangulation depends on camera distance

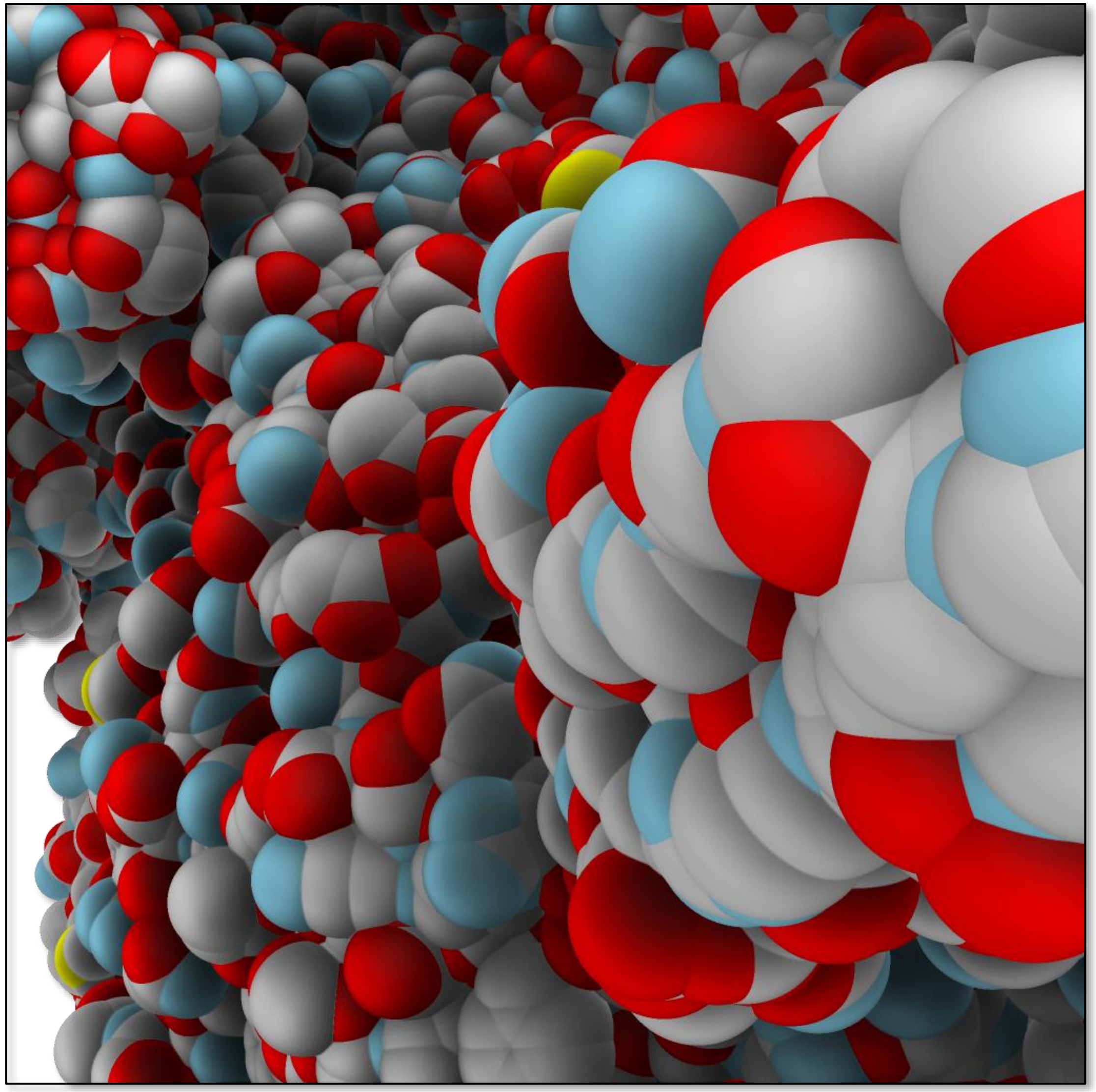
- Requires tessellation or geometry shader

Implicit modeling solves all theses issues

Implicit Rendering with “infinite” detail



Grapevine Fanleaf Virus (294k atoms/points)



LINKÖPING UNIVERSITY

Implicit Modeling

Define curves with implicit equations

$$f(x, y) = 0$$
$$f(x, y, z) = 0$$

- *Implicit* = not solved for x, y, or z

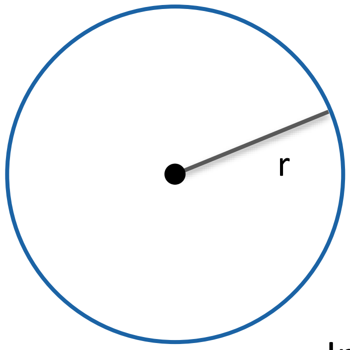
Corresponds to a point set

Curves in 2D, surfaces in 3D

- Circle, plane, sphere, cylinder, cone, …

Advantages

- Infinite level of detail
- Efficient for computing intersections



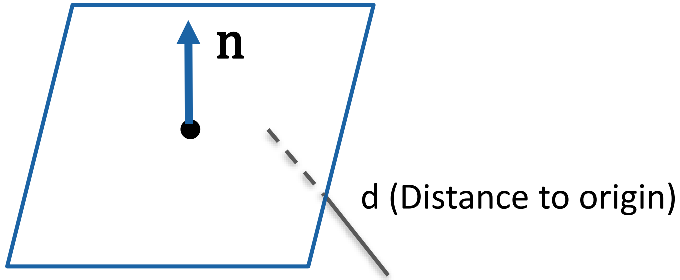
$$f(x, y) = x^2 + y^2 - r^2 = 0$$

if $f(x, y) > 0$ ➔ outside

if $f(x, y) < 0$ ➔ inside

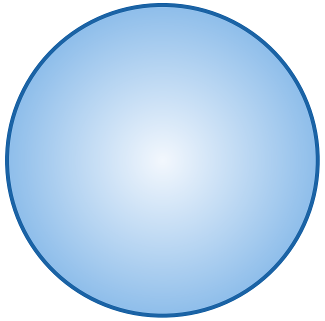
Implicit circle

Implicit Modeling (cont.)



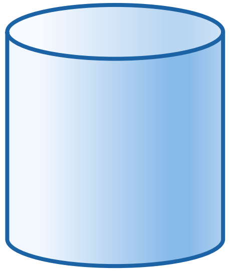
Plane

$$f(x, y, z) = n_x x + n_y y + n_z z + d = 0$$



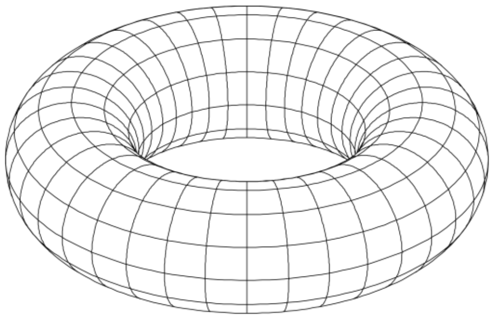
Sphere

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2 = 0$$

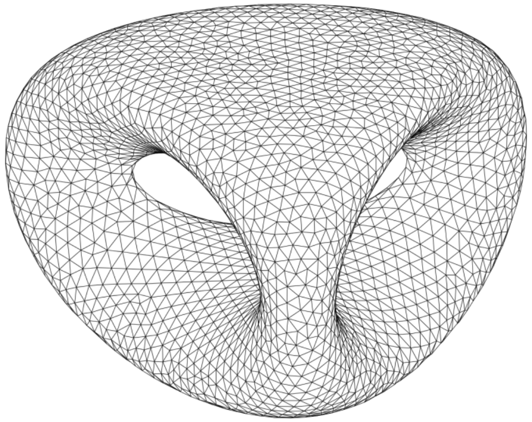


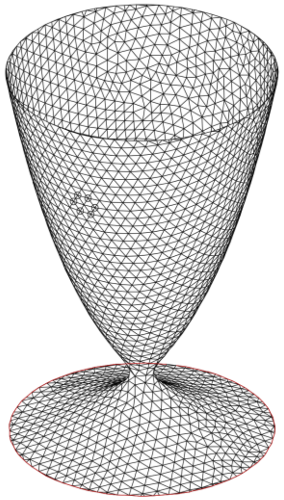
Cylinder (infinite, $\pm z$)

$$(x - x_0)^2 + (y - y_0)^2 - r^2 = 0$$



Torus





[wikipedia]

Rendering

- **Implicit:** Compute ray intersections
- **Explicit:** Convert to polygon mesh using surface triangulation
  - For example, Marching Cubes algorithm

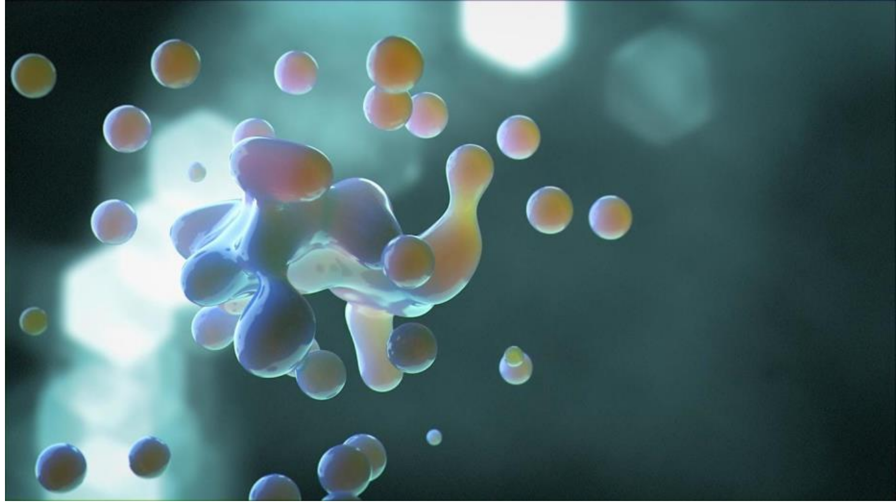
Metaballs

Implicit function depends on the distance between a point $\boldsymbol{p}$ and a set of particles $\boldsymbol{p}_i$

$$f(\boldsymbol{p}) = \sum_i D(|\boldsymbol{p} - \boldsymbol{p}_i|) - T$$

Threshold (isovalue)

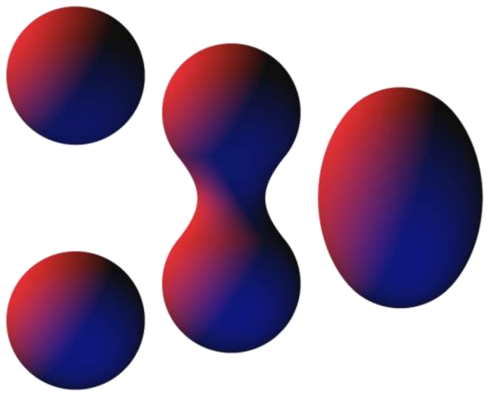
Distance function

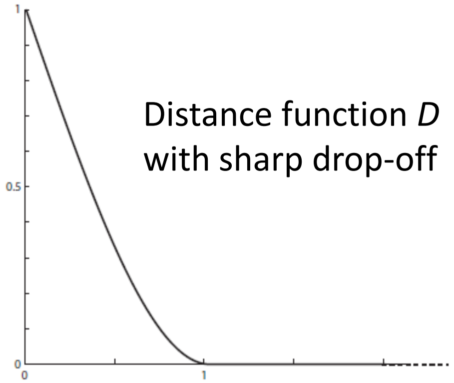


© Elmar Glaubauf [chaosgroup.com]

Creates an organic look

- Can be used to model liquids and droplets





Distance function *D* with sharp drop-off

© Philip 486 [dribbble.com]

Constructive Solid Geometry (CSG)

Technique for solid modeling

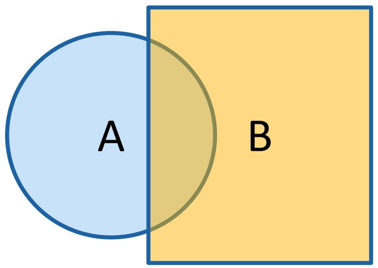
- Set of Boolean operations

Often found in CAD

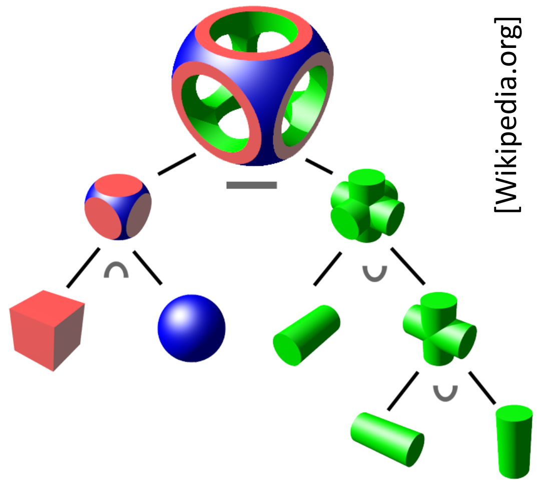
- Corresponds to cutting, drilling, welding, …

Requires a closed surface

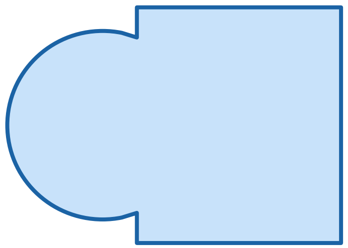
- Well-defined outside and inside

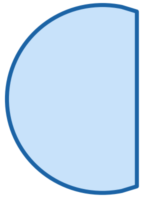


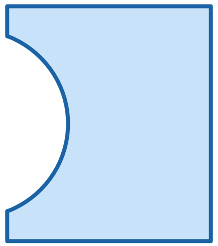
Input

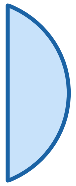

[Wikipedia.org]

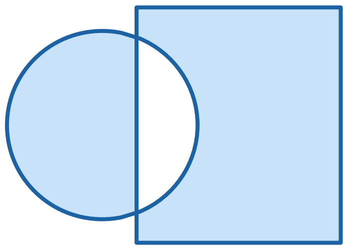
Binary tree of CSG operations


Union
A or B


Subtraction
A not B

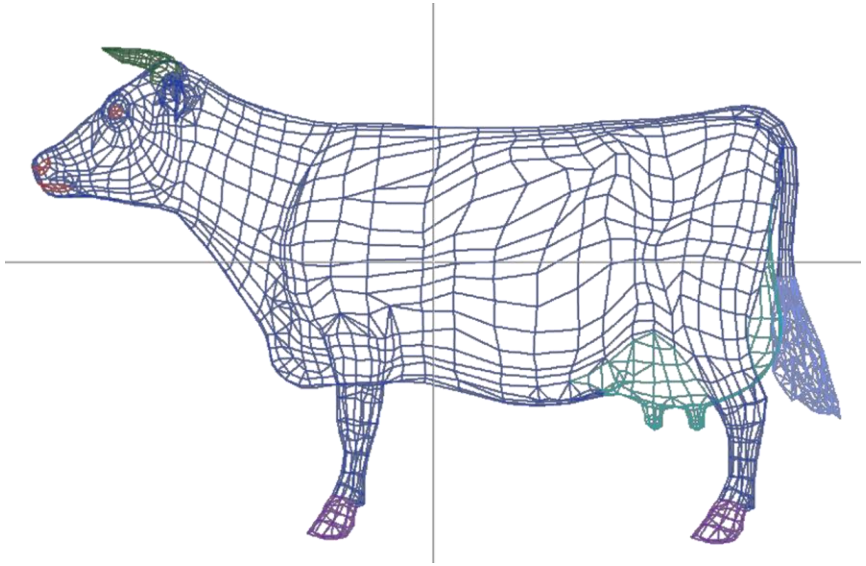

Subtraction
B not A


Intersection
A and B


Combine
A xor B

Summary

– Modeling –



Polygon meshes

- Quad meshes for modeling, triangles for rendering
- Half-edge structure for adjacency
- Subdivision surfaces

Parametric curves & surfaces

- Cubic Bézier curves & patches in Computer Graphics
- Compact representation & resolution-independent
- Intuitively modifiable via control points

Implicit modeling & CSG

References – Computer Graphics at a Glance

[Gumhold 2003] Stefan Gumhold. *Splatting illuminated ellipsoids with depth correction*. International Fall Workshop on Vision, Modelling and Visualization (VMV 2003), pp. 245-252, 2003.

[Klein 2004] Thomas Klein and Thomas Ertl. *Illustrating Magnetic Field Lines using a Discrete Particle Model*. Workshop on Vision, Modelling and Visualization (VMV 2004), pp. 387-394, 2004.

3ds max tutorials on modeling

- Tutorials
  https://area.autodesk.com/all/tutorials/3ds-max/
- Polygon modeling
  https://area.autodesk.com/tutorials/series/getting-started-in-3ds-max-2018/getting-started-polygon-modeling-part-1/
- Spline modeling
  https://area.autodesk.com/tutorials/3ds-max-modeling-techniques-modeling-with-splines/

Coming up next

Materials

- Illumination Models
- Texturing
- Reflection mapping, bump mapping
- Complex materials, procedural textures