CSCI 420 Computer Graphics

Lecture 9

Polygon Meshes and Implicit Surfaces

Polygon Meshes

Implicit Surfaces

Constructive Solid Geometry

[Angel Ch. 10]

Oded Stein
University of Southern California

What is a surface?

- Mathematics: Differential geometry
- Artist: Something I can edit with a tool
- Computer programmer: Data structure

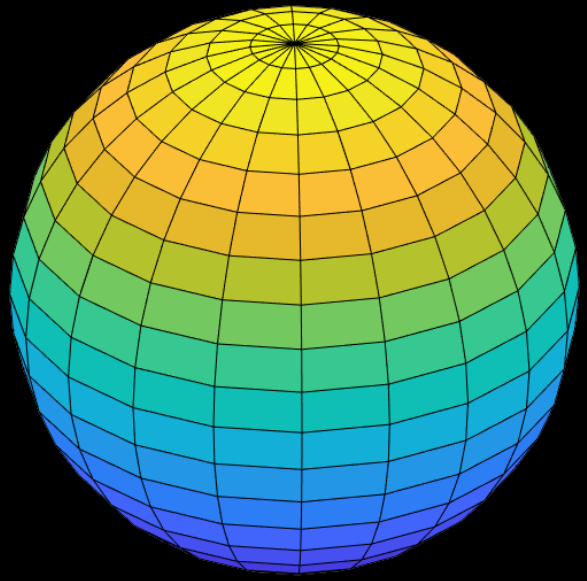
mathematician

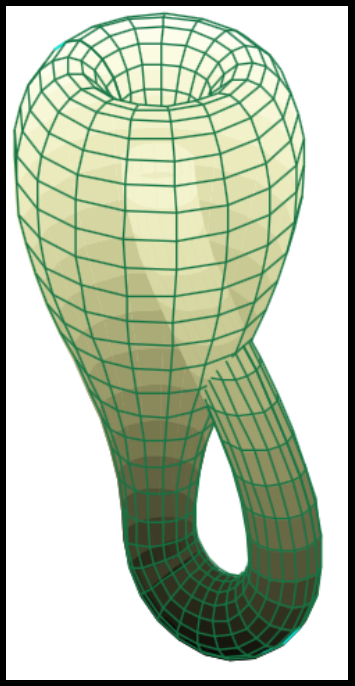
artist

programmer

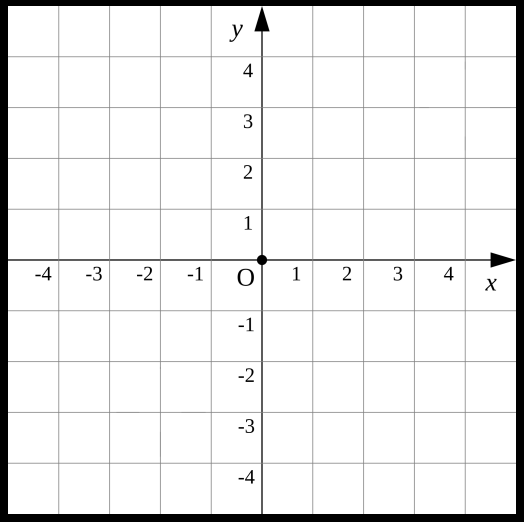
Mathematics of surfaces

A surface is an object that is locally a plane





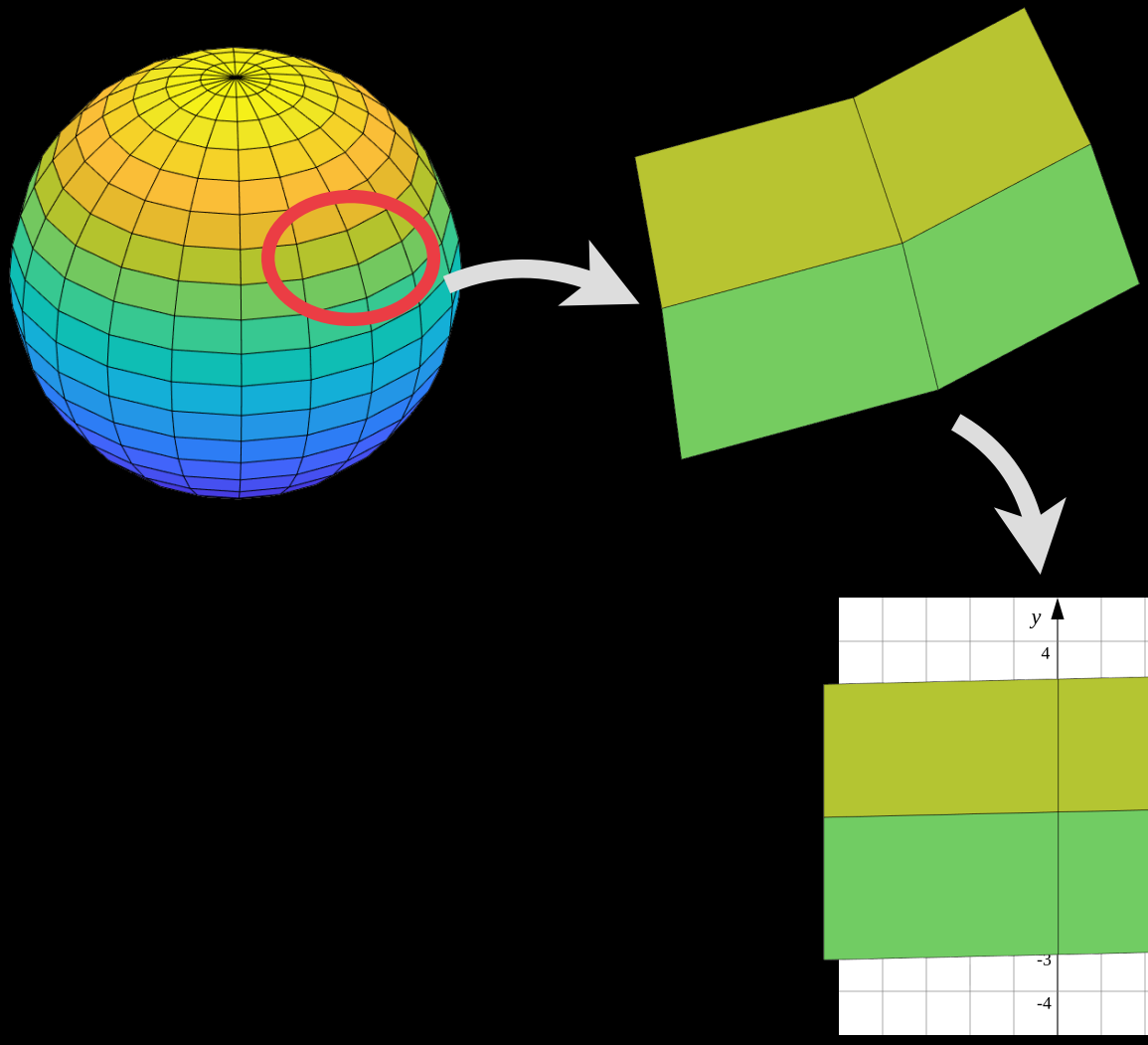
the plane R^2



Mathematics of surfaces

A surface is an object that is locally a plane

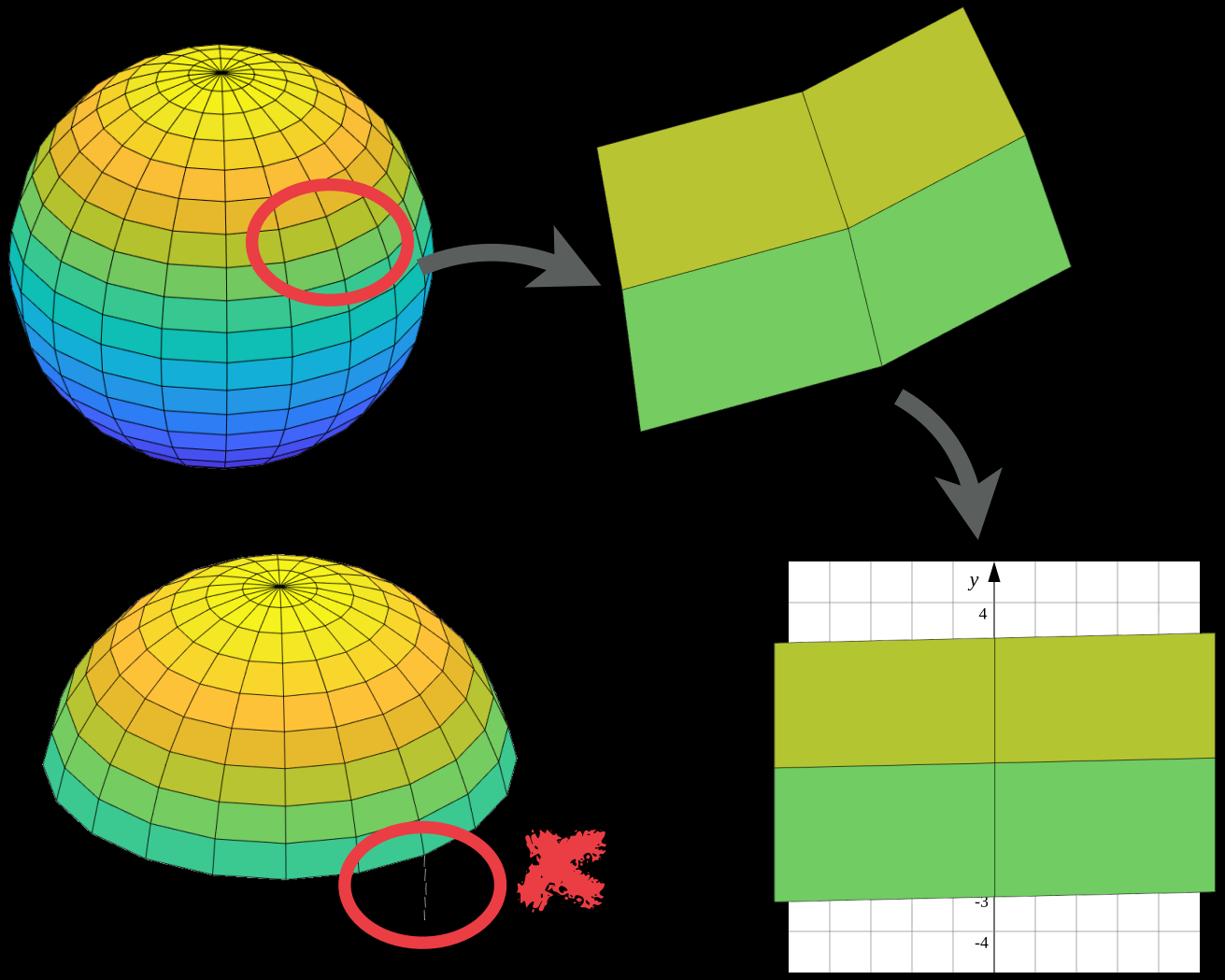
- I can pick a small part of a surface and squish it so it looks like the plane.



Mathematics of surfaces

A surface is an object that is locally a plane

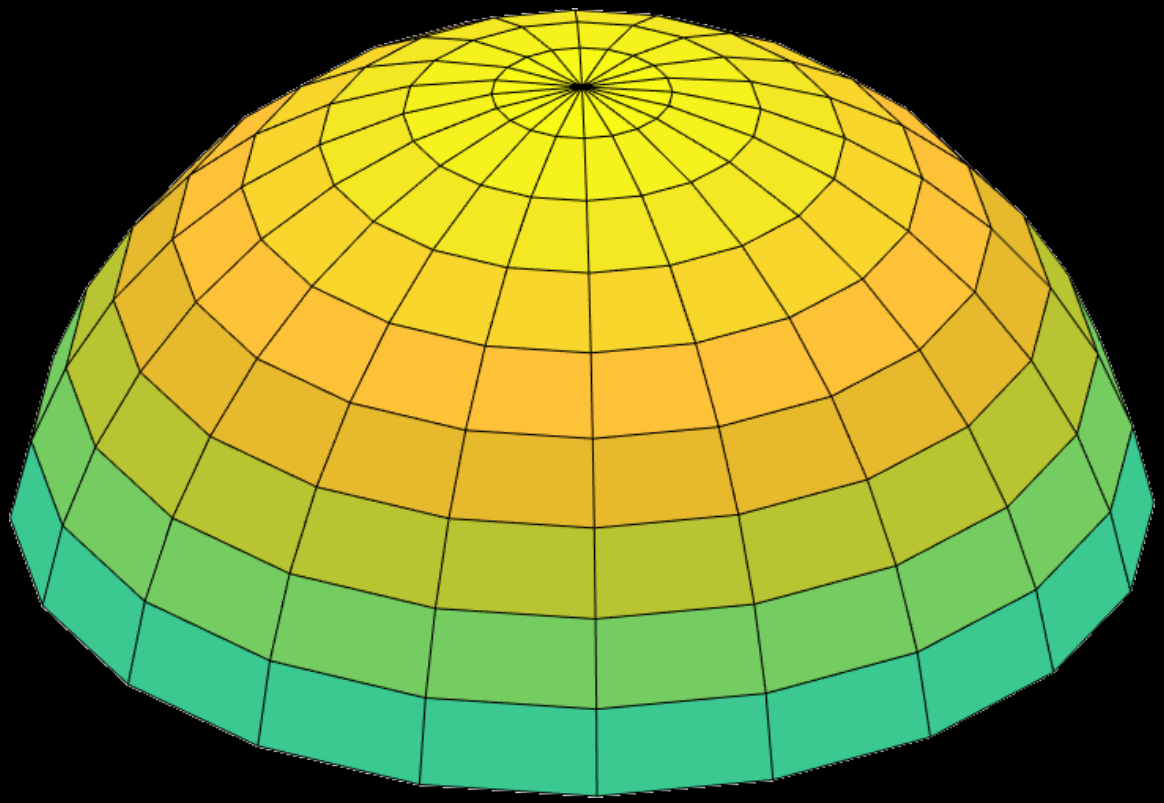
- I can pick a small part of a surface and squish it so it looks like the plane.



Mathematics of surfaces

A surface is an object that is locally a plane

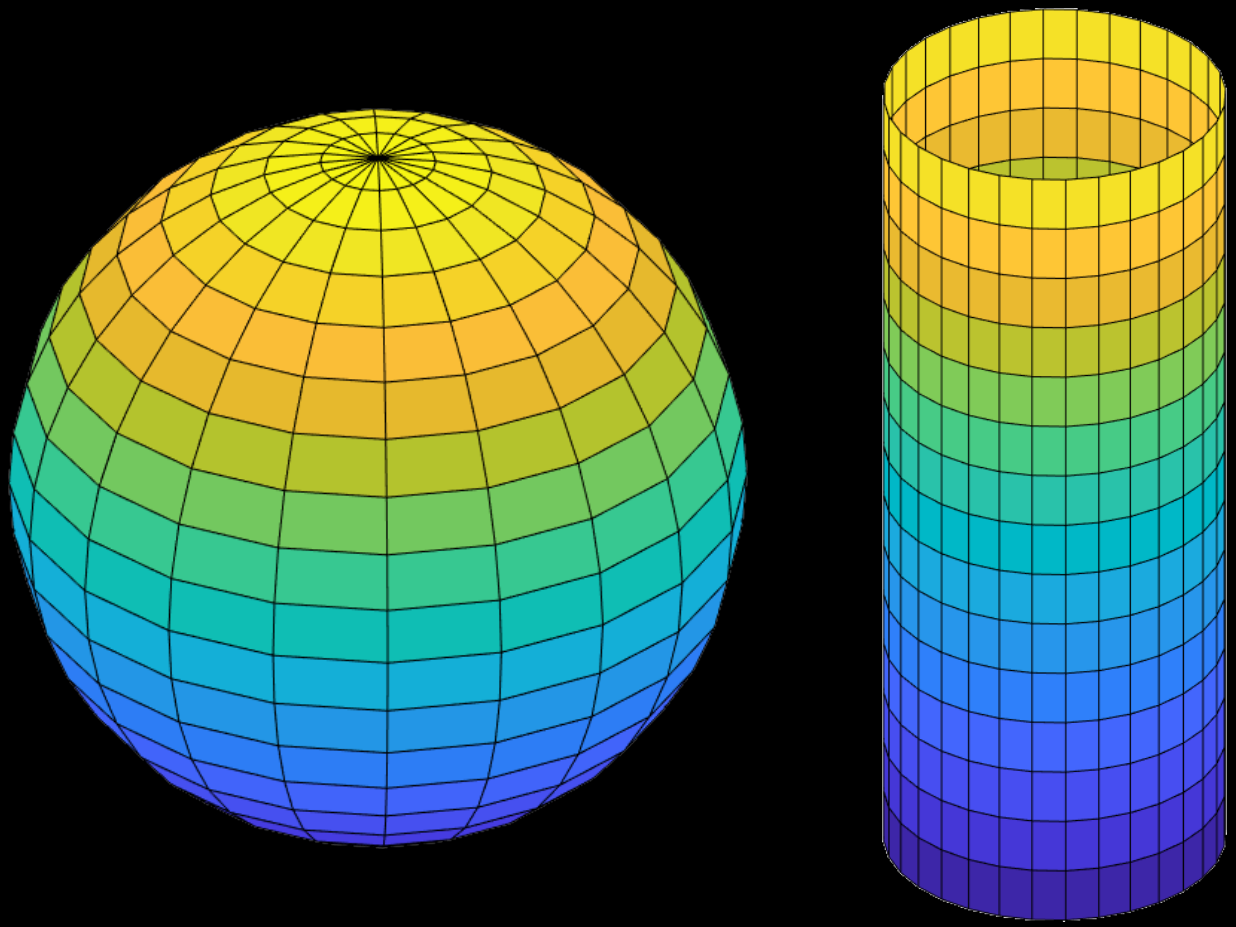
- I can pick a small part of a surface and squish it so it looks like the plane.
- A surface can have a boundary that looks like a half-plane.



Mathematics of surfaces

A surface is an object that is locally a plane

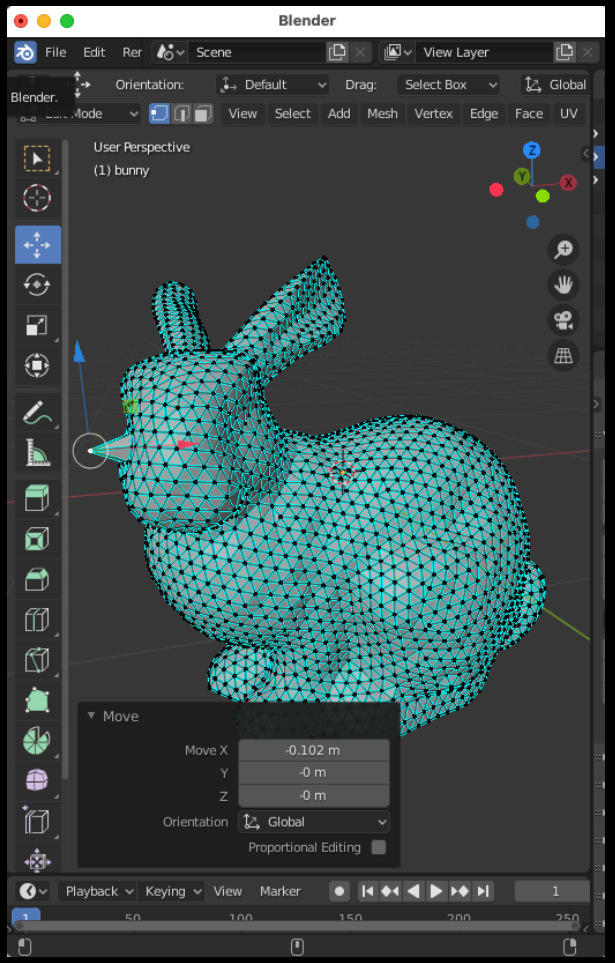
- I can pick a small part of a surface and squish it so it looks like the plane.
- A surface can have a boundary that looks like a half-plane.
- Globally, a surface does not have to look like a plane.



Artist's view of surfaces

- A surface is an 3D object that I create in a modeling software and is displayed in my engine / renderer.
- They are your users!

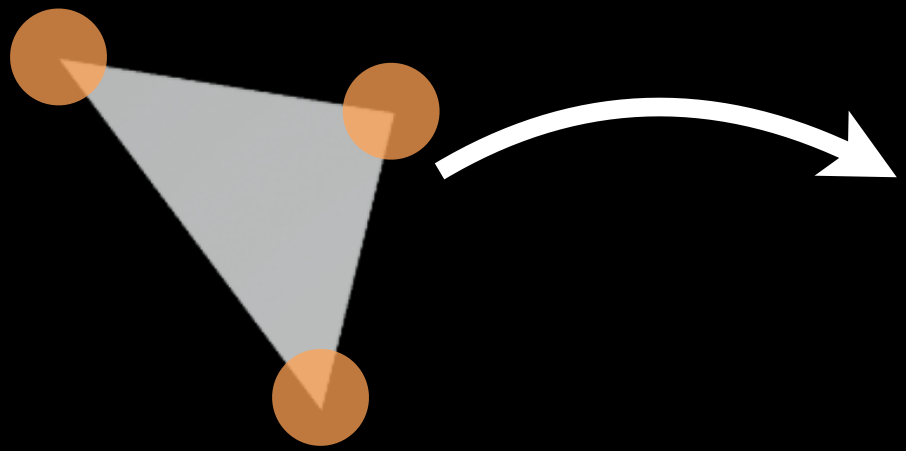
surface



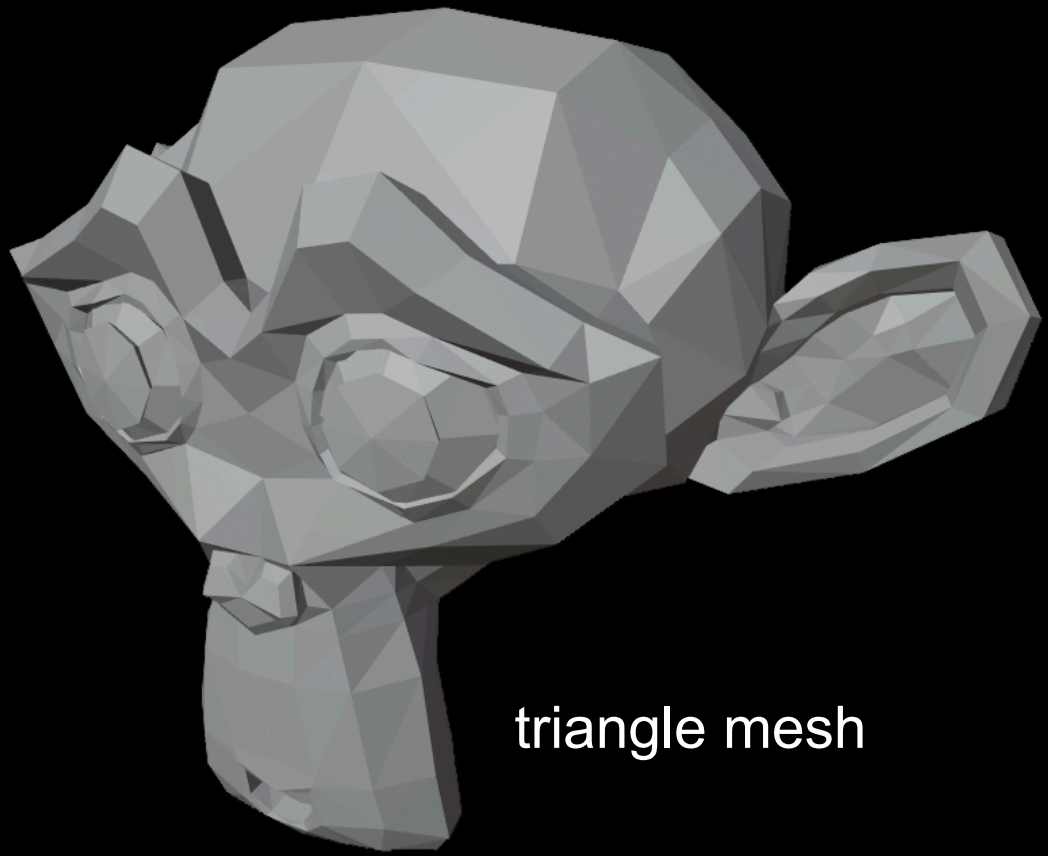
modeling software

Programmer's view of surfaces

A surface is some sort of representation of a shape as data.



vertices



triangle mesh

Programmer's view of surfaces

A surface is some sort of representation of a shape as data.

quad mesh

point cloud

Modeling Complex Shapes

Source: Wikipedia

- An equation for a sphere is possible, but how about an equation for a telephone, or a face?
- Complexity is achieved using simple pieces
  - polygons, parametric surfaces, or implicit surfaces
- Goals
  - Model *anything* with arbitrary precision (in principle)
  - Easy to build and modify
  - Efficient computations (for rendering, collisions, etc.)
  - Easy to implement (a minor consideration...)

What do we need from shapes in Computer Graphics?

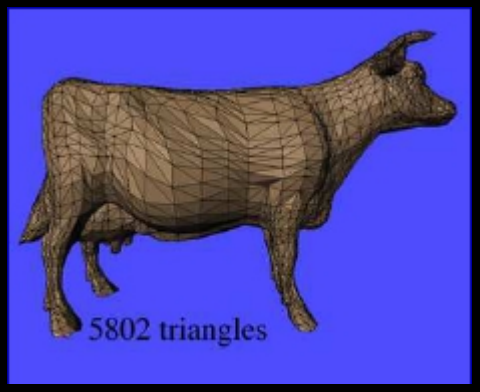
- Local control of shape for modeling
- Ability to model what we need
- Smoothness and continuity
- Ability to evaluate derivatives
- Ability to do collision detection
- Ease of rendering

No single technique solves all problems!

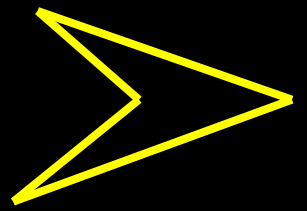
Shape Representations

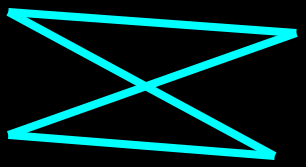
Polygon Meshes
Parametric Surfaces
Implicit Surfaces

Polygon Meshes

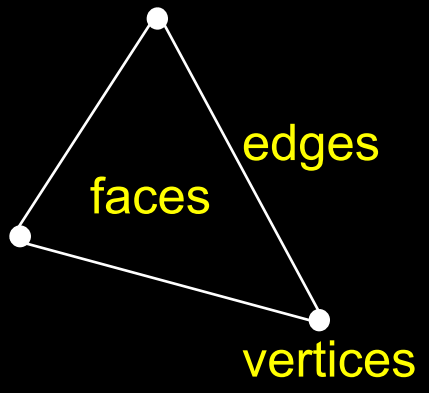


- Any shape can be modeled out of polygons
  - if you use enough of them…
- Polygons with how many sides?
  - Can use triangles, quadrilaterals, pentagons, … n-gons
  - Triangles are most common.
  - When > 3 sides are used, ambiguity about what to do when polygon nonplanar, or concave, or self-intersecting.





- Polygon meshes are built out of
  - *vertices* (points)
  - *edges* (line segments between vertices)
  - *faces* (polygons bounded by edges)



What sets polygon meshes apart?

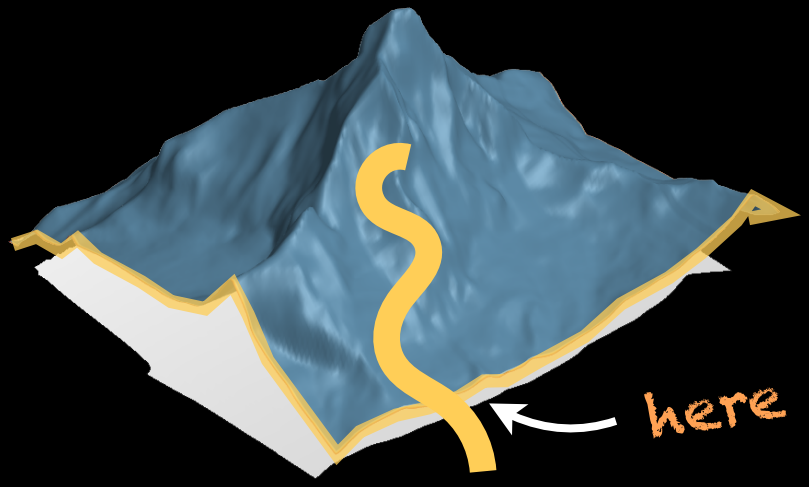
- **Explicit** surface representation
  - No need to solve an equation before drawing

- Solid
  - Not just samples on the smooth object, but solid approximation

- Have geometric quantities like smooth surfaces
  - Area
  - Boundary
  - Normals

Boundary of a surface

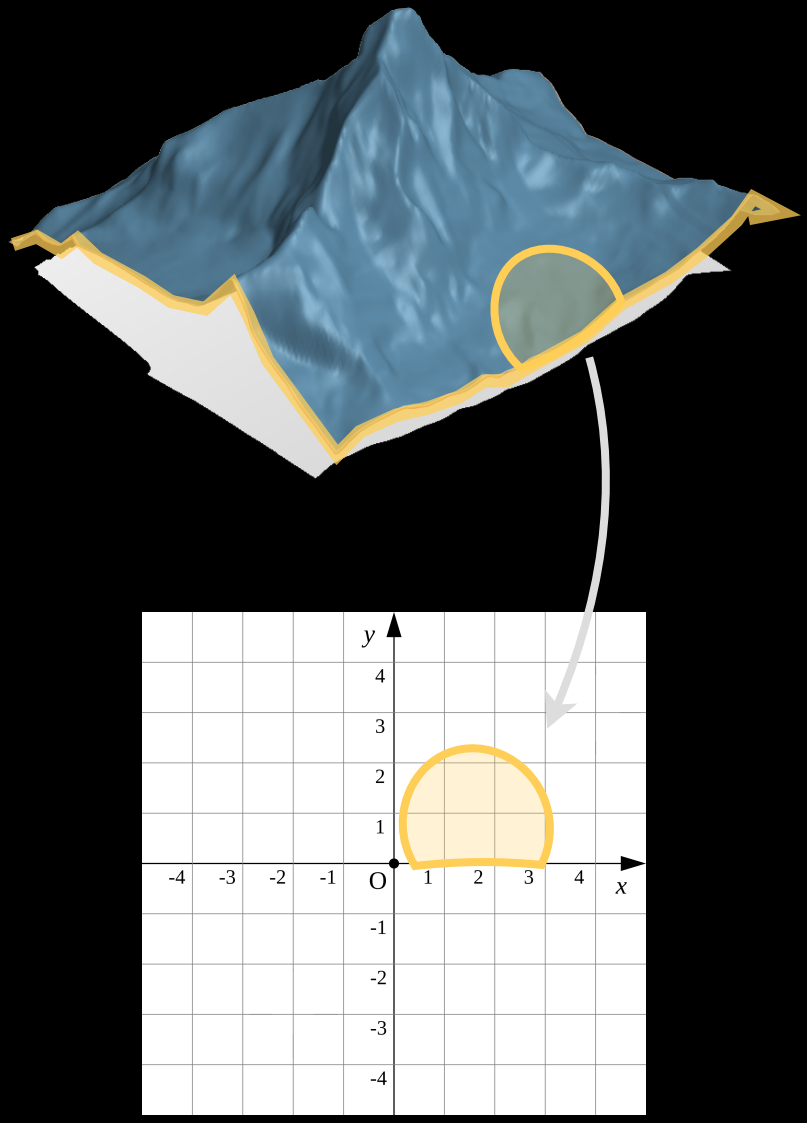
- intuitively: when you walk, you fall off the surface



Boundary of a surface

- intuitively: when you walk, you fall off the surface

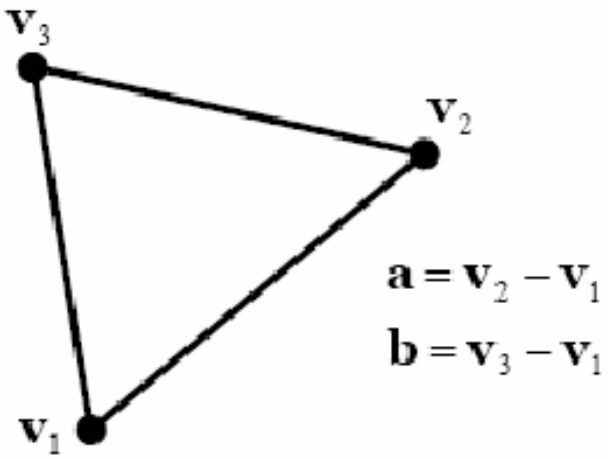
- mathematically: maps to the upper half-plane of R^2



Boundary of a surface

- intuitively: when you walk, you fall off the surface
- mathematically: maps to the upper half-plane of R^2
- on triangle meshes: all boundary edges

Normals



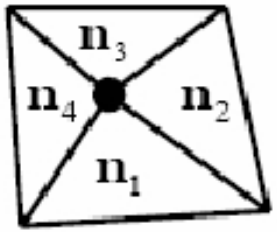
Triangle defines unique plane

- can easily compute normal

$$\mathbf{n} = \frac{\mathbf{a} \times \mathbf{b}}{\|\mathbf{a} \times \mathbf{b}\|}$$

- depends on vertex orientation!
- clockwise order gives

$$\mathbf{n}' = -\mathbf{n}$$

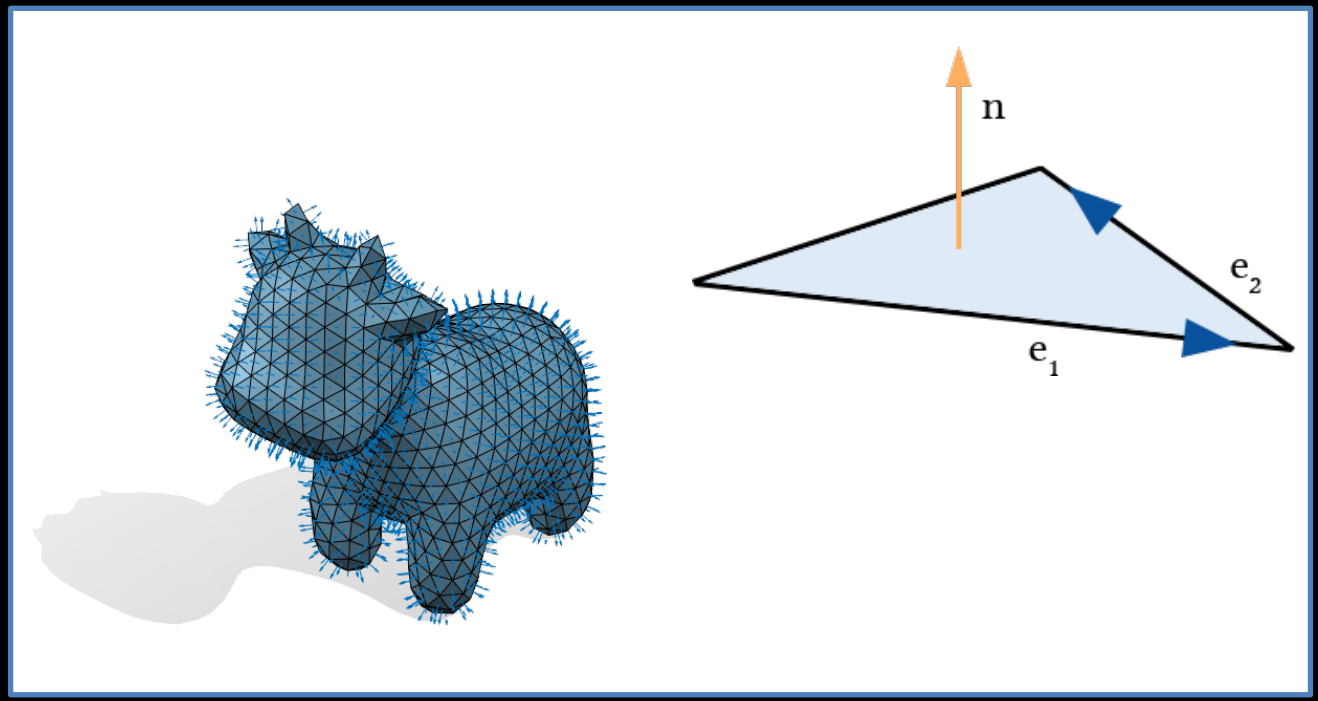


Vertex normals less well defined

- can average face normals
- works for smooth surfaces
- but not at sharp corners
  - think of a cube

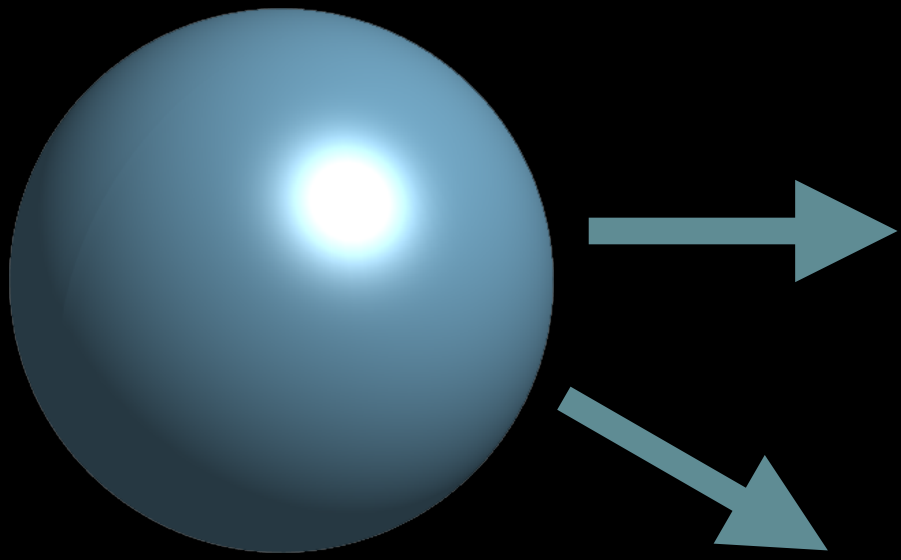
Per-face normals

- The normal vector $\mathbf{n}$ is the unit-length perpendicular vector to a triangle and positively oriented.
- $\tilde{\mathbf{n}} = \mathbf{e}_1 \times \mathbf{e}_2, \quad \mathbf{n} = \tilde{\mathbf{n}}/\|\tilde{\mathbf{n}}\|$
- normals point outside



Per-vertex normals

- smooth surfaces have normals at every point
- we might want to approximate normals at vertices too
- average per-vertex normals from our per-face normals



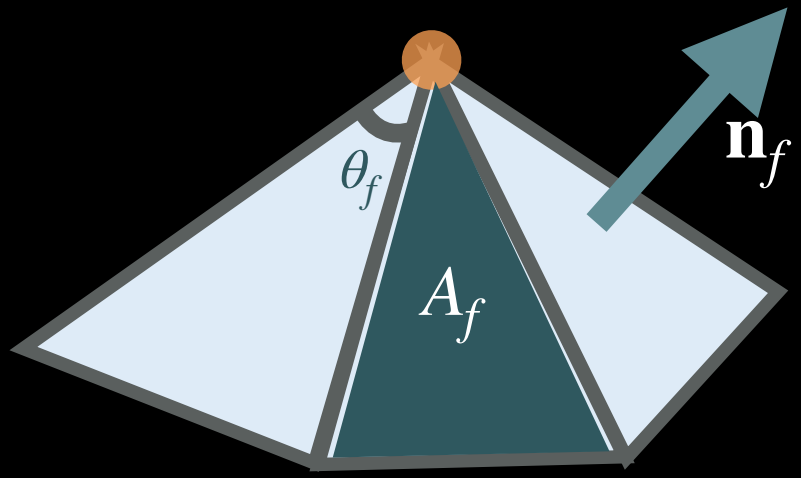
$$\tilde{\mathbf{n}}_v = \sum_{\text{face } f \text{ contains } v} w_f \, \mathbf{n}_f$$

$$\mathbf{n}_v = \tilde{\mathbf{n}}_v / \|\tilde{\mathbf{n}}_v\|$$

Per-vertex normals

$$\tilde{\mathbf{n}}_v = \sum_{\text{face } f \text{ contains } v} w_f \, \mathbf{n}_f \qquad \text{weight} \qquad \mathbf{n}_v = \tilde{\mathbf{n}}_v / \|\tilde{\mathbf{n}}_v\|$$

- smooth surfaces have normals at every point
- we might want to approximate normals at vertices too
- average per-vertex normals from our per-face normals



Where Meshes Come From

- Specify manually
  - Write out all polygons
  - Write some code to generate them
  - Interactive editing: move vertices in space

- Acquisition from real objects
  - Laser scanners, vision systems
  - Generate set of points on the surface
  - Need to convert to polygons

Data Structures for Polygon Meshes

- Simplest (but dumb)
  - float triangle[n][3][3]; (each triangle stores 3 (x,y,z) points)
  - redundant: each vertex stored multiple times
- Vertex List, Face List
  - List of vertices, each vertex consists of (x,y,z) geometric (shape) info only
  - List of triangles, each a triple of vertex id's (or pointers) topological (connectivity, adjacency) info only

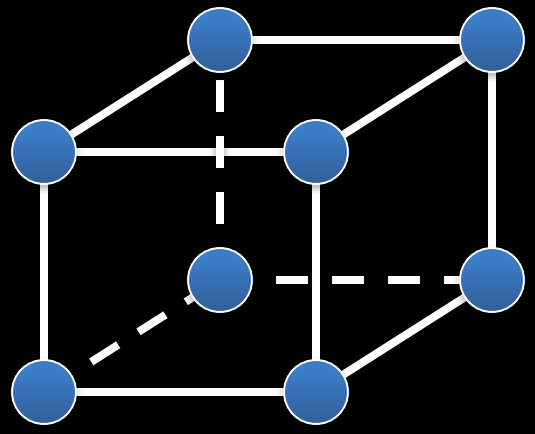
  Fine for many purposes, but finding the faces adjacent to a vertex takes O(F) time for a model with F faces. Such queries are important for topological editing.
- Fancier schemes:

  Store more topological info so adjacency queries can be answered in O(1) time.

  Winged-edge data structure – edge structures contain all topological info (pointers to adjacent vertices, edges, and faces).

A File Format for Polygon Models: OBJ

```
#  OBJ file for a 2x2x2  cube
v  -1.0 1.0 1.0          - vertex 1
v  -1.0 -1.0 1.0         - vertex 2
v  1.0 -1.0 1.0          - vertex 3
v  1.0 1.0 1.0           -…
v  -1.0 1.0 -1.0
v  -1.0 -1.0 -1.0
v  1.0 -1.0 -1.0
v  1.0 1.0 -1.0
f  1  2  3  4
f  8  7  6  5
f  4  3  7  8
f  5  1  4  8
f  5  6  2  1
f  2  6  7  3
```



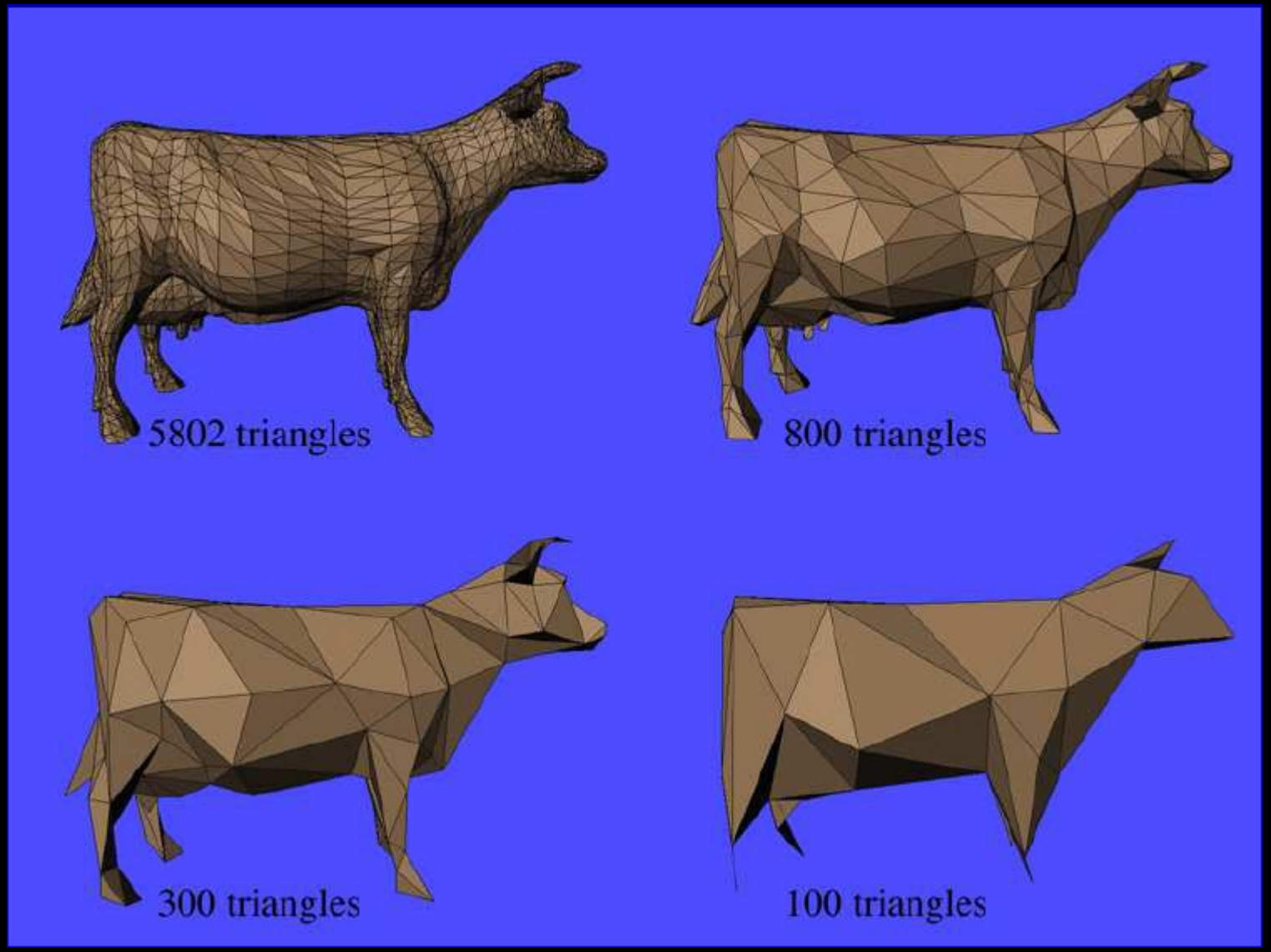
Syntax:

v *x y z* — *a vertex at (x,y,z)*

f v_1 v_2 … v_n — *a face with vertices v_1, v_2, … v_n*

*anything* — *comment*

How Many Polygons to Use?



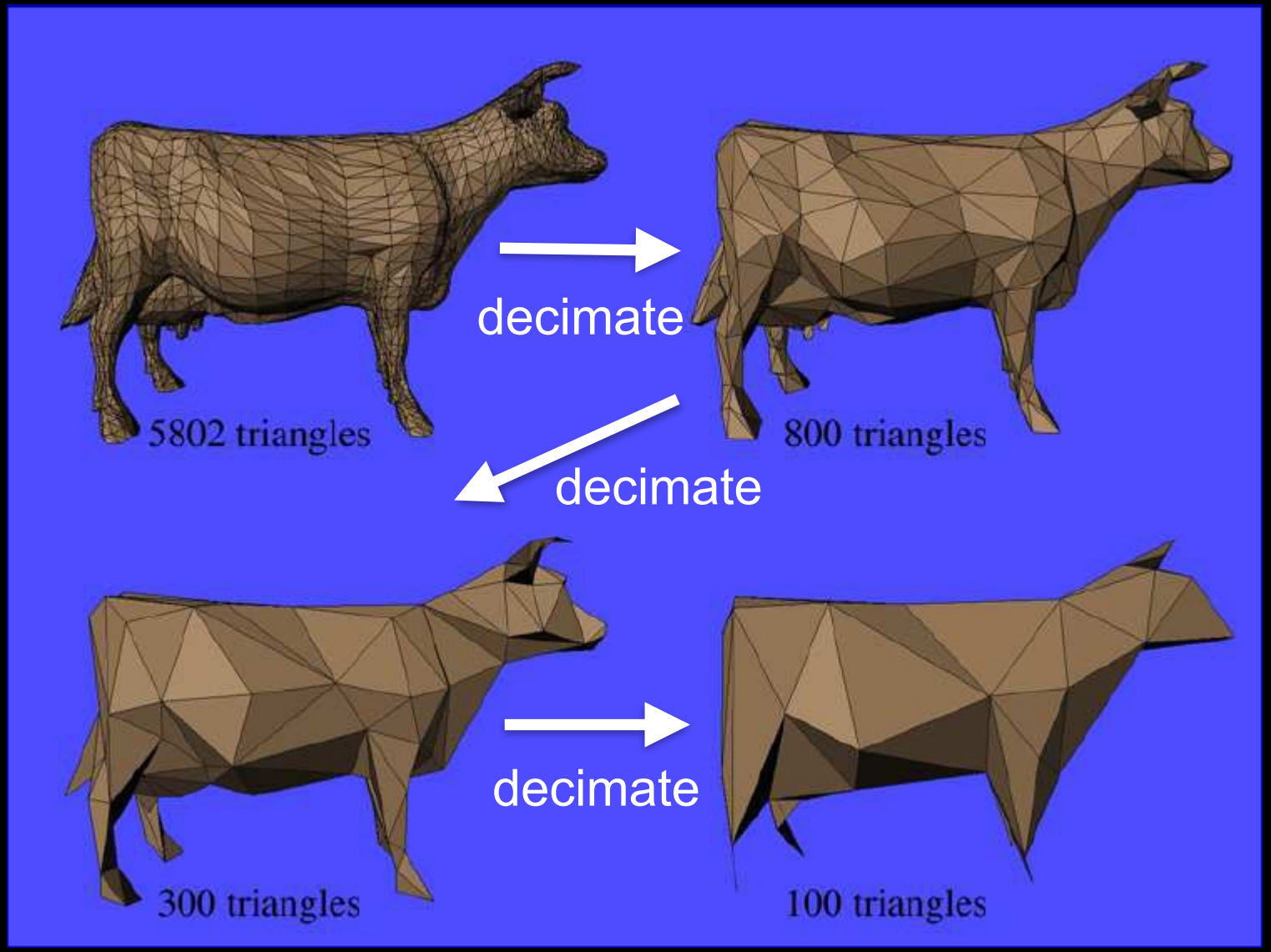
Why Level of Detail?

- Different models for near and far objects
- Different models for rendering and collision detection
- Compression of data recorded from the real world

We need automatic algorithms for reducing the polygon count without

- losing key features
- getting artifacts in the silhouette
- popping

Decimation algorithms



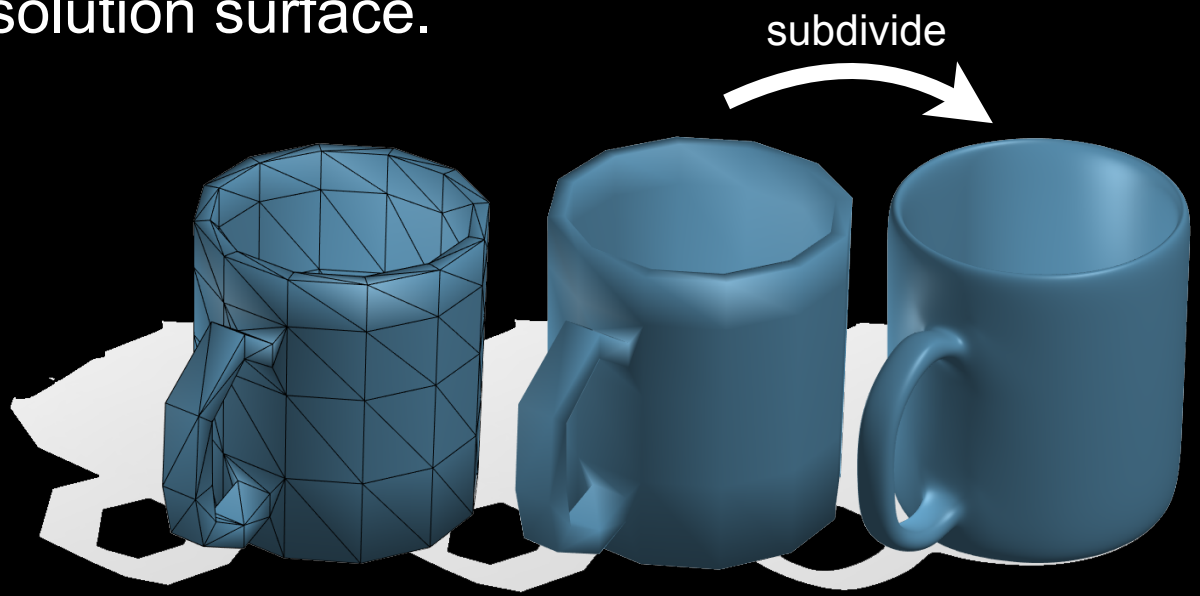
qslim

LoD: dynamic level of detail

- Implementing LoD in an engine:
  - Precompute different meshes for different resolutions.
  - Use heuristic to determine how large the object will be on screen.
  - Load the correct instance of the object to render.
- Where should this dynamic loading happen?
  - CPU?
  - Vertex shader?
  - Fragment shader?
  - Tessellation/geometry shader?

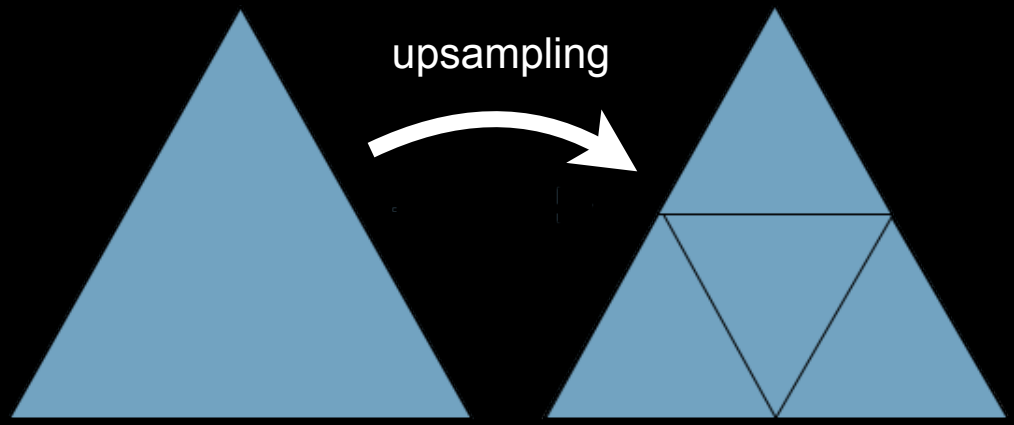
Subdivision

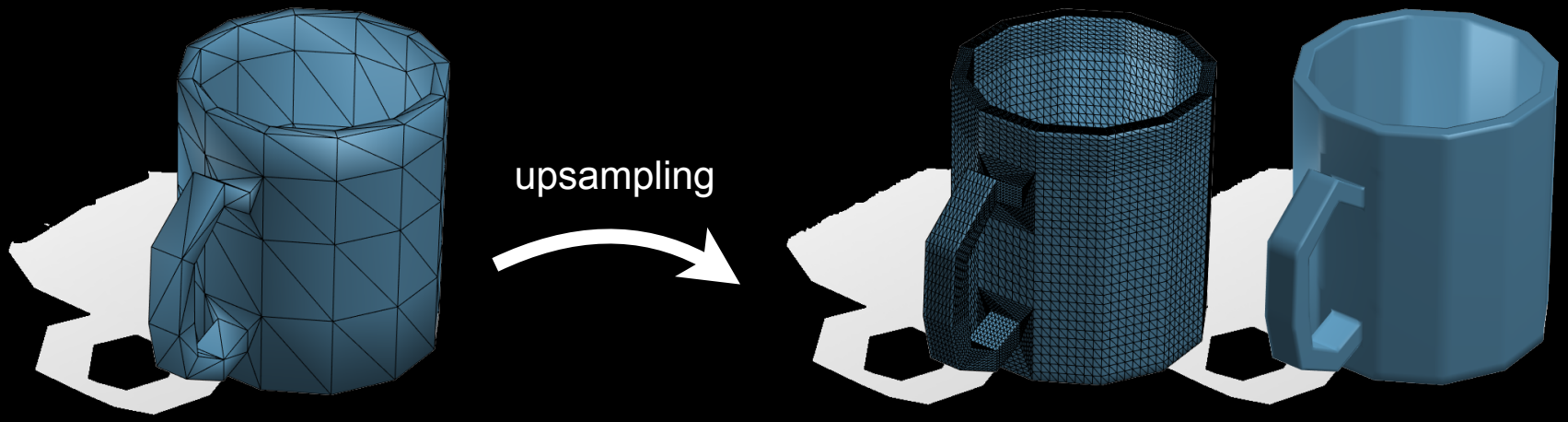
- There is only so much information in a simple shape.
- To display a smooth shape, we need lots of polygons.
- Why should we have to store lots of polygons?
  - Only store necessary information
  - Simple mathematical formula to get from small number of stored information to detailed high-resolution surface.



Subdivision

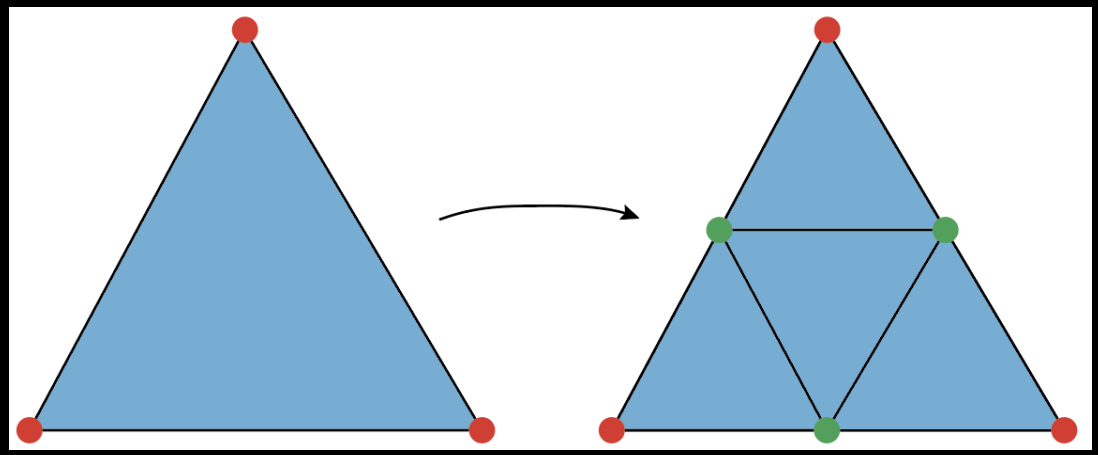
- It's not enough to simple replace each triangle with more triangles.





Subdivision

- After each upsampling step we need to move the vertices to make the surface smooth



- Subdivision rules tell us where to place new vertices (green) and where to move old vertices (red).
  - For triangles: Loop subdivision rules
  - For quadrilaterals: Catmull-clark subdivision rules

Subdivision

- Where is this implemented in our system?
  - CPU?
  - Vertex shader?
  - Fragment shader?
  - Tessellation/geometry? shader?

Problems with Triangular Meshes?

- Need a lot of polygons to represent smooth shapes
- Need a lot of polygons to represent detailed shapes

- Hard to edit
- Need to move individual vertices
- Intersection test? Inside/outside test?

Shape Representations

Polygon Meshes
Parametric Surfaces
Implicit Surfaces

Parametric Surfaces

$$p(u,v) = [x(u,v), y(u,v), z(u,v)]$$

- Need to do some math to get from parameters to the actual surface

Parametric Surfaces

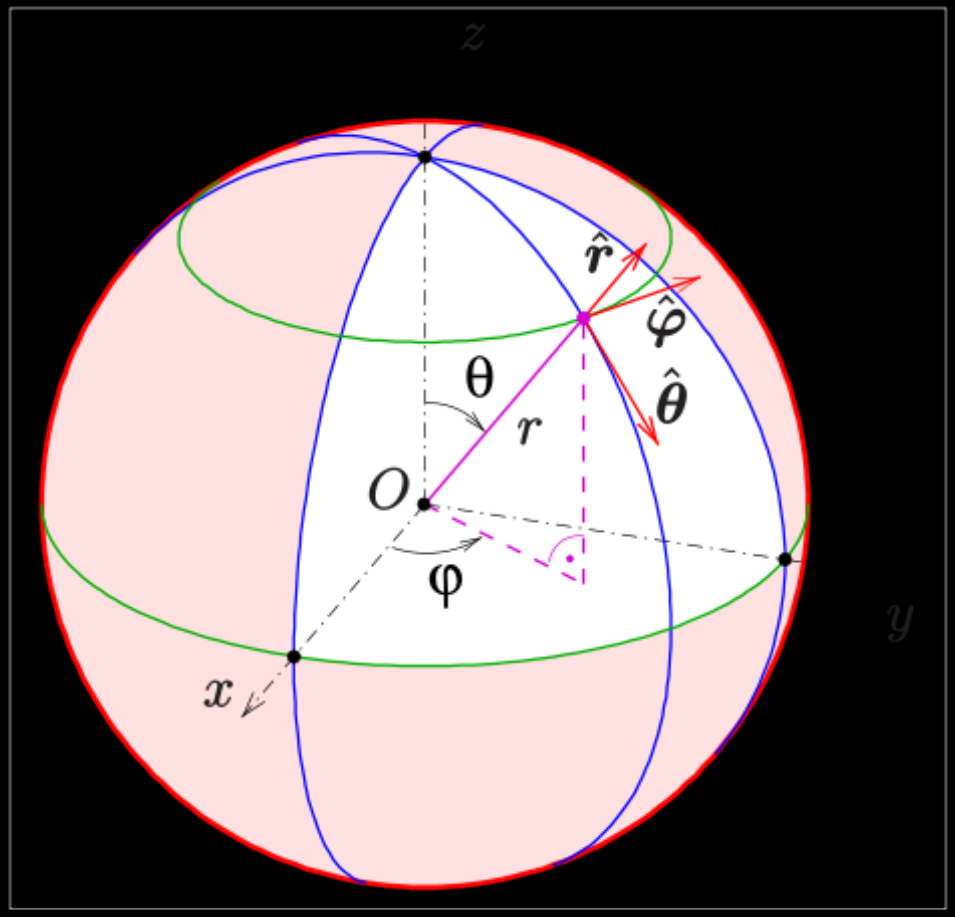
p(u,v) = [x(u,v), y(u,v), z(u,v)]

- Spherical coordinate system

$$x = r \sin\theta \cos\varphi$$

$$y = r \sin\theta \sin\varphi$$

$$z = r \cos\theta$$

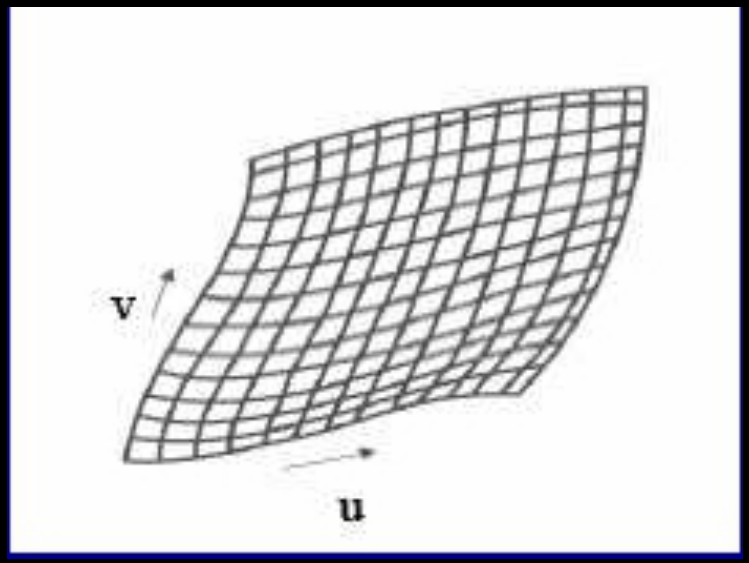


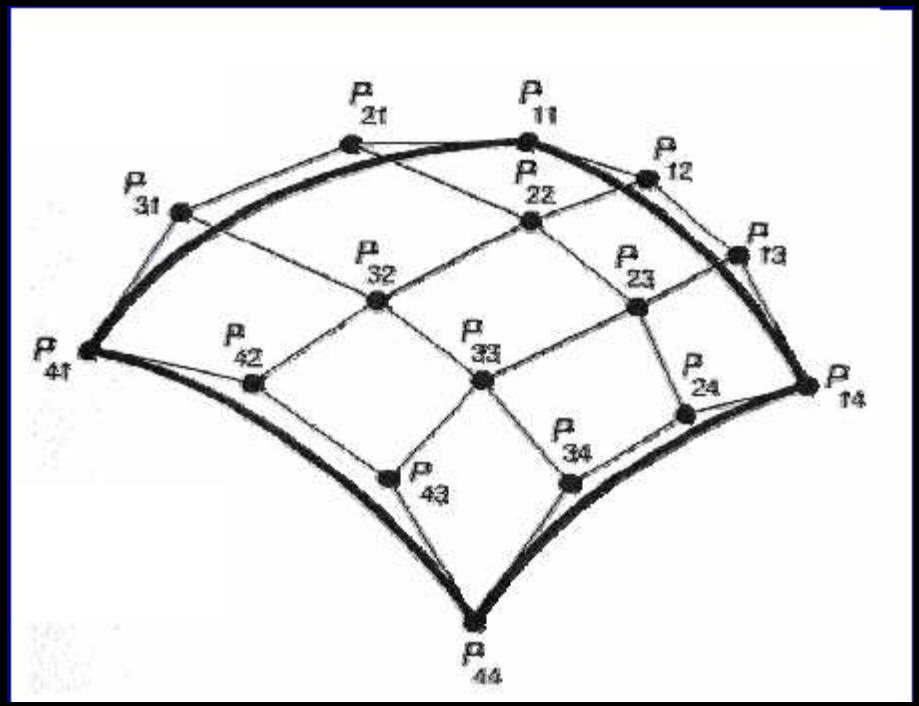
https://commons.wikimedia.org/wiki/File:Kugelkoord-lokb-e.svg

Parametric Surfaces

$$p(u,v) = [x(u,v), y(u,v), z(u,v)]$$

– e.g. plane, cylinder, bicubic surface, swept surface



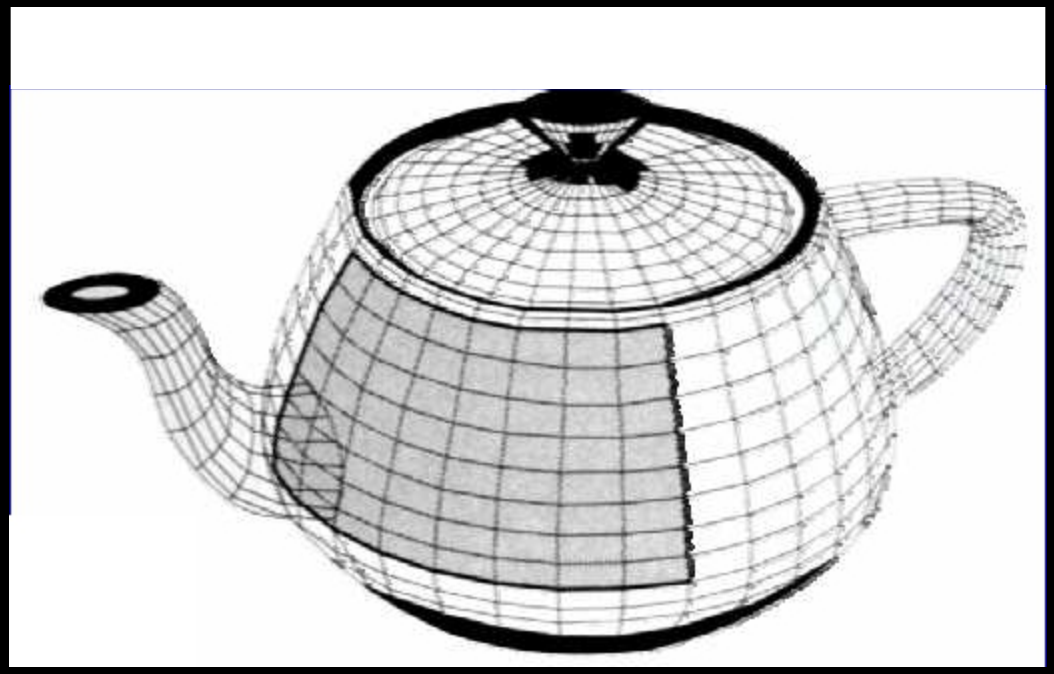


Bezier patch

Parametric Surfaces

$$p(u,v) = [x(u,v), y(u,v), z(u,v)]$$

– e.g. plane, cylinder, bicubic surface, swept surface



the Utah teapot

Parametric Surfaces

Why better than polygon meshes?

- Much more compact
- More convenient to control --- just edit control points
- Easy to construct from control points

What are the problems?

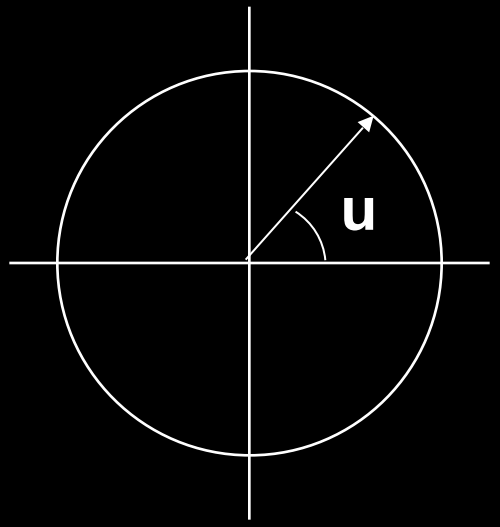
- Work well for smooth surfaces
- Must still split surfaces into discrete number of patches
  - Difficult transition between patches
- Rendering times are higher than for polygons
  - Evaluate mathematical formula every time
- Intersection test? Inside/outside test?

Shape Representations

Polygon Meshes
Parametric Surfaces
Implicit Surfaces

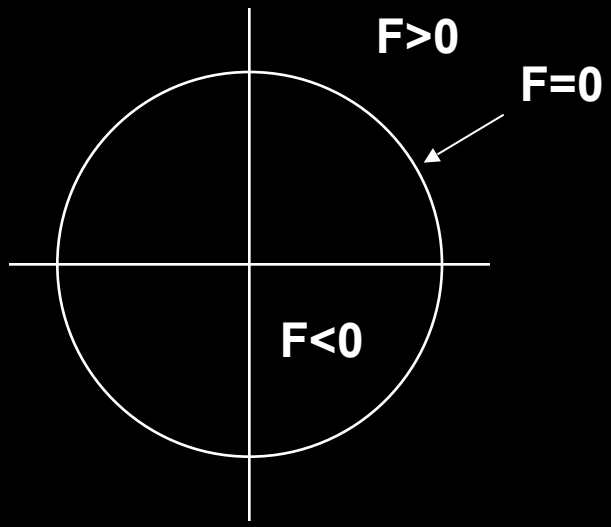
Two Ways to Define a Circle

Parametric



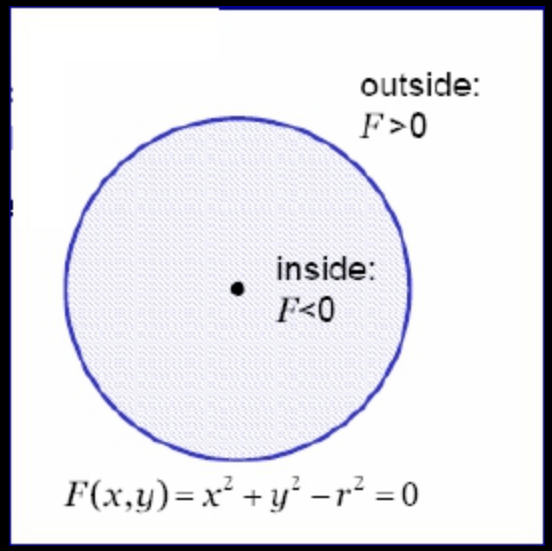
$x = f(u) = r \cos(u)$
$y = g(u) = r \sin(u)$

Implicit



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

Implicit Surfaces



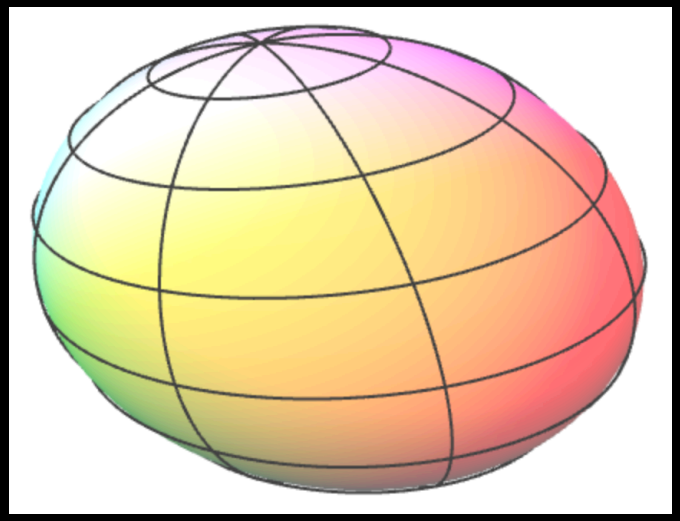
- well defined inside/outside
- polygons and parametric surfaces do not have this information

- Computing is hard:
  - implicit functions for a cube?
  - telephone?

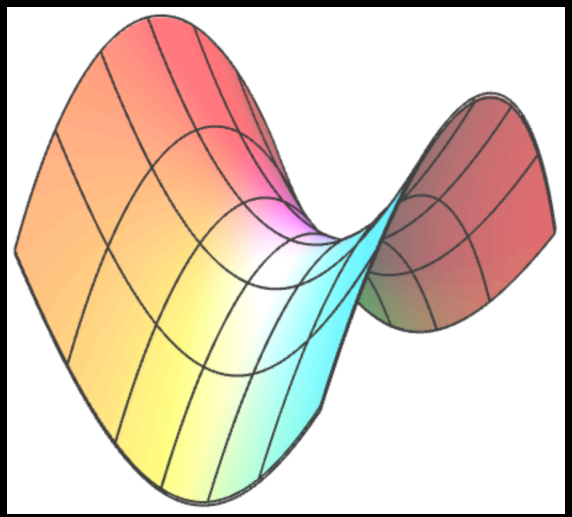
- Implicit surface: F(x,y,z) = 0
  - e.g. plane, sphere, cylinder, quadric, torus, blobby models
    sphere with radius *r* : $F(x,y,z) = x^2+y^2+z^2-r^2 = 0$
  - terrible for iterating over the surface
  - great for intersections, inside/outside test

Quadric Surfaces

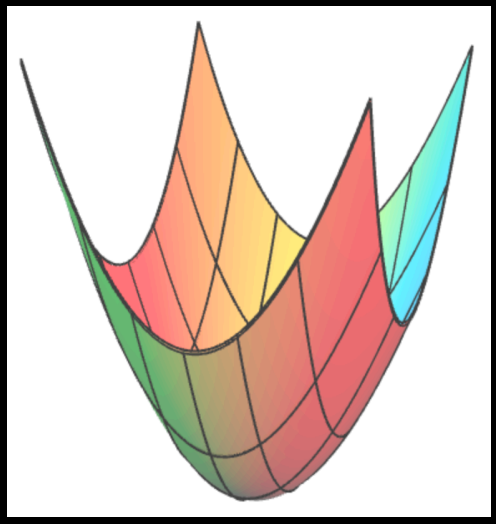
$F(x,y,z) = ax^2+by^2+cz^2+2fyz+2gzx+2hxy+2px+2qy+2rz+d = 0$



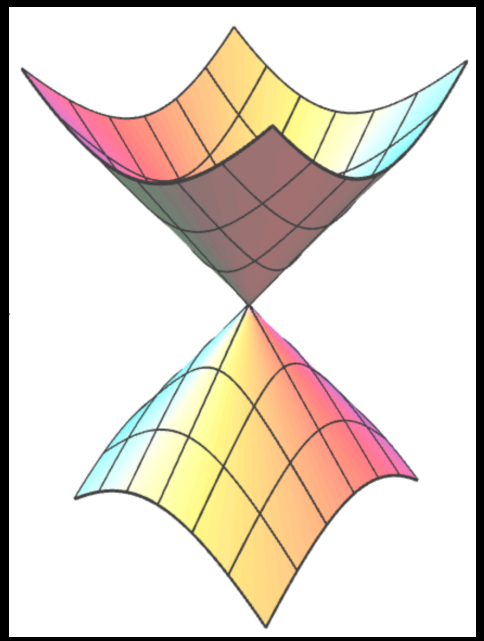
ellipsoid



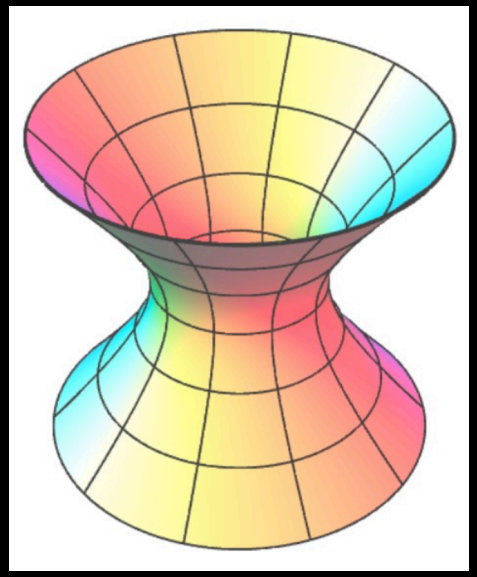
hyperbolic paraboloid



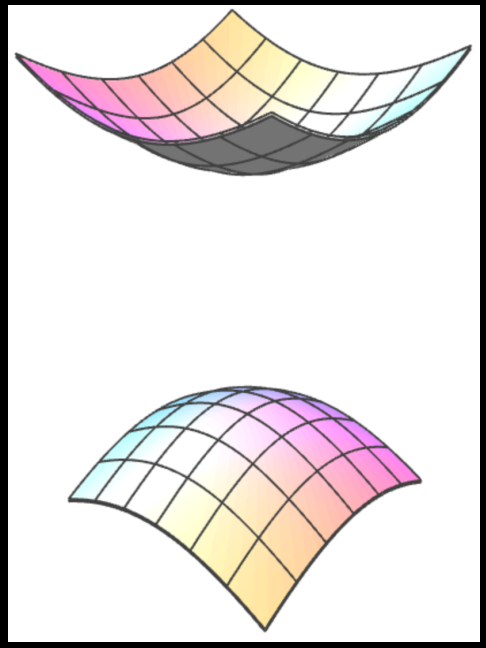
elliptic paraboloid



double cone

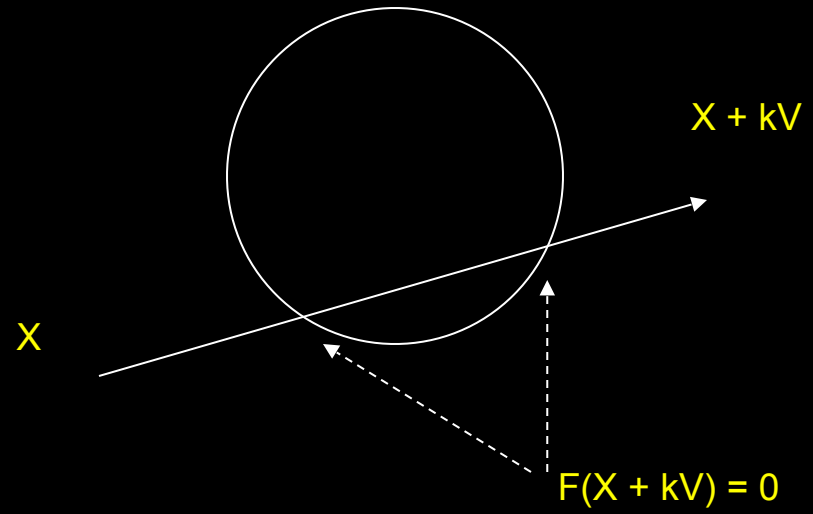


hyperboloid of one sheet

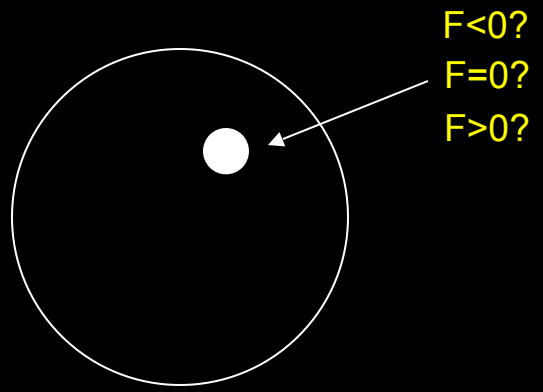


hyperboloid of two sheets

What Implicit Functions are Good For



Ray - Surface Intersection Test



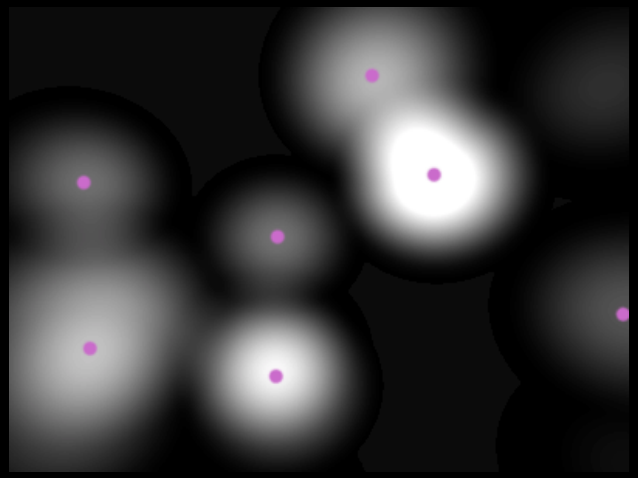
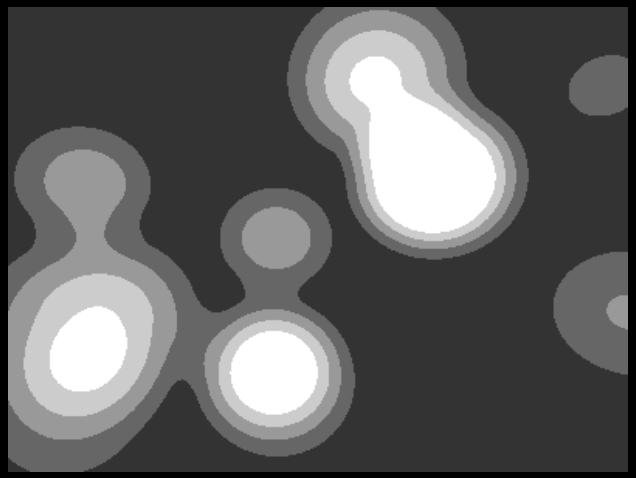
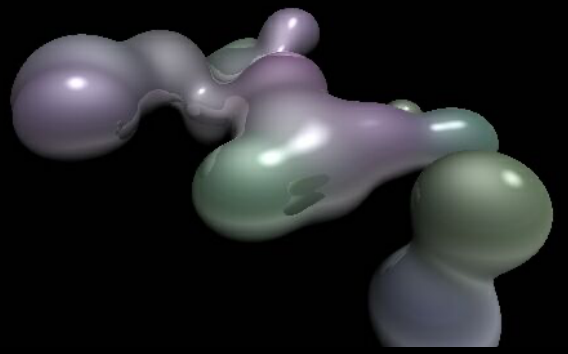
Inside/Outside Test

Surfaces from Implicit Functions

- Constant Value Surfaces are called (depending on whom you ask):
  - constant value surfaces
  - level sets
  - isosurfaces **(if you ask me)**

- Nice Feature: you can add them! (and other tricks)
  - this merges the shapes
  - When you use this with exponential potentials e^{-x^2}, it's called *Blobs*, *Metaballs,* or *Soft Objects*. Great for modeling animals.

Surfaces from Implicit Functions

http://www.geisswerks.com/ryan/BLOBS/blobs.html

Blobby Models

Source: blender.org (2017)

How to draw implicit surfaces?

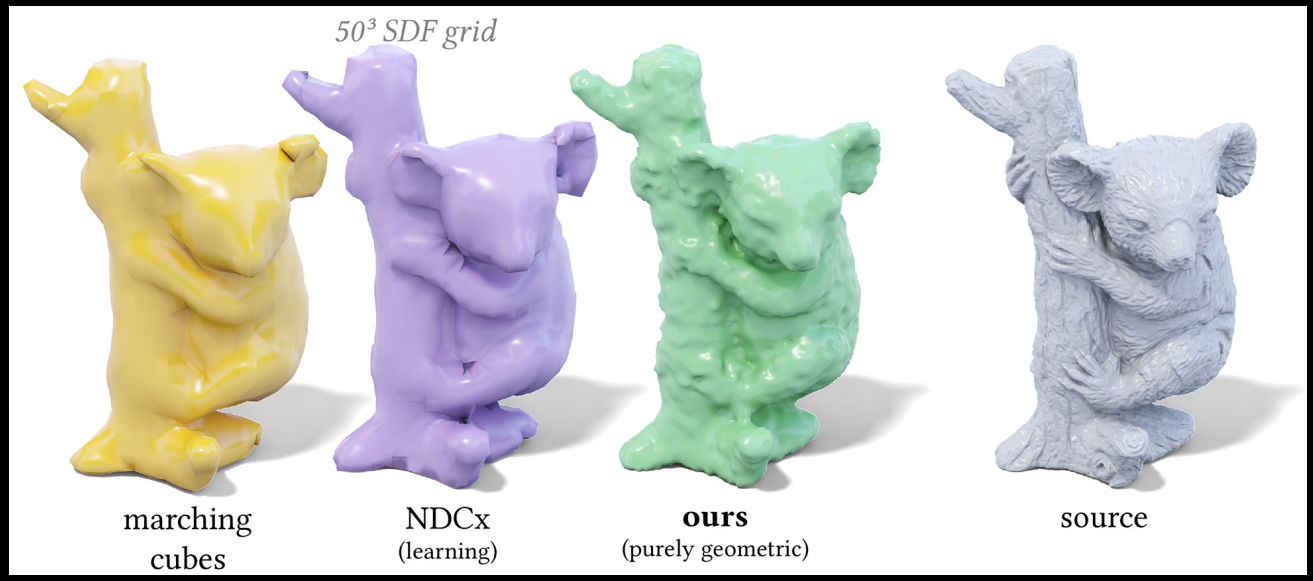
- It's easy to ray trace implicit surfaces
  - because of that easy intersection test
- Volume Rendering can display them
  - But triangle-based OpenGL has a hard time.
- Convert to polygons: the Marching Cubes algorithm
  - Divide space into cubes
  - Evaluate implicit function at each cube vertex
  - Do root finding or linear interpolation along each edge
  - Polygonize on a cube-by-cube basis

Signed distance functions

- Normal implicit surface $f(\mathbf{p}) = 0$ is the surface
- If $\mathbf{p} \neq 0$, no further opinion (except sign)
- Why are we not using all the information contained in all the other points in space?
- Signed distance function
  - $|f(\mathbf{p})|$ is the distance from the surface
  - $f(\mathbf{p}) < 0$ inside the surface, $f(\mathbf{p}) > 0$ outside
  - $f(\mathbf{p}) = 0$ is the surface

Signed distance functions

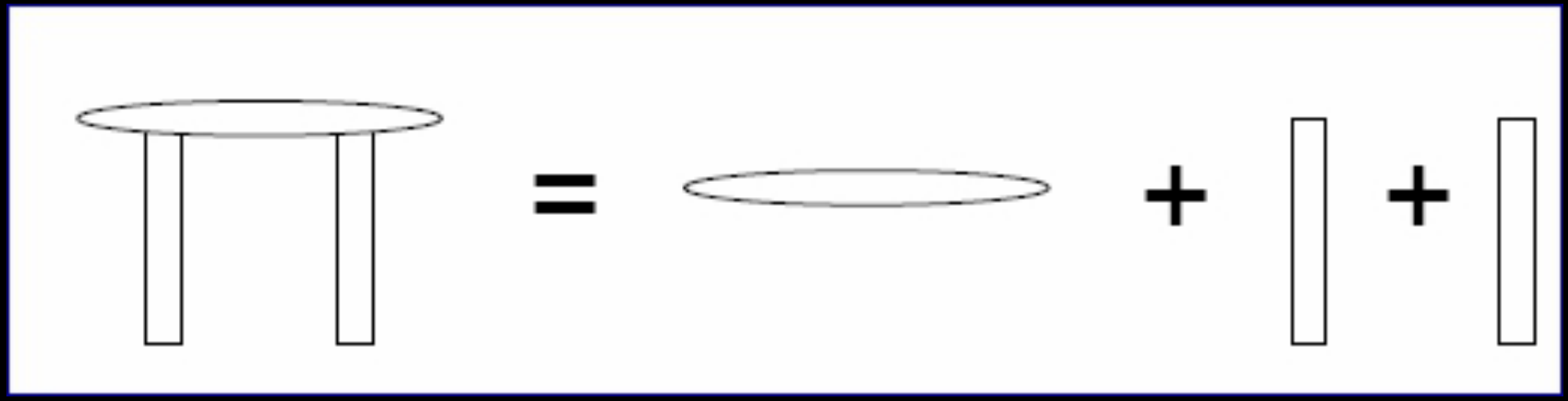
- Very useful for some kinds of simulation
  - Convenient access to normal vector
  - Convenient access to distance
- Can recover more detail from less data



https://odedstein.com/projects/reach-for-the-spheres/

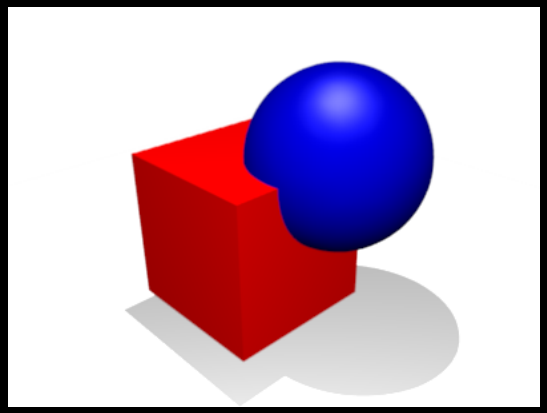
Constructive Solid Geometry (CSG)

- Generate complex shapes with basic building blocks
- Machine an object - saw parts off,
  drill holes, glue pieces together



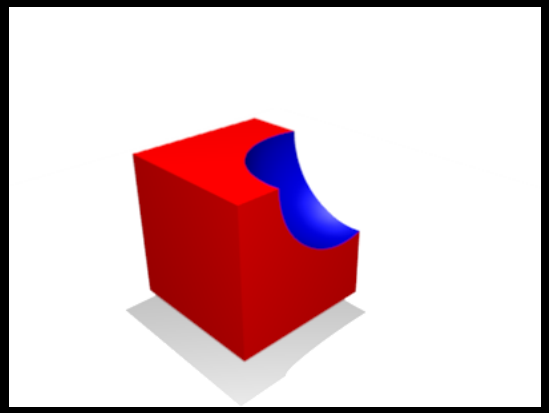
Constructive Solid Geometry (CSG)

union



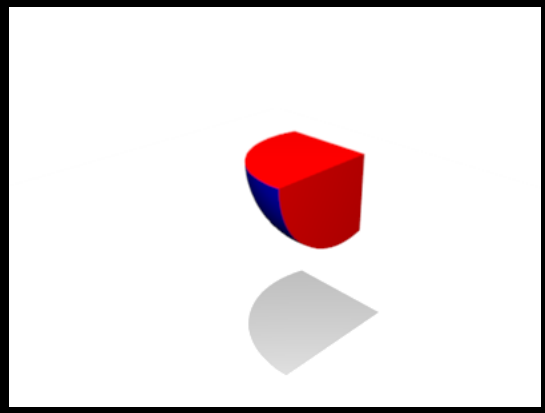
the merger
of two objects
into one

difference



the subtraction
of one object
from another

intersection



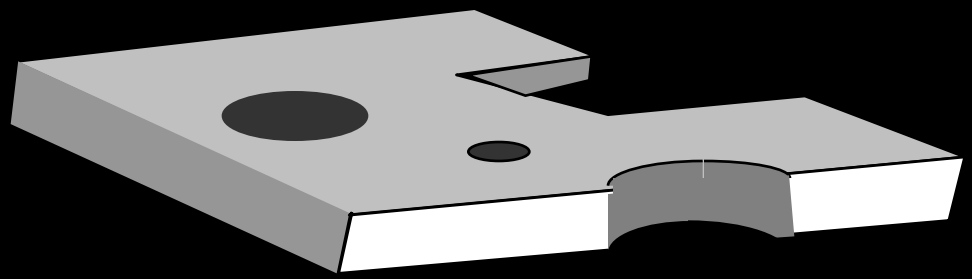
Source: Wikipedia

the portion
common to
both objects

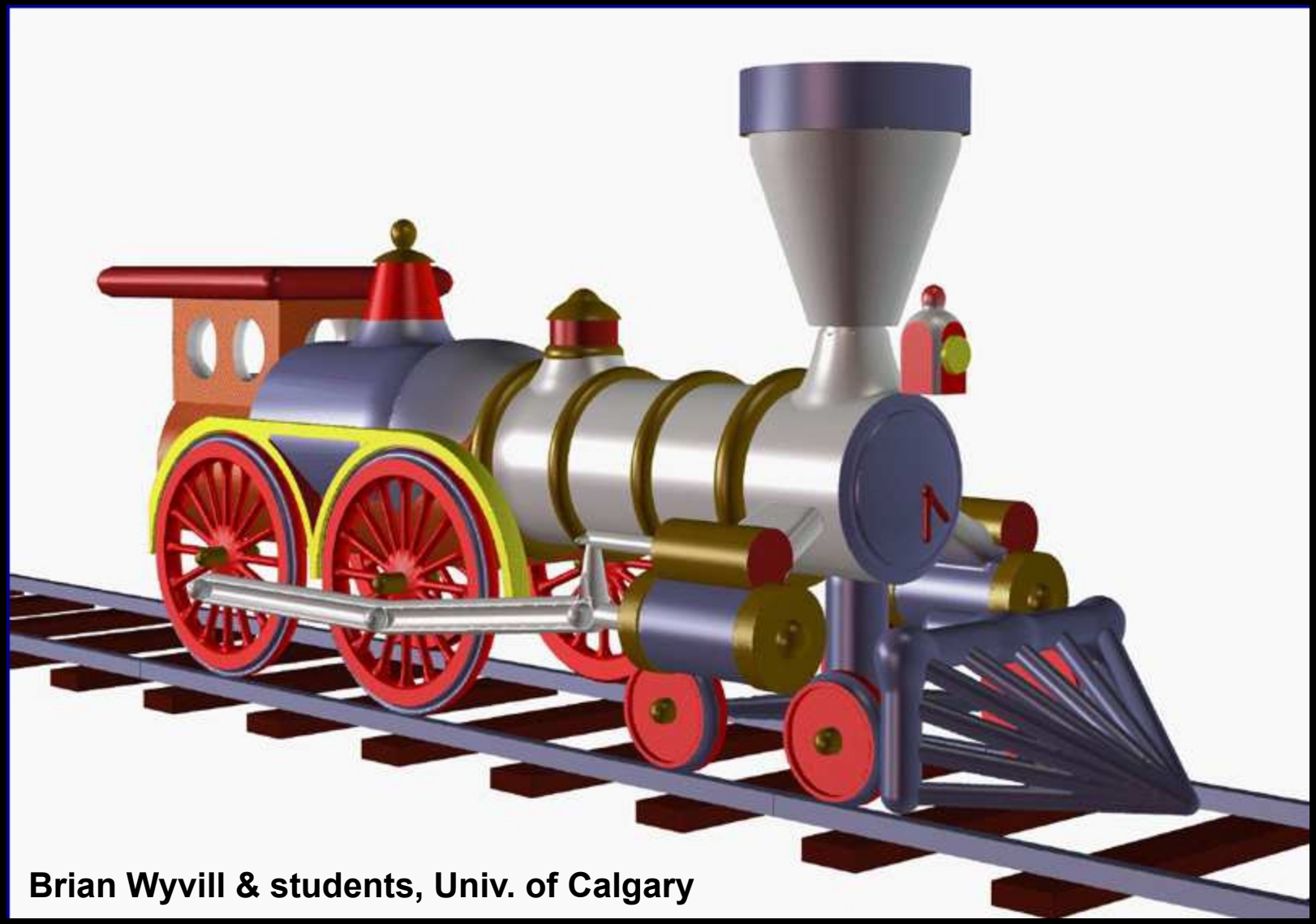
Constructive Solid Geometry (CSG)

- Generate complex shapes with basic building blocks
- Machine an object - saw parts off,
  drill holes, glue pieces together

- This is sensible for objects that are actually made
  that way (human-made, particularly machined objects)



A CSG Train



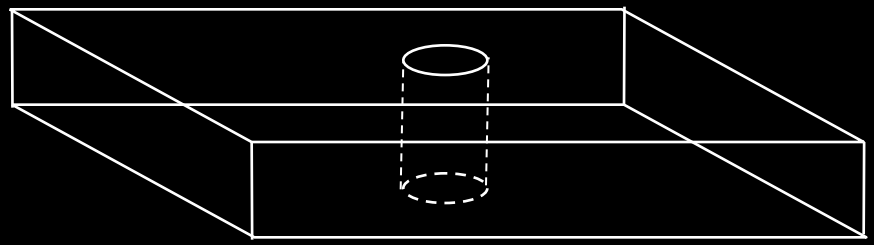
Brian Wyvill & students, Univ. of Calgary

Negative Objects

Use point-by-point boolean functions

- remove a volume by using a negative object
- e.g. drill a hole by subtracting a cylinder

Subtract  From 

To get 

Inside(BLOCK-CYL) = Inside(BLOCK) And Not(Inside(CYL))

Set Operations

- UNION: Inside(A) || Inside(B)
  - ➢ Join A and B
- INTERSECTION: Inside(A) && Inside(B)
  - ➢ Chop off any part of A that sticks out of B
- SUBTRACTION: Inside(A) && (! Inside(B))
  - ➢ Use B to Cut A

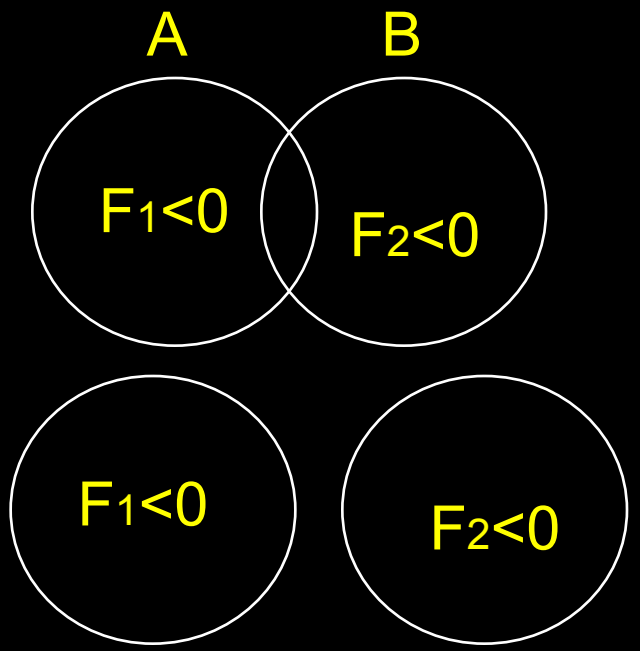
Examples:

- Use cylinders to drill holes
- Use rectangular blocks to cut slots
- Use half-spaces to cut planar faces
- Use surfaces swept from curves as jigsaws, etc.

Implicit Functions for Booleans

- Recall the implicit function for a solid: F(x,y,z)<0
- Boolean operations are replaced by arithmetic:
  - MAX replaces AND (intersection)
  - MIN replaces OR (union)
  - MINUS replaces NOT(unary subtraction)

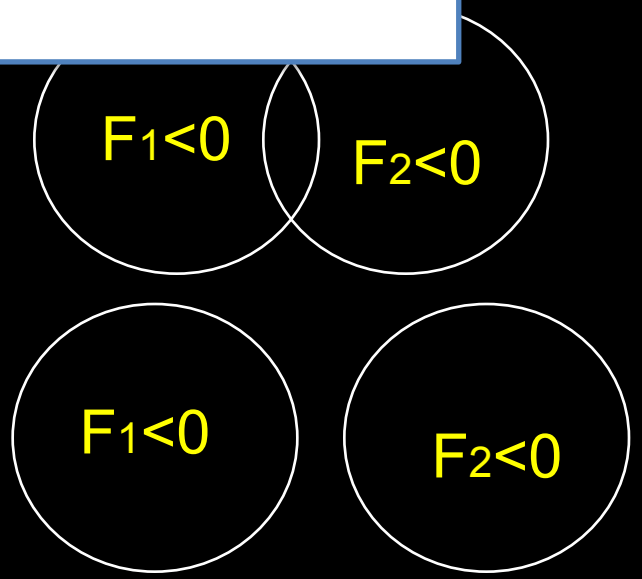
- Thus
  - F(Intersect(A,B)) = MAX(F(A),F(B))
  - F(Union(A,B)) = MIN(F(A),F(B))
  - F(Subtract(A,B)) = MAX(F(A), -F(B))



Implicit Functions for Booleans

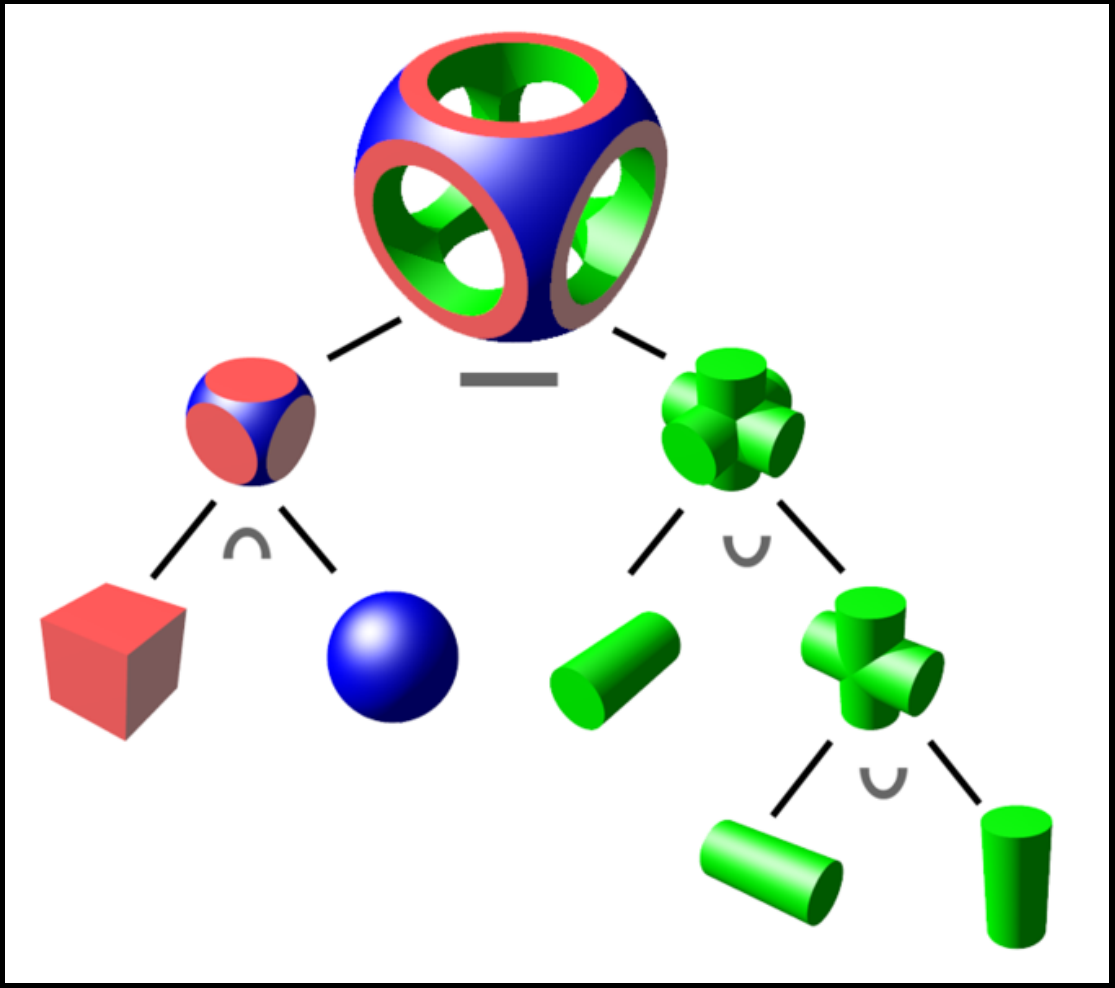
- Re
- Bo
  - –
  - –
  - –
- Thus
  - F(Intersect(A,B)) = MAX(F(A),F(B))
  - F(Union(A,B)) = MIN(F(A),F(B))
  - F(Subtract(A,B)) = MAX(F(A), -F(B))

What about signed distance functions?



CSG Trees

- Set operations yield tree-based representation



Source: Wikipedia

Implicit Surfaces

- Good for smoothly blending multiple components
- Clearly defined solid along with its boundary
- Intersection test and Inside/outside test are easy

- Need to polygonize to render --- expensive
- Interactive control is not easy
- Fitting to real world data is not easy
- Always smooth

Summary

- Polygonal Meshes
- Parametric Surfaces
- Implicit Surfaces
- Constructive Solid Geometry