CSE528 Computer Graphics: Theory, Algorithms, and Applications

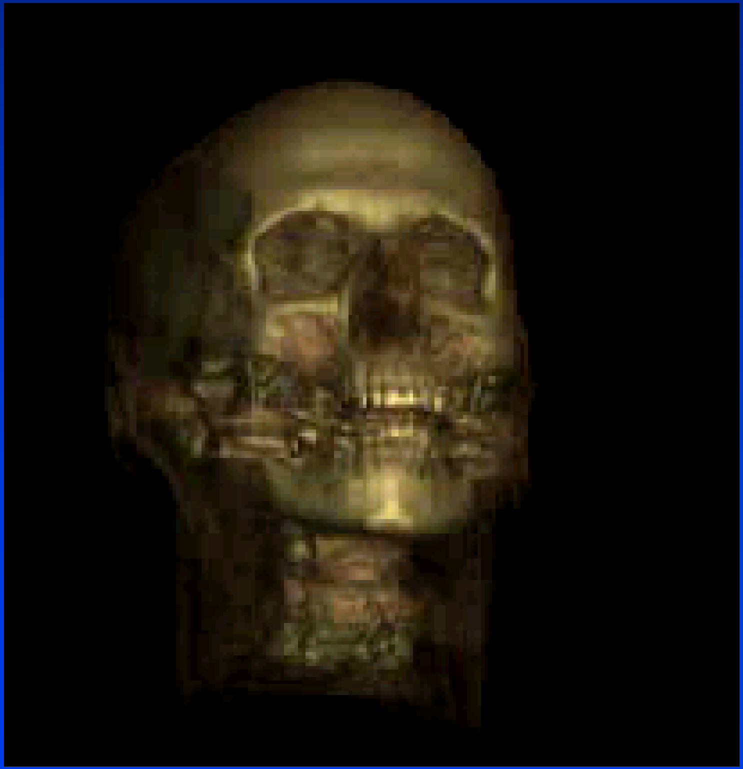
Hong Qin
Rm. 366 NEW CS Building
Department of Computer Science
Stony Brook University (SUNY at Stony Brook)
Stony Brook, New York 11794-2424
Tel: (631)632-8450; Fax: (631)632-8334
qin@cs.sunysb.edu or qin@cs.stonybrook.edu
http://www.cs.stonybrook.edu/~qin

Solid Modeling Basics

- Represent objects' solid interiors
  - Surface may not be described explicitly

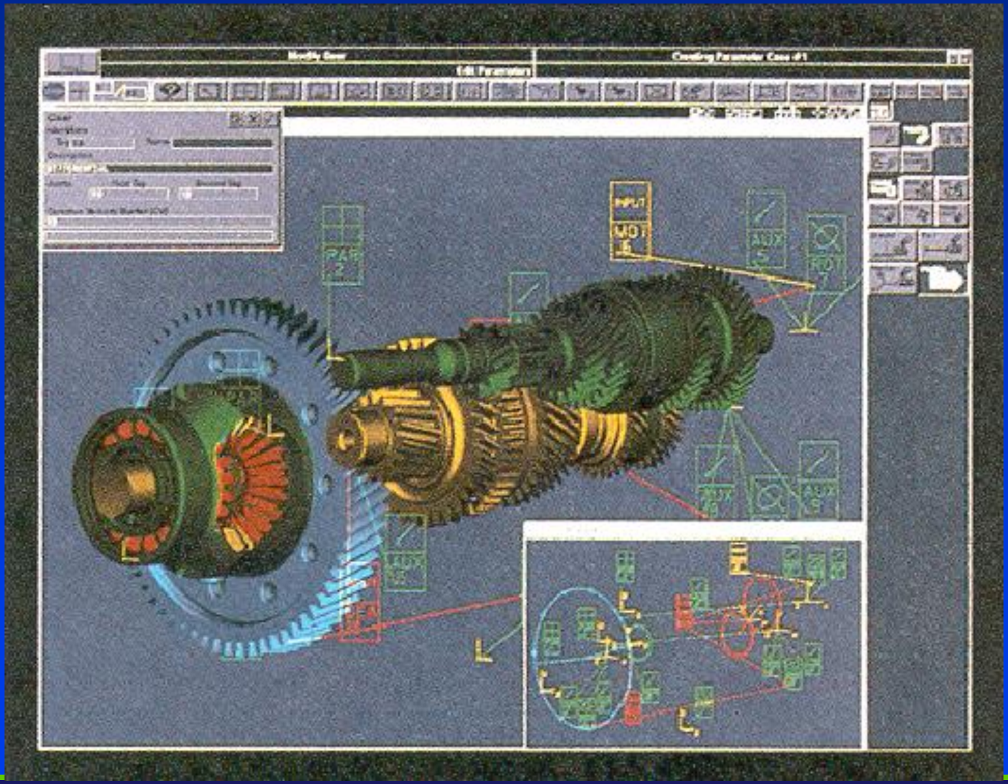
Motivation

- **Some acquisition methods to generate solids**
  - Example: Different medical imaging modalities



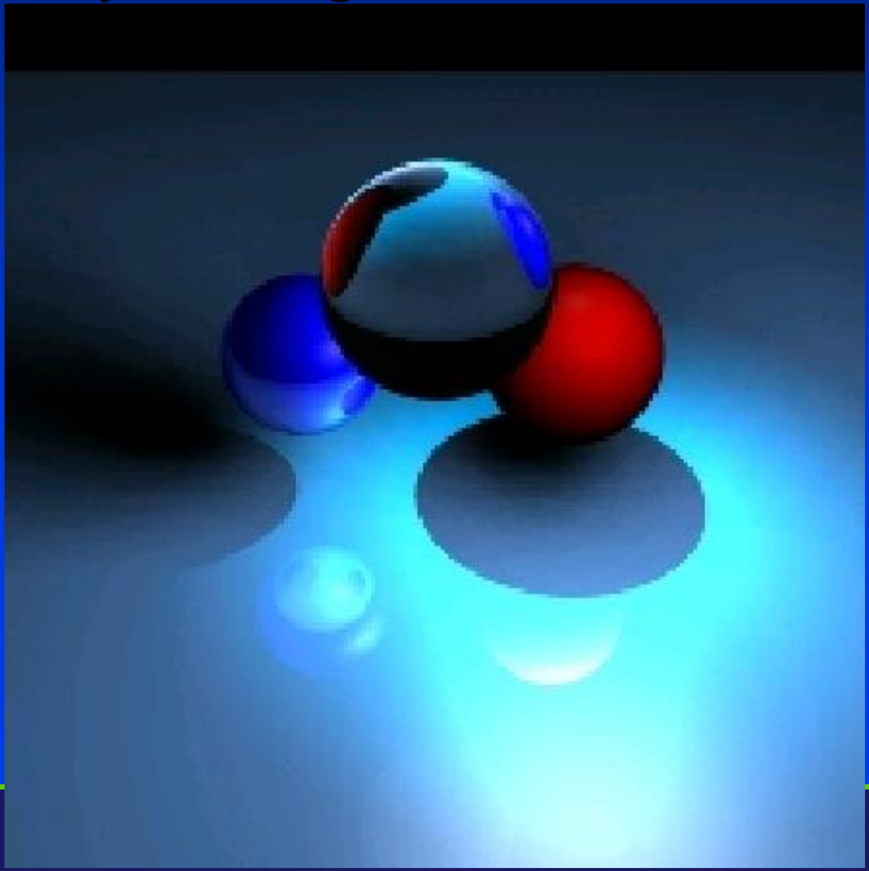
Motivation

- **Some applications to require solids**
  - Example: CAD/CAM/CAE



Motivation

- **Some algorithms to require solids**
  - Example: Ray tracing with refraction



Solid Modeling: A Brief History

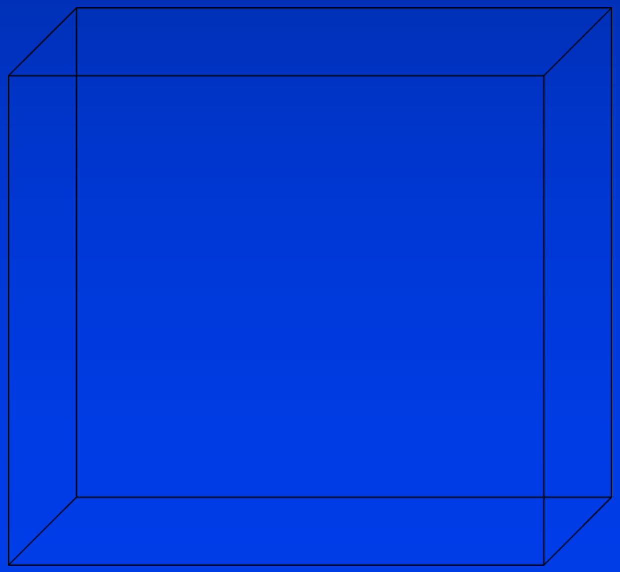
- CNC (Computer Numerical Control): ~1950
- Mainframe computers: ~1960's
- B-REP: 1970 (Baumgart)
- CSG: 1974 (Ian Braid)

Solid Modeling Representations

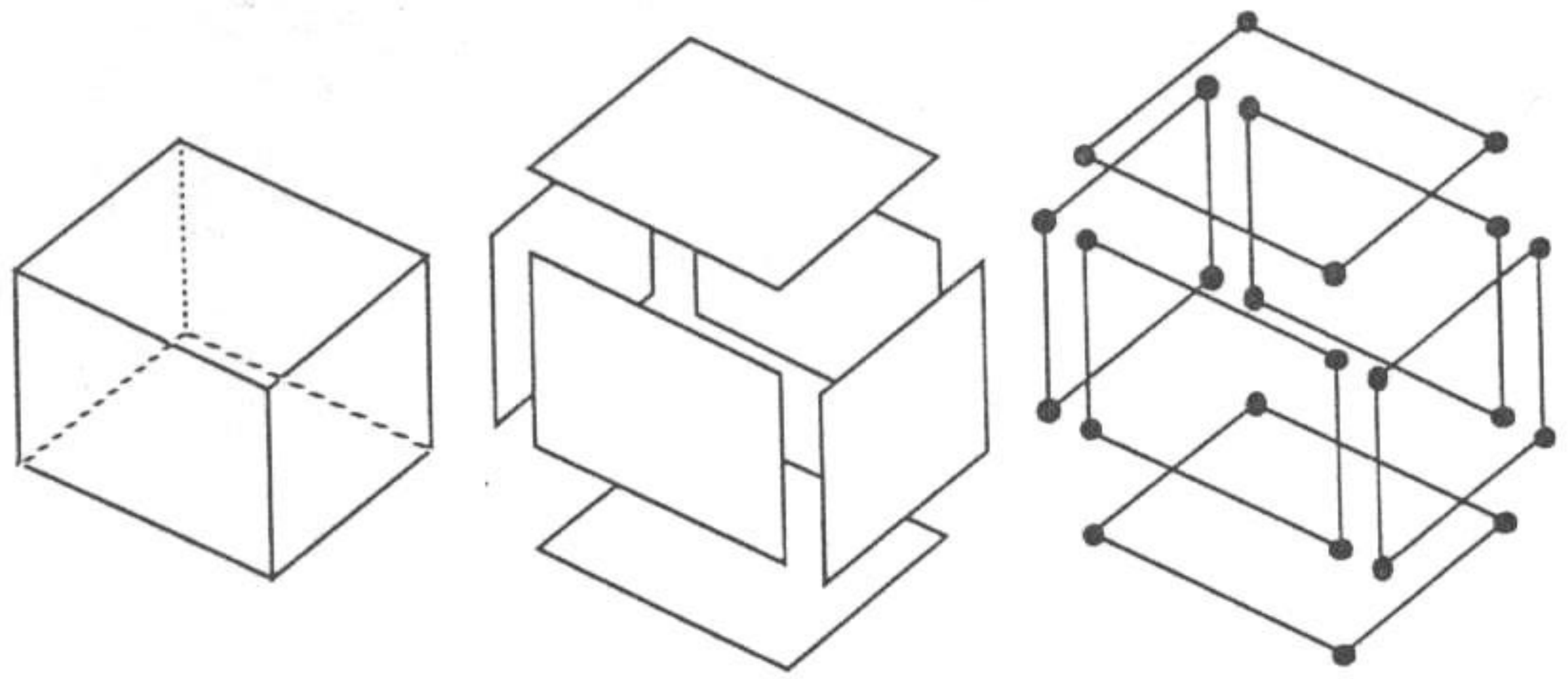
- Boundary representation (Surface representation)
- Constructive Solid Geometry (CAD/CAM/CAE)
- Voxels (Medical imaging modalities)
- Quadtrees & Octrees (Computational geometry)
- Binary Space Partitions (Computational geometry)

3D and Solid Representation

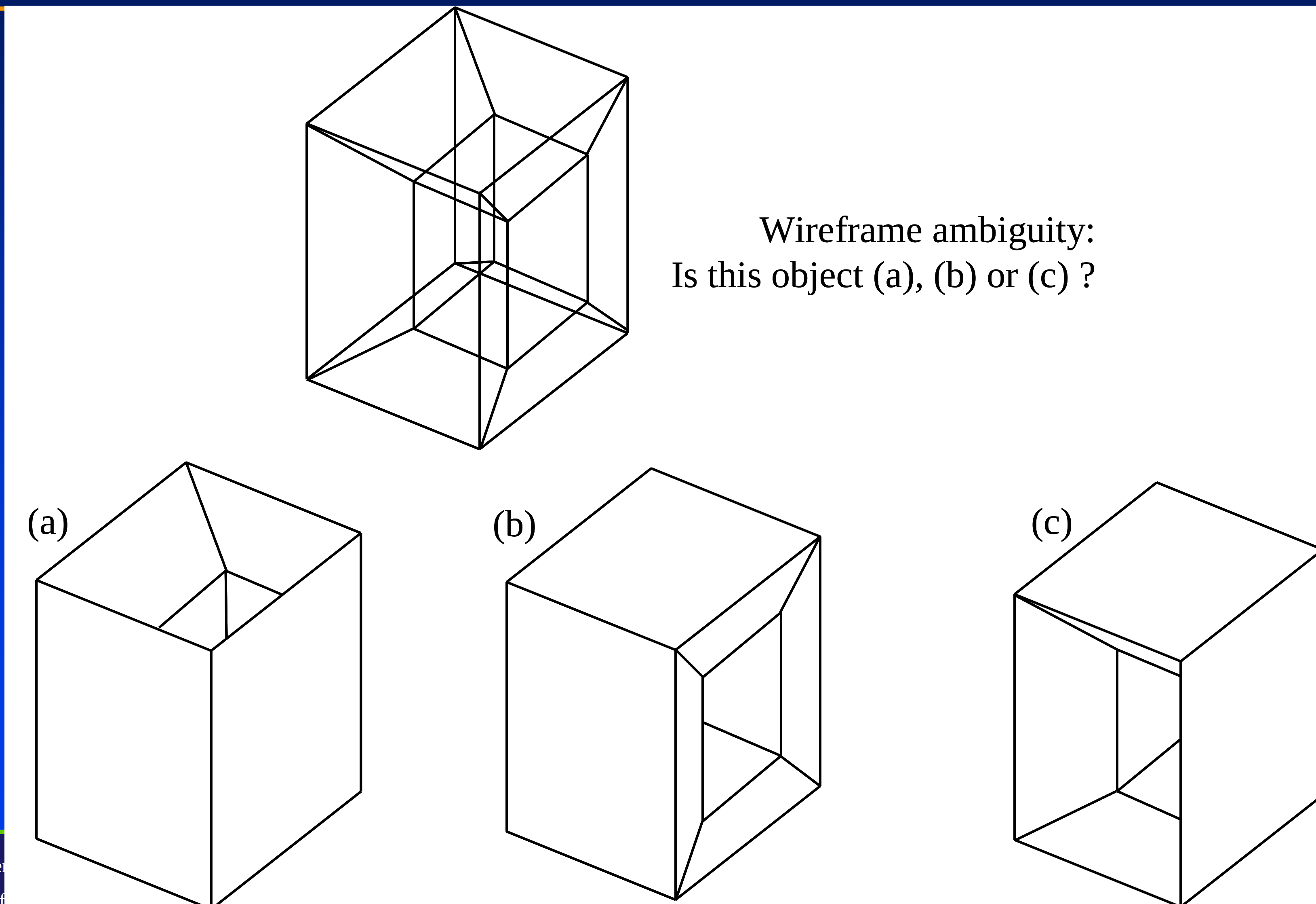
- Wireframe models
- Stores each edge of an object
- Data structure: the vertices (start point, end point)
- The equation of the edge-curve



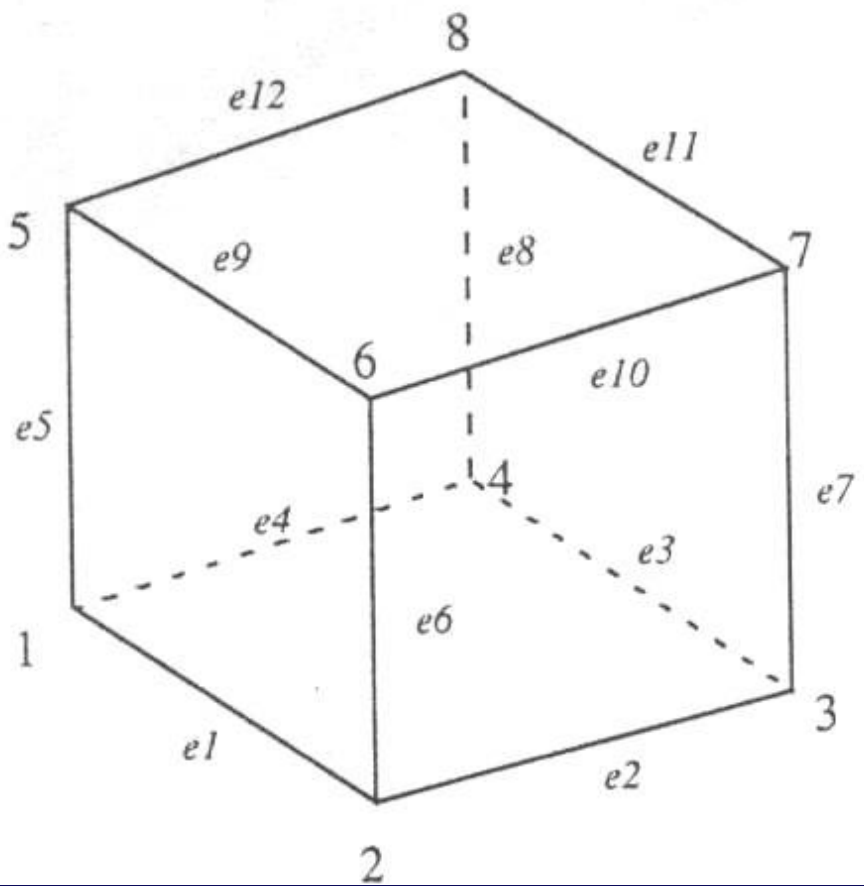
Boundary Models



Wireframe Problem: Ambiguity



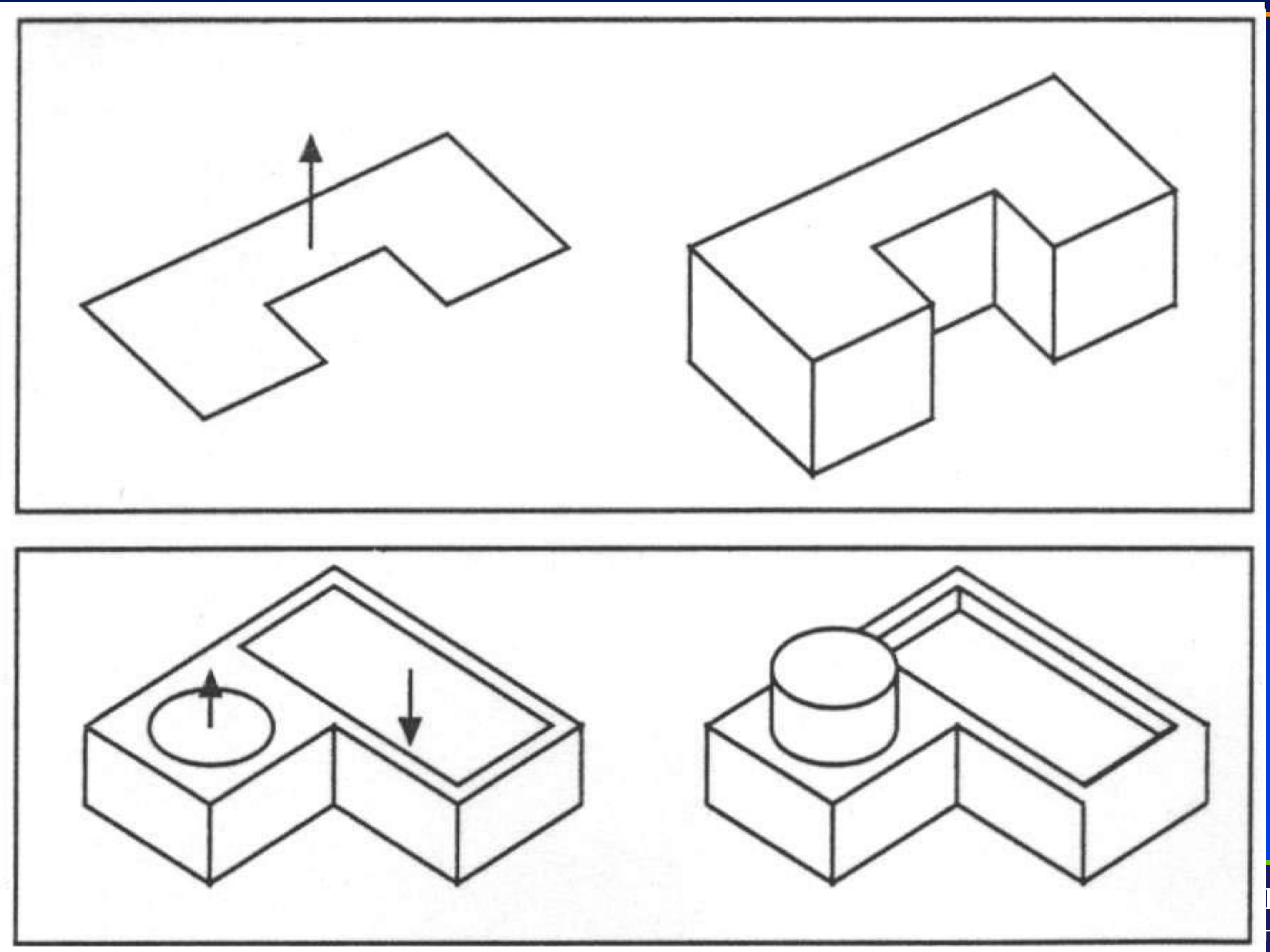
Vertex-Based B-REP



v1	x1	y1	z1
v2	x2	y2	z2
v3	x3	y3	z3
v4	x4	y4	z4
v5	x5	y5	z5
v6	x6	y6	z6
v7	x7	y7	z7
v8	x8	y8	z8

f1	v1	v2	v3	v4
f2	v6	v2	v1	v5
f3	v7	v3	v2	v6
f4	v8	v4	v3	v7
f5	v5	v1	v4	v8
f6	v8	v7	v6	v5

Procedural Models (Sweeping)



Popular Methods

- Constructive Solid Geometry (CSG)
- Boundary representation (B-REP)
- Spatial enumeration (voxels, octrees, etc.)
- Implicit representation

Solid Modeling: Fundamental Goals

- Problems of wireframe models: lack of robustness, incompleteness, limited applicability.
- Complete representation of solid objects that are adequate for answering any geometric questions (from robots) without help of human user.
- Two major issues: integrity and complexity

Solid Models

- Decomposition models
- Constructive models (CSG)
- Boundary models (B-rep)
- Non-manifold models

Properties of Solid Modeling

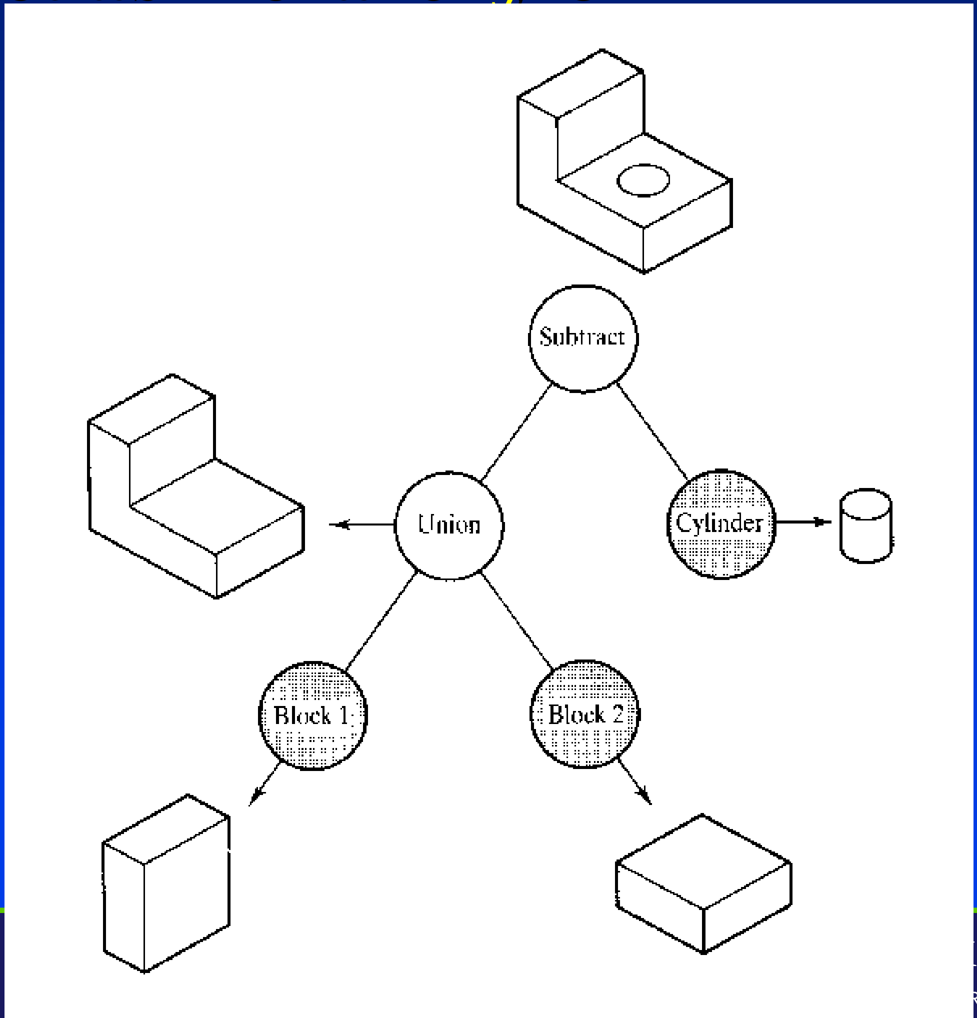
- Expressive power
- Validity: manufacturability
- Unambiguity and uniqueness
- Description languages: operations for construction
- Conciseness: storage requirement
- Computational ease and applicability: Computing power requirements

Solid Modeling Approaches

- Decomposition models: voxel, volume rendering, iso-surface extraction
- Constructive models: combination of primitives with set-theoretic operations: CSG
- Boundary models: in terms of its boundary: B-REP
- Non-manifold models: a hybrid of decomposition models and boundary models

Constructive Solid Geometry (CSG)

- **Represent solid object as hierarchy of Boolean operations**
  - Union
  - Intersection
  - Difference



Constructive Solid Geometry (CSG)

- Introduced by: Ian Braid (Cambridge University, ~74)

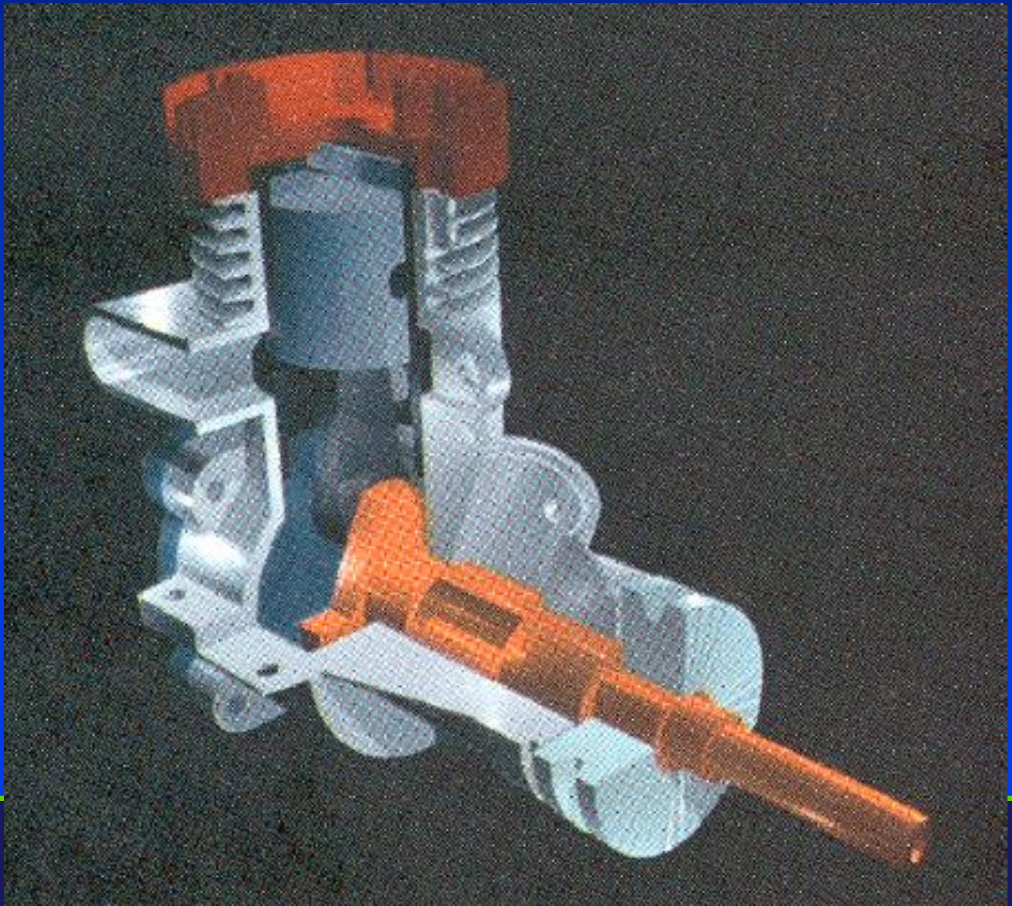
- Basic concepts: Combine simple primitives together using set operations (model construction using Boolean operations)
  - Union, Intersection, Subtraction (Difference)
- Intuitive operations for building more complex shapes

- Primitives: small set of shapes
- Transformations: Scaling, Rotation, Translation
- Set-theoretic Operations Union, Intersection, Difference

- Combinations of these → Solid Parts

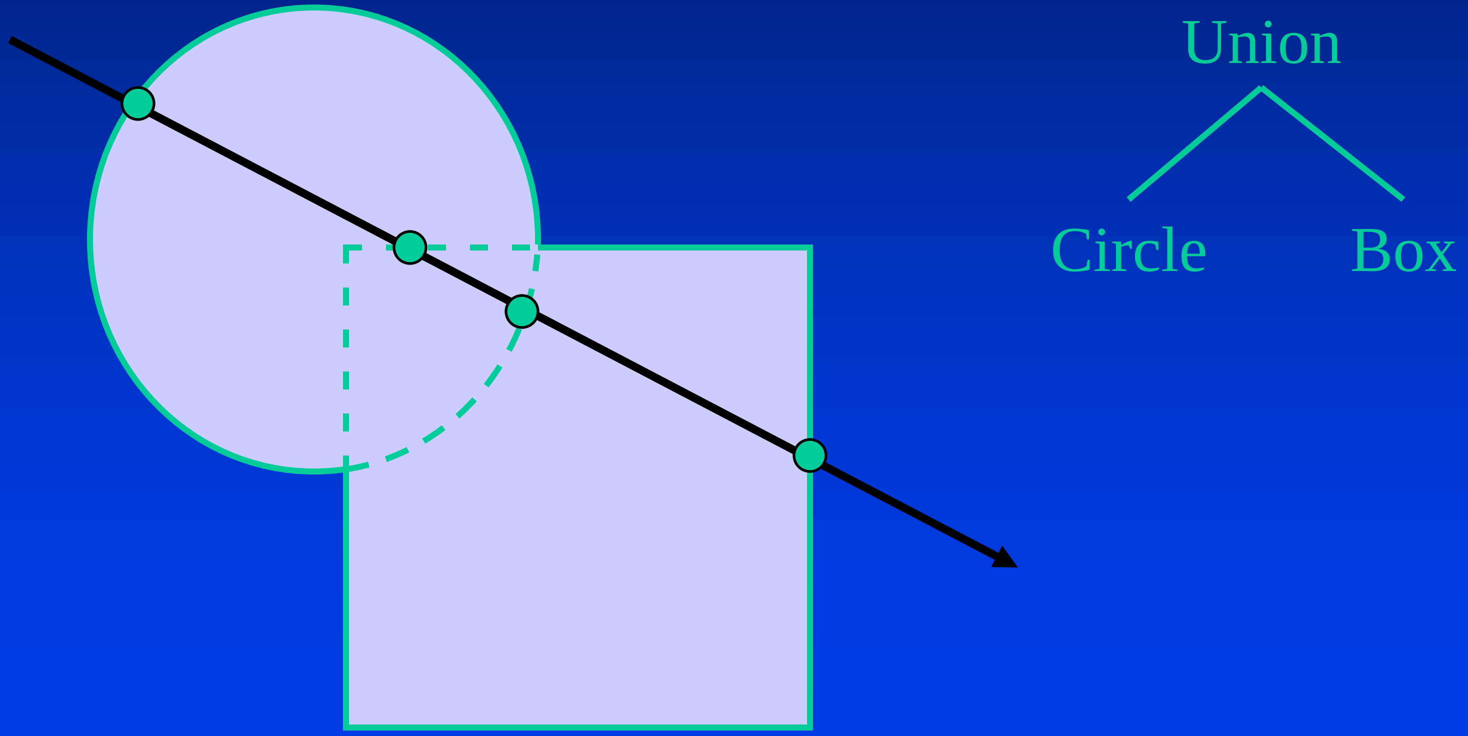
CSG Acquisition

- **Interactive modeling programs**
  - CAD/CAM/CAE

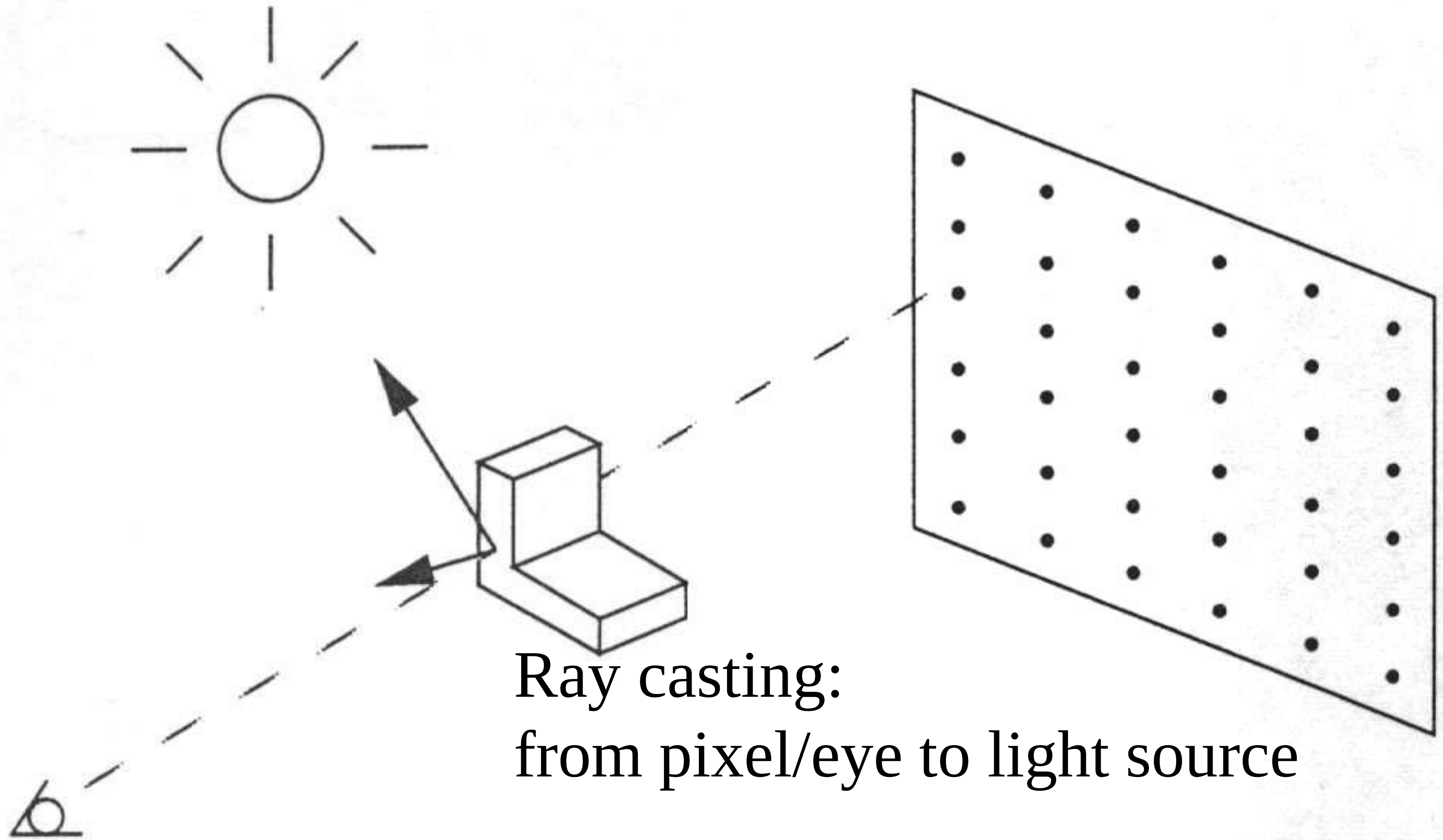


CSG Display & Analysis

- **Ray-casting**

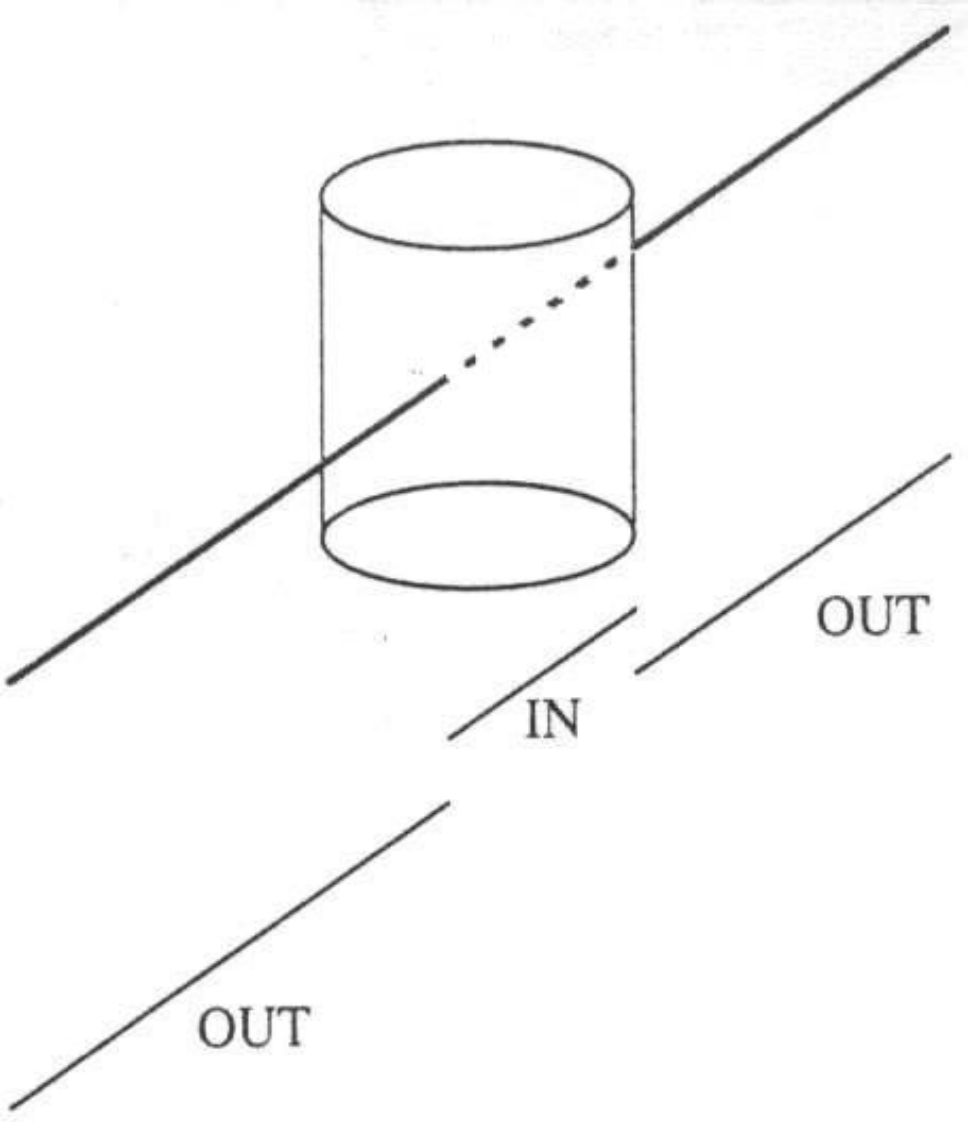


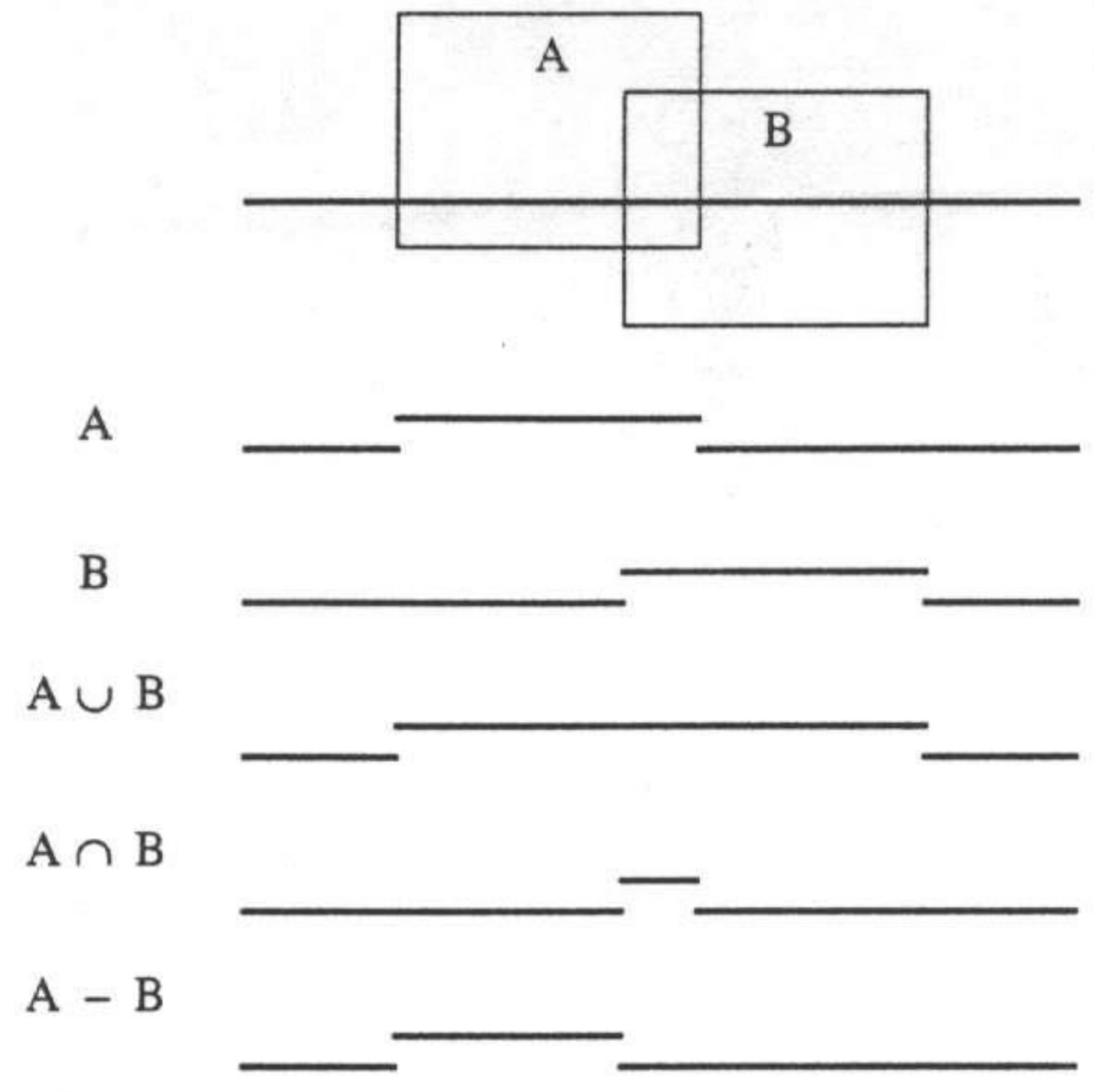
Ray Casting



Ray casting:
from pixel/eye to light source

Ray Classification




A
B
A
B
A ∪ B
A ∩ B
A – B

CSG Trees: Ray Tracing

- INPUT: Assume that we have a ray *R* and a CSG tree *T*

- If *T* is a solid,
  - compute all intersections of *R* with *T*
  - return parameter values and normals
- If *T* is a transformation
  - apply inverse transformation to *R* and recursion
  - apply inverse transpose of transformation to normals
  - return parameter values
- Otherwise *T* is a Boolean operation
  - recursion on two children to obtain two sets of intervals
  - apply operation in *T* to intervals
  - return parameter values.

- OUTPUT: Display closest intersection points

CSG Trees: Inside/Outside Test

- Given a point *p* and a tree *T*, determine if *p* is inside/outside the solid defined by *T*

- If *T* is a solid
  - Determine if *p* is inside *T* and return
- If *T* is a transformation
  - Apply the inverse transformation to *p* and recursion
- Otherwise *T* is a Boolean operation
  - Recursion to determine inside/outside of left/right children
  - If T is Union
    - If either child is inside, return inside, else outside
  - If T is Intersection
    - If both children are inside, return inside, else outside
  - If T is Subtraction
    - If *p* is inside left child and outside right child, return inside, else outside

Application: Computing Volume

- Put bounding box around object
- Pick n random points inside the box
  - Determine if each point is inside/outside the CSG Tree
- Volume $\approx \frac{\#inside}{n}$

Questions?

- Can we use a different set of primitives ?
- Is the CSG representation unique ?
- How to determine if two solids are identical ?

Problems with CSG

- Non-unique representation
- Difficulty of performing analysis for some tasks

Boolean Operators

U* (regular union)

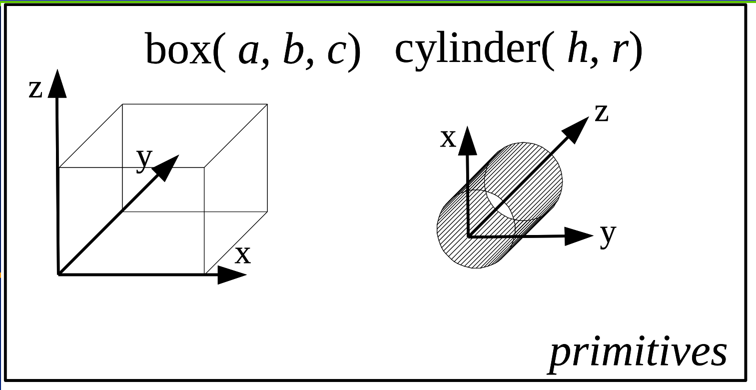
-* (regular difference)

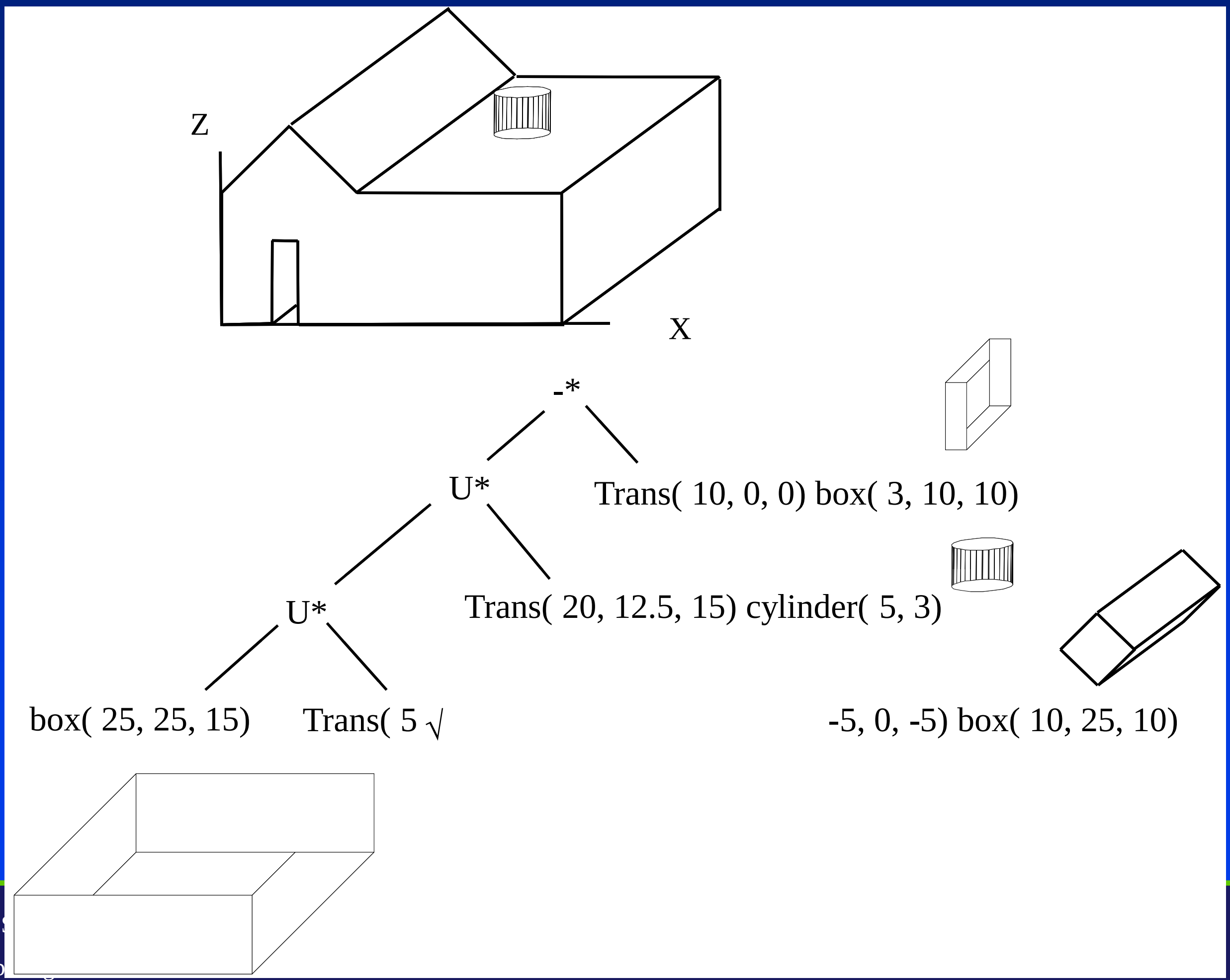
∩* (regular intersection)

CSG Tree:

Sequence of operators → design

CSG Examples





Regularized Operators

- Is the set of 3D solids is closed with respect to (U, -, ∩)?
- closure of a set S: *k*S
- interior of a set S: *i*S

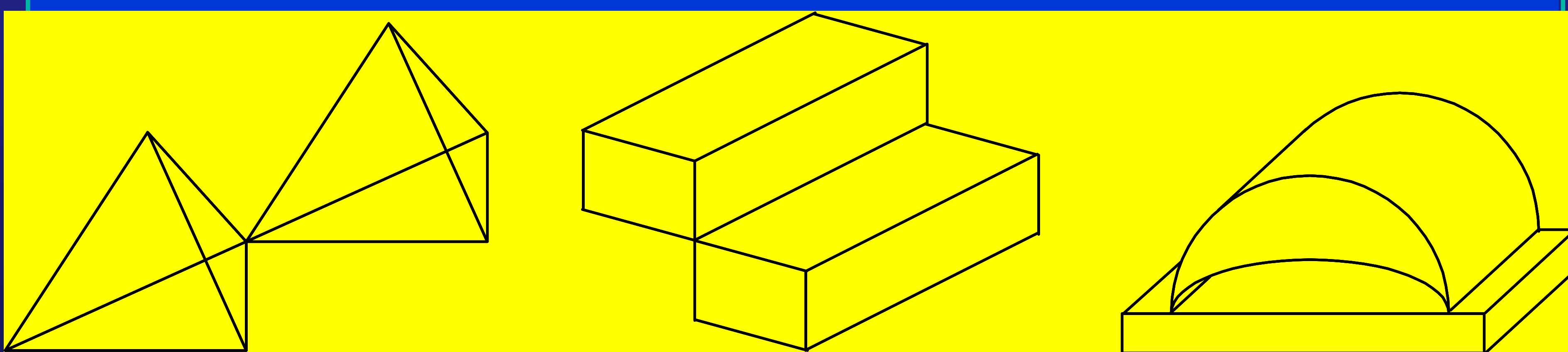
- A U* B = *k i* (A U B)
- A -* B = *k i* (A –B)
- A ∩* B = *k i* (A ∩ B)
- Why is closure over operations important?
- Uniform data structures

Regularized Operators

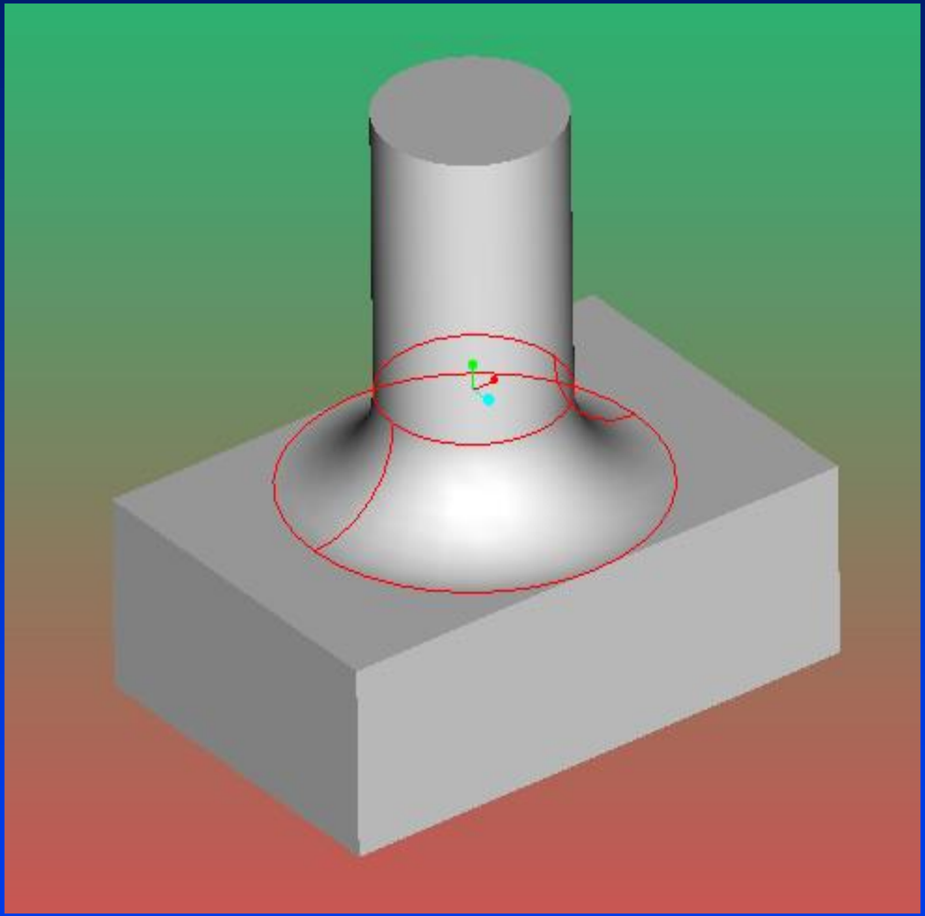
- Maintain solid as a *regular 2-Manifold*

- 2-Manifold regular solids
- Open neighborhood of each point is similar to an open disc

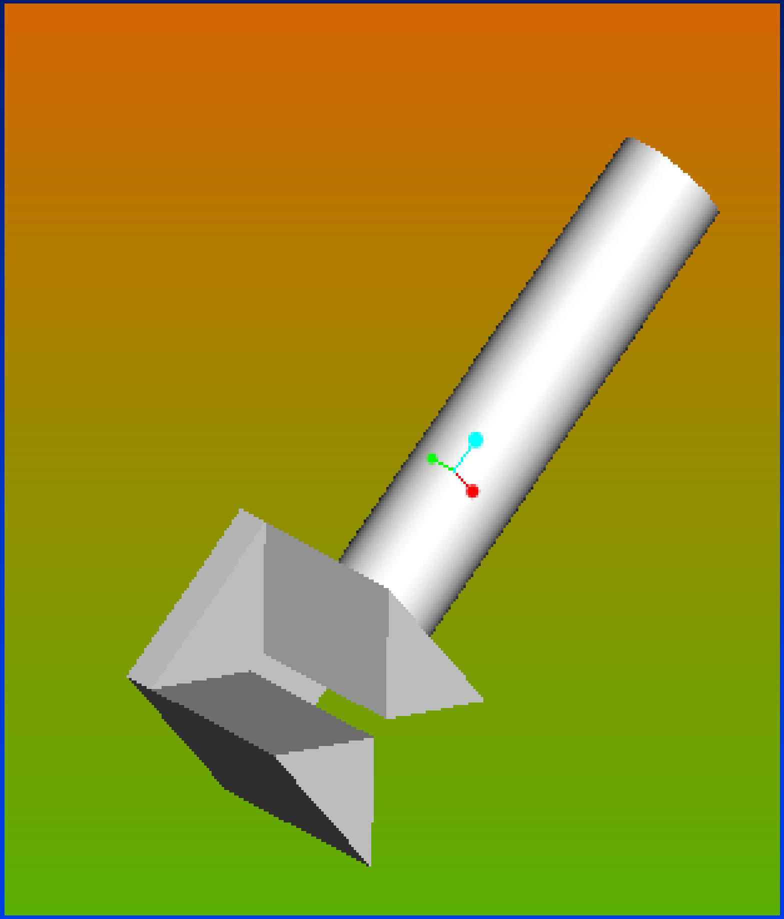
Non 2-Manifold:



Examples of Solid Models

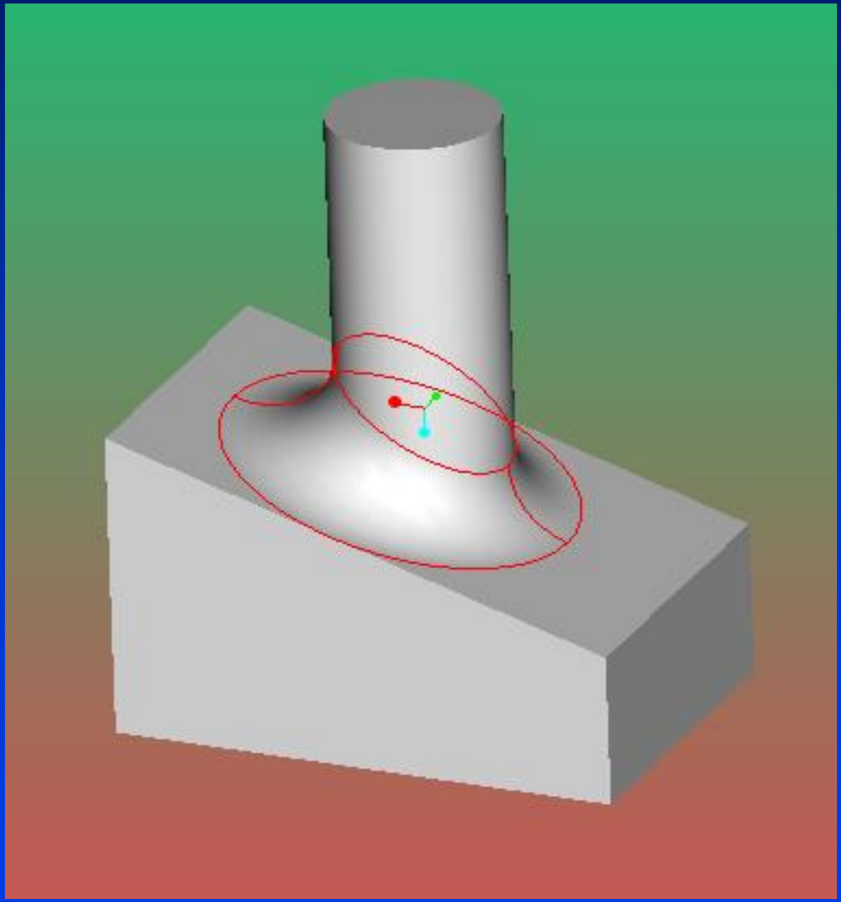


Torus

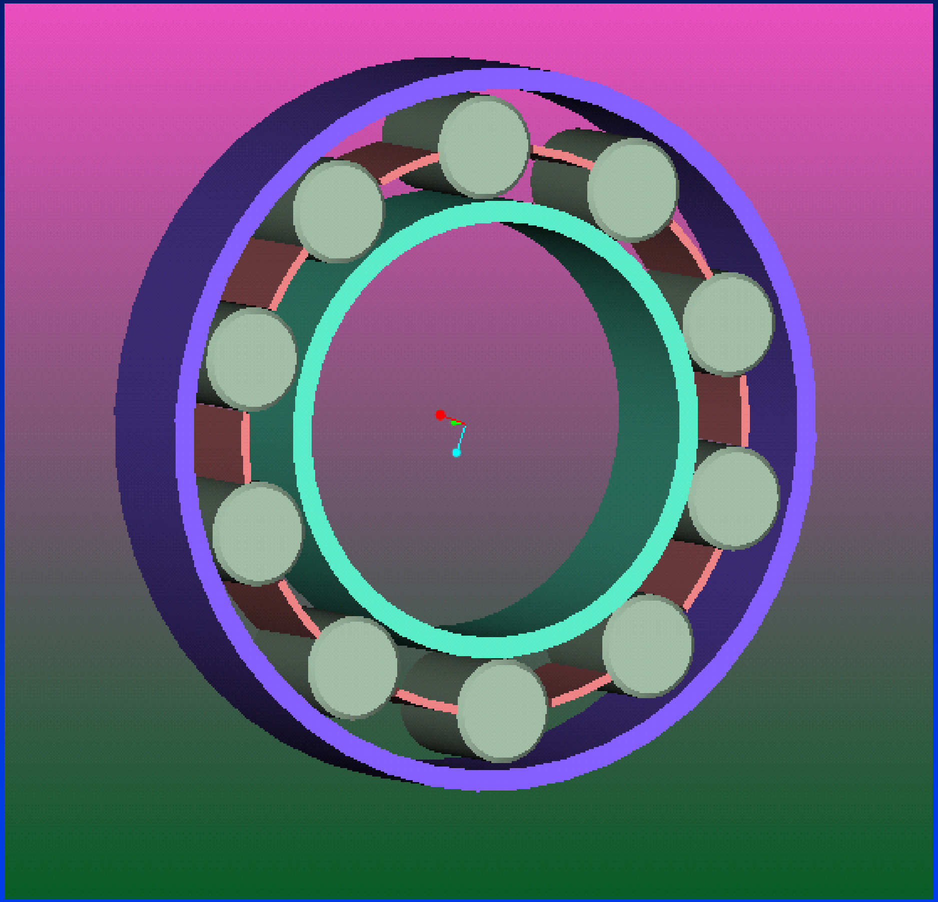


Lock

More Examples

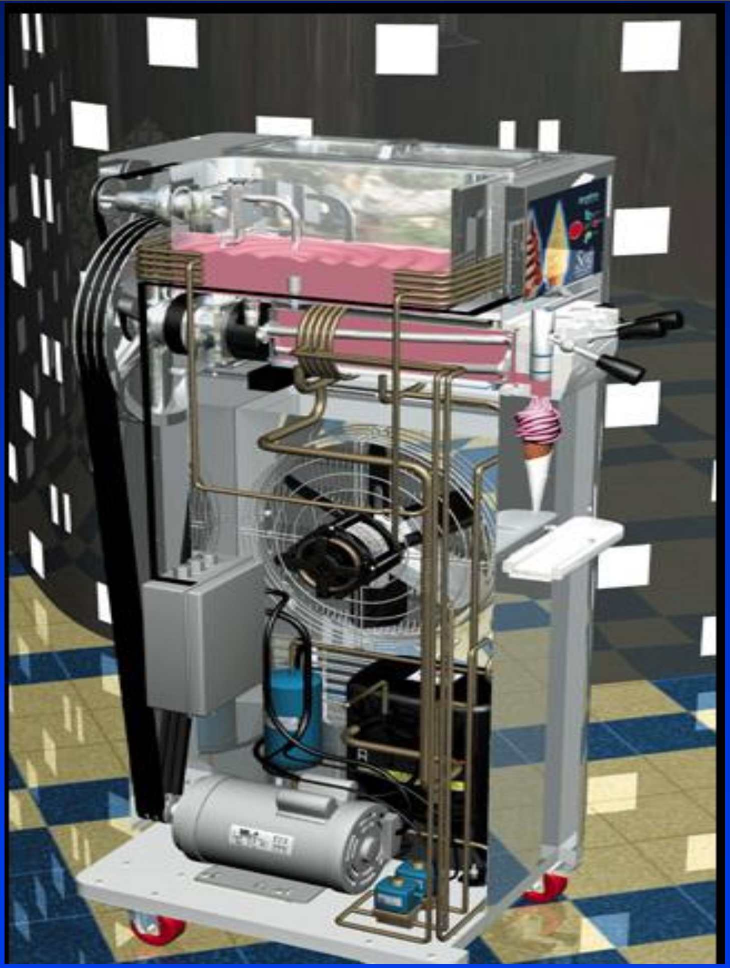


Slanted Torus



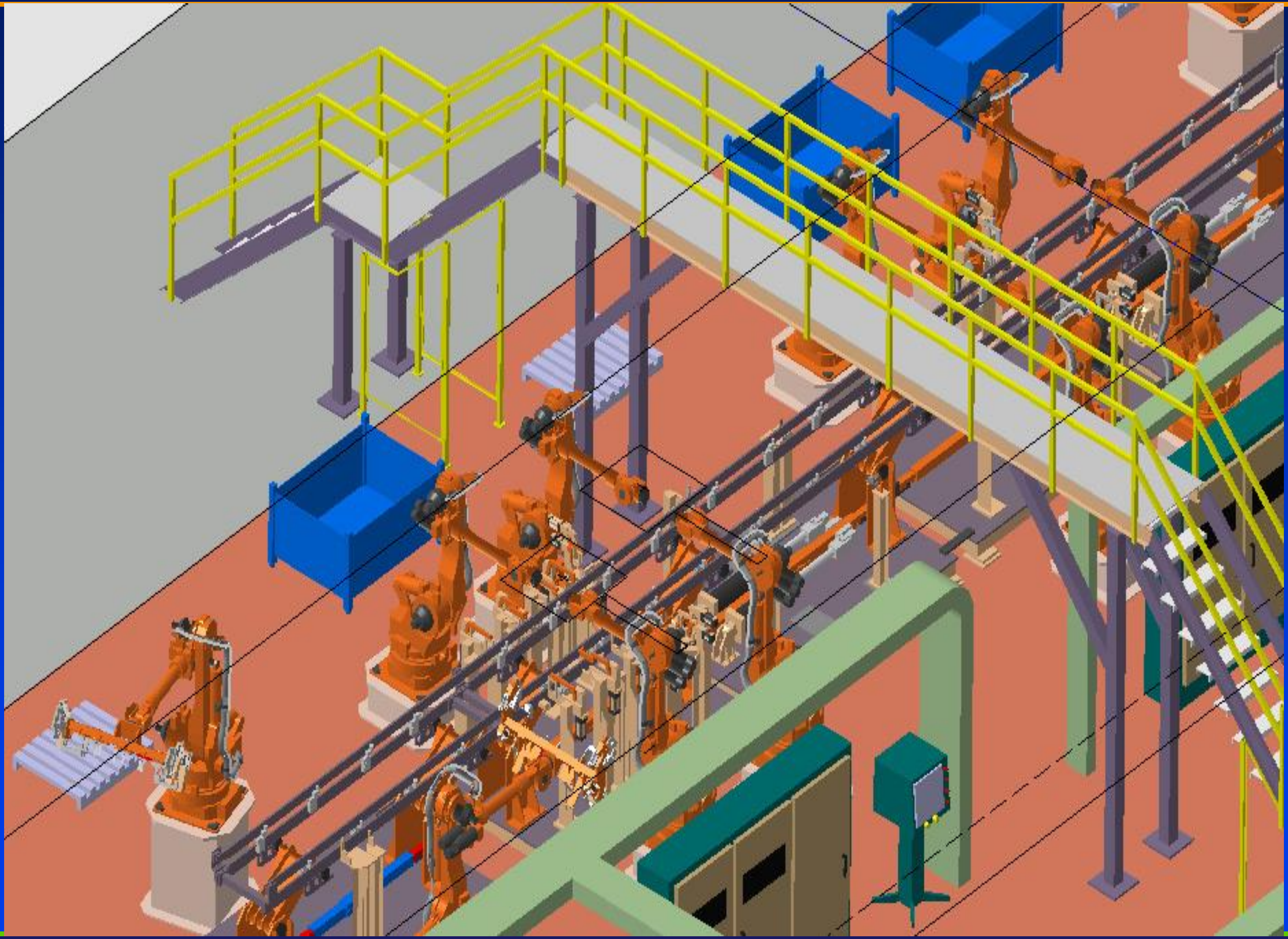
Bearing

Examples

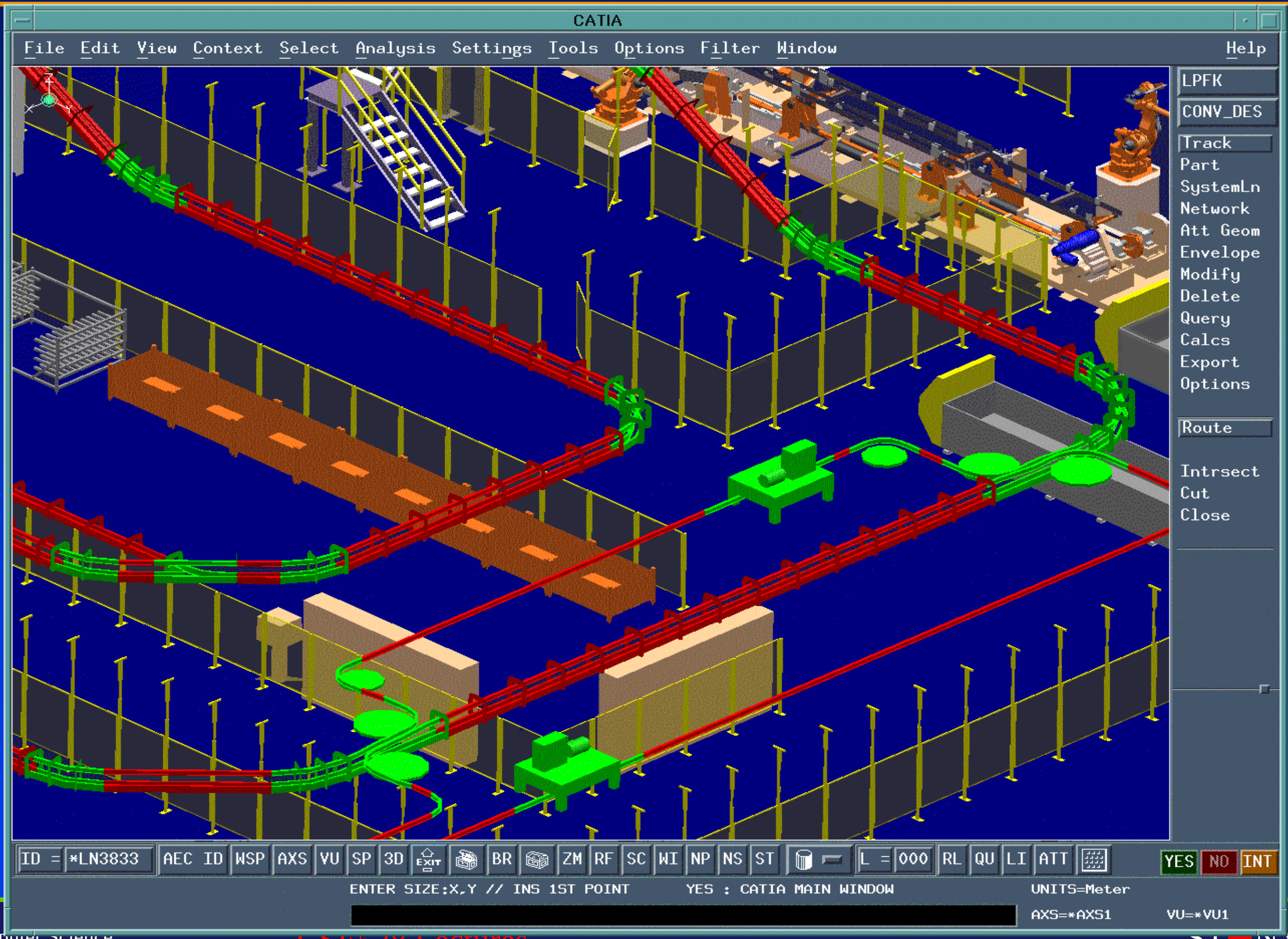


Solid Model of an Ice-Cream Machine

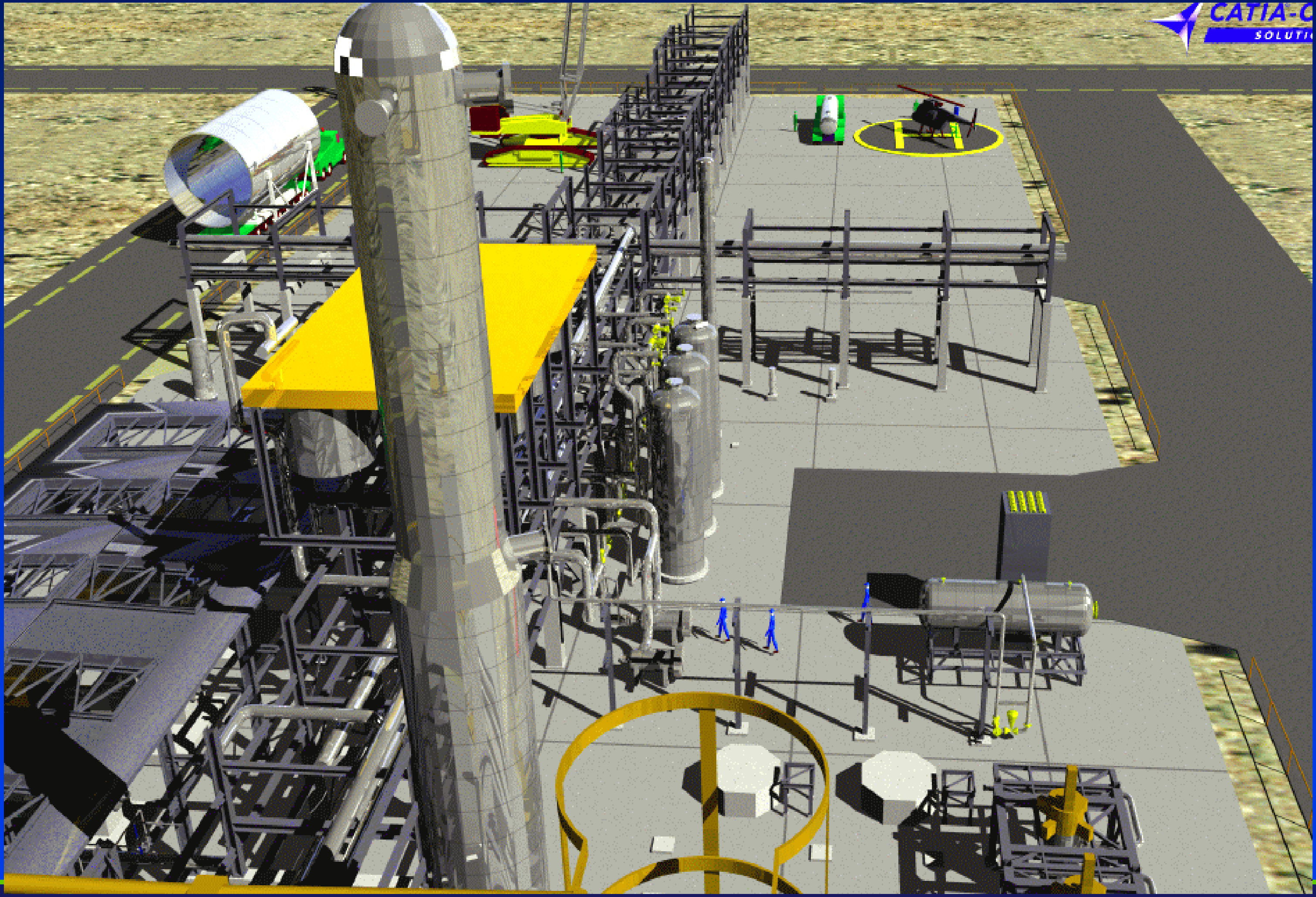
Chemical Plants



Chemical Plants



Chemical Plants (Example)

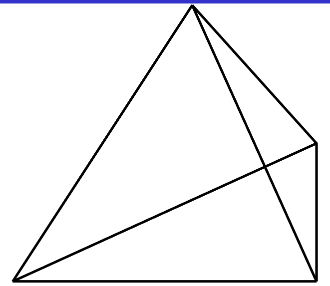


B-REP

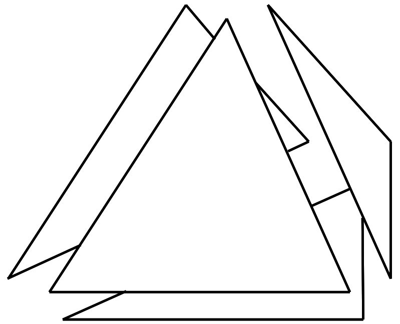
Boundary of a solid…

Boundary of surfaces…

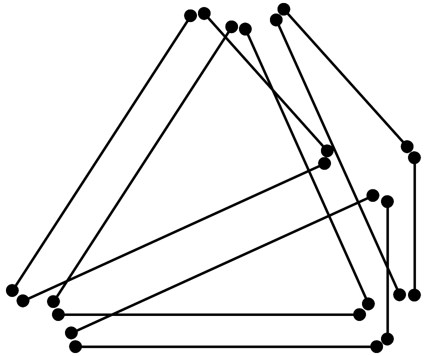
Boundary of curves (edges)…



(a) Solid: bounded, connected subset of E^3

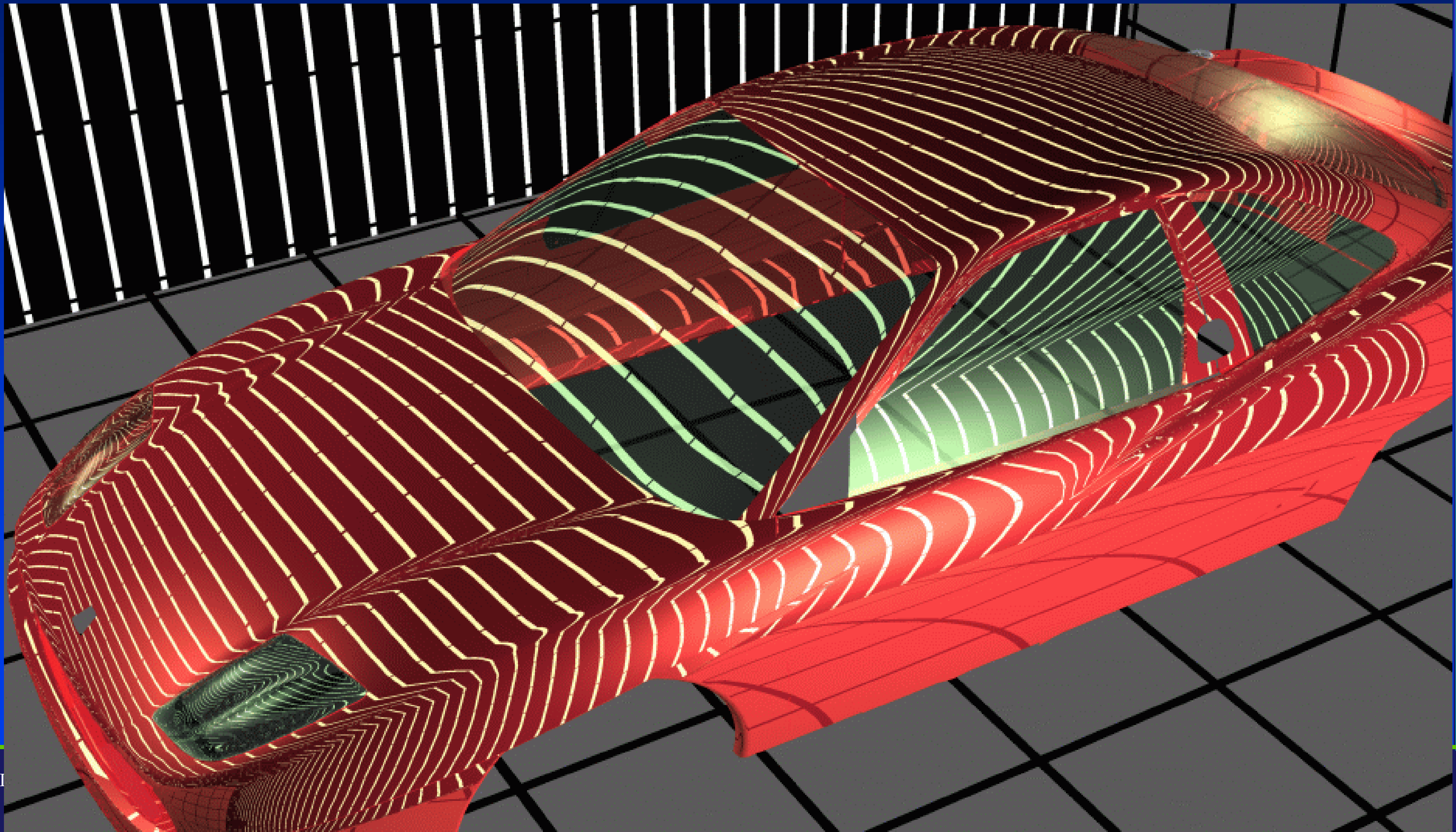


(b) Faces: boundary of solid
bounded, connected subsets of Surfaces

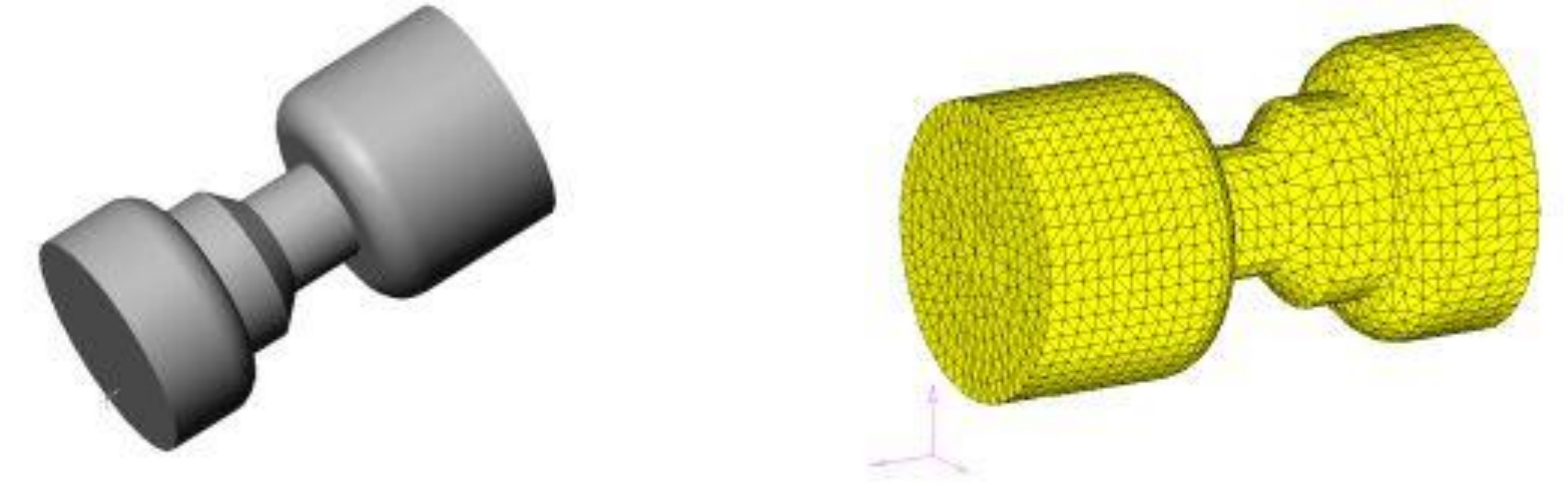


(c) Edges: boundary of faces
bounded, connected subsets of curves

Surface Modeling



B-REP Polyhedral Models

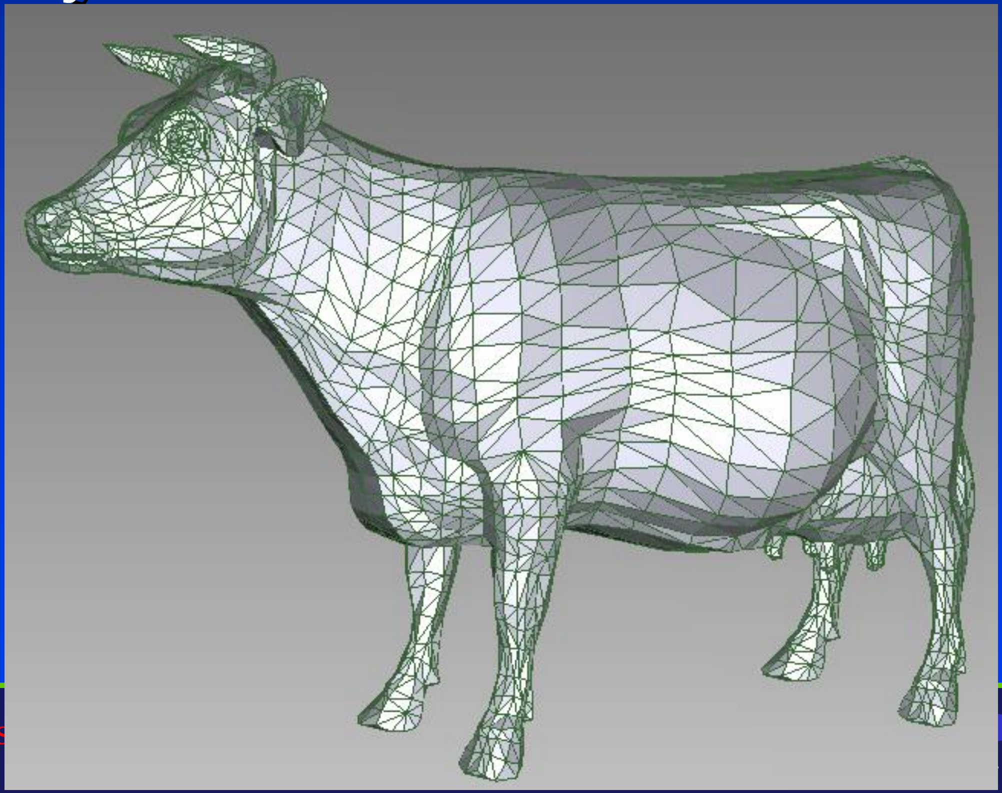


B-REP (Boundary REPresentation)

- What entities define the
- Boundary of a solid ?
- Boundary of surfaces?
- Boundary of curves (edges) ?
- Boundary of points ?

Boundary Representation

- Stores the boundary of a solid
  - Geometry: vertex locations
  - Topology: connectivity information
    - Vertices
    - Edges
    - Faces

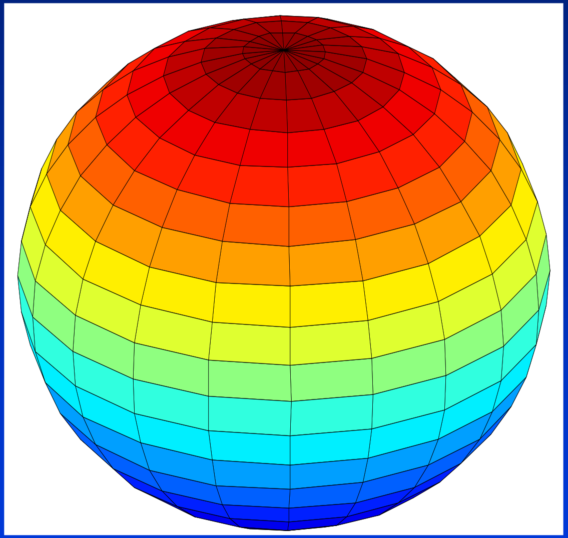


Using a Boundary Model

- Compute volume, weight
- Compute surface area
- Point inside/outside solid
- Intersection of two faces
- ….

Polygonal Meshes

- Planar polygons (planar facets or faces) are used to model the surface of complex objects



- In ‘Contours’, a polyline was represented by a list of coordinates for the vertices that connect the line segments
- Here, a polygonal mesh is represented by the list of vertex coordinates for the vertices that define the planar polygons in the mesh

Boundary Representation

- Constant time adjacency information
  - For each vertex,
    - Find edges/faces touching vertex
  - For each edge,
    - Find vertices/faces touching edge
  - For each face,
    - Find vertices/edges touching face

Polygonal Meshes - Representation by List of vertices

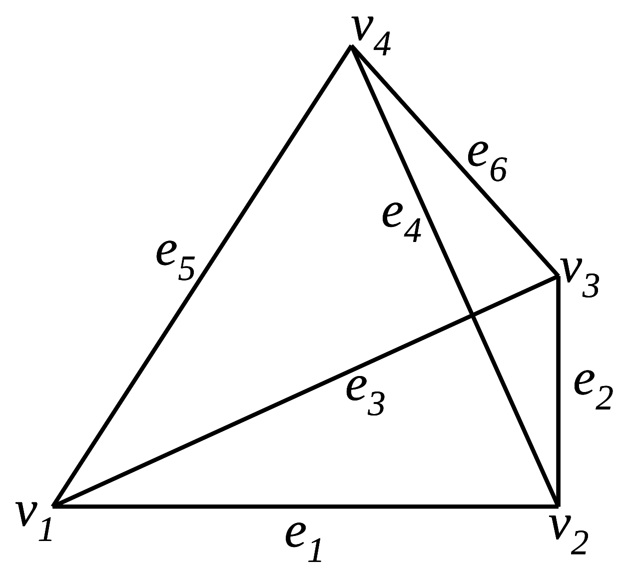
- As many polygons tend to share each vertex, an indirect representation that allows each vertex to be listed only once is used
  - Number the vertices from 1 to n; store the coordinates for each vertex once:
    $v_1 = (x_1, y_1, z_1)$

    ..

    ..

    $v_n = (x_n, y_n, z_n)$
  - Represent each face by a list of vertices in the polygon for the face; for consistency, follow the convention of listing then in the order of being encountered (clockwise around the face)
- Easy to find all the vertices for a given face, and any change in the coordinates of a vertex automatically (indirectly) changes all faces that use the vertex
- Does not explicitly represent the edges between adjacent faces
- Does not provide an efficient way to find all faces that include a given vertex
- Winged edge data structure resolves these problems

An Edge-Based Model



Faces:

f_1	e_1 e_4 e_5
f_2	e_2 e_6 e_4
f_3	e_3 e_5 e_6
f_4	e_3 e_2 e_1

Edges:

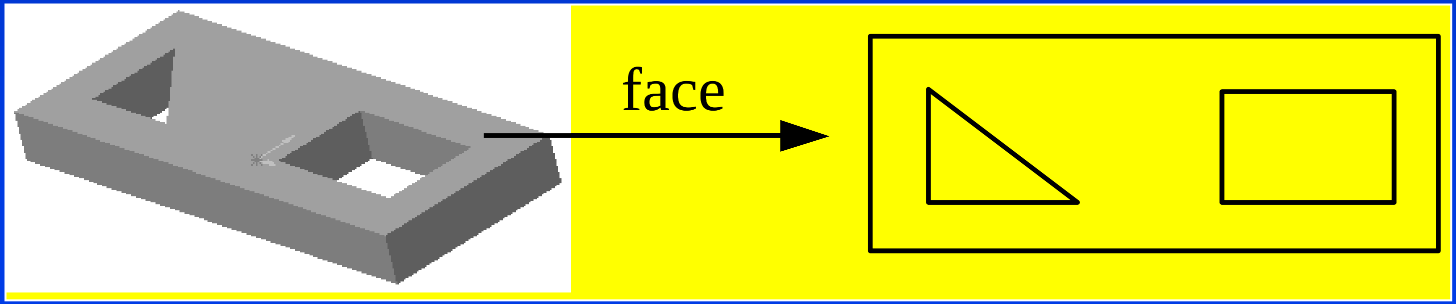
e_1	v_1 v_2
e_2	v_2 v_3
e_3	v_3 v_1
e_4	v_2 v_4
e_5	v_1 v_4
e_6	v_3 v_4

Vertices:

v_1	x_1 y_1 z_1
v_2	x_2 y_2 z_2
v_3	x_3 y_3 z_3
v_4	x_4 y_4 z_4
v_5	x_5 y_5 z_5
v_6	x_6 y_6 z_6

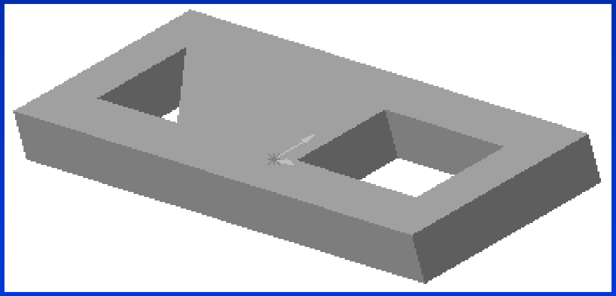
Edge-Based Models

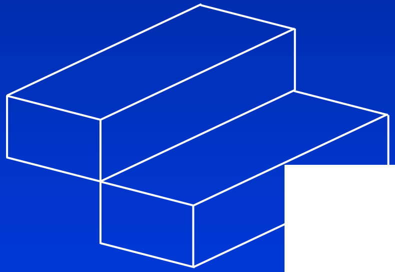
- Less efficient algorithms for computing surface area: (1) identify loops; (2) compute area of each loop; and (3) compute area of face



Observations

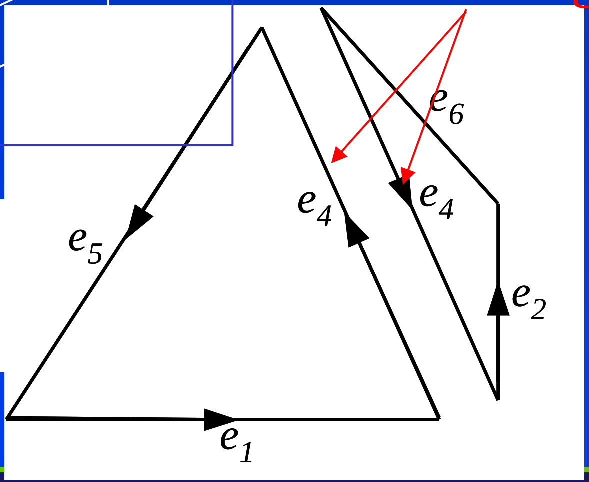
2-Manifold => Each edge is shared by exactly 2 faces





Face CCW convention =>

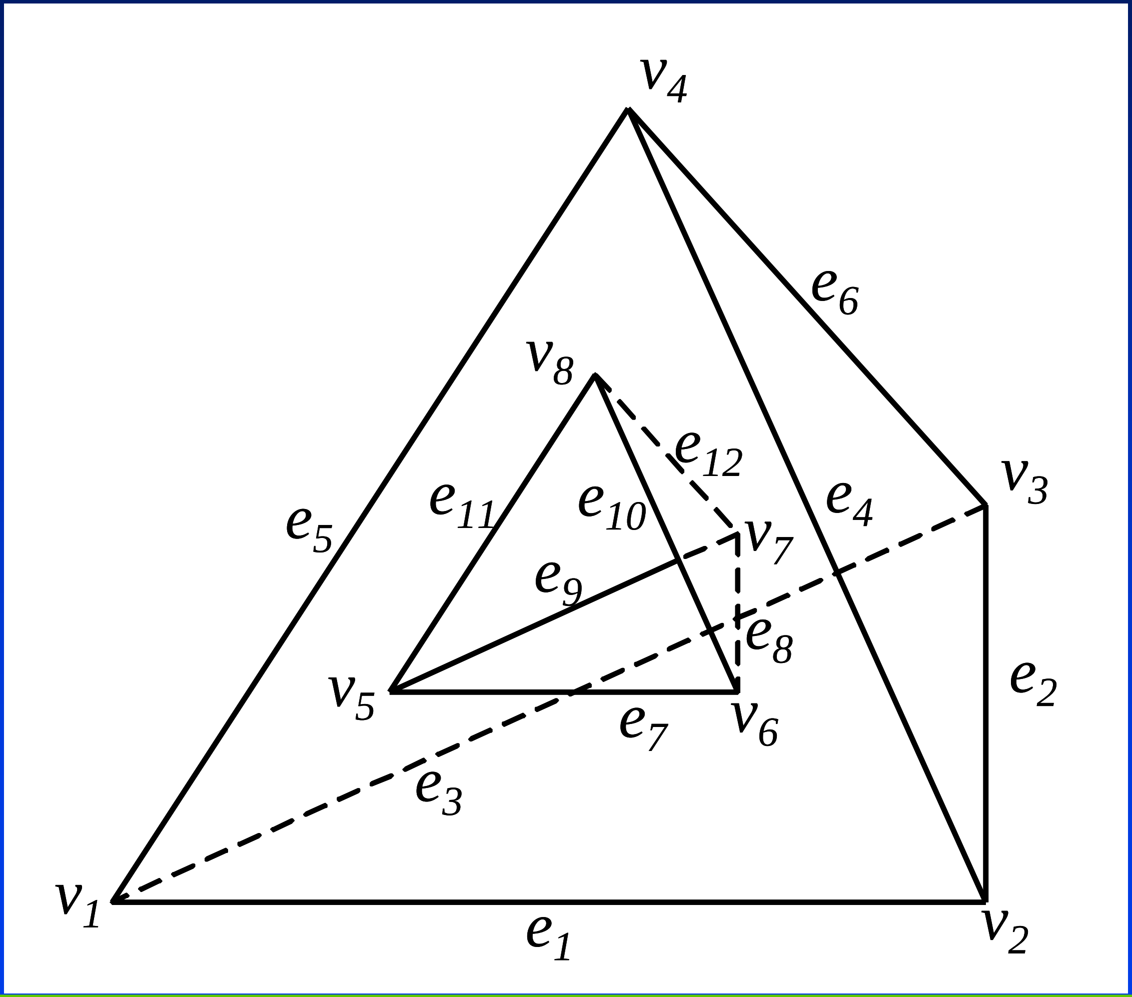
Each edge is once +ve, once -ve



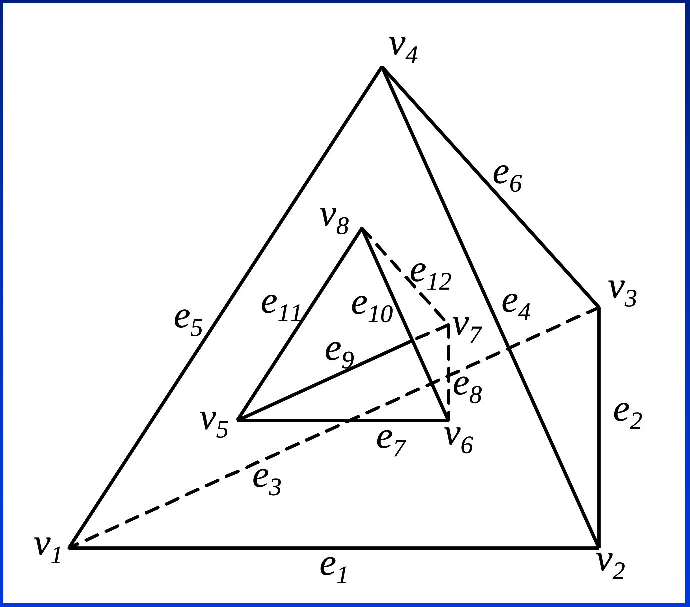
Boundary Representation

- Advantages
  - Explicitly stores neighbor information
  - Easy to render/display
  - Easy to calculate volume
  - Nice-looking surface
- Disadvantages
  - CSG very difficult
  - Inside/Outside test hard

B-REP Example



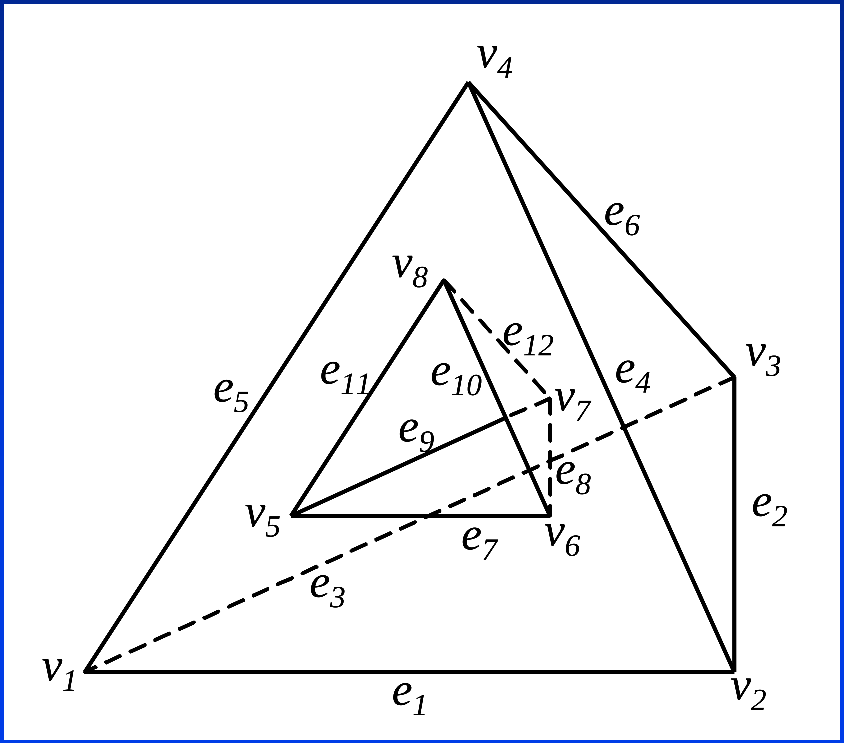
B-REP Example



Vertices:

v_1	$x_1\ y_1\ z_1$
v_2	$x_2\ y_2\ z_2$
v_3	$x_3\ y_3\ z_3$
v_4	$x_4\ y_4\ z_4$
v_5	$x_5\ y_5\ z_5$
v_6	$x_6\ y_6\ z_6$
v_7	$x_7\ y_7\ z_7$
v_8	$x_8\ y_8\ z_8$
v_9	$x_9\ y_9\ z_9$
v_{10}	$x_{10}\ y_{10}\ z_{10}$
v_{11}	$x_{11}\ y_{11}\ z_{11}$
v_{12}	$x_{12}\ y_{12}\ z_{12}$

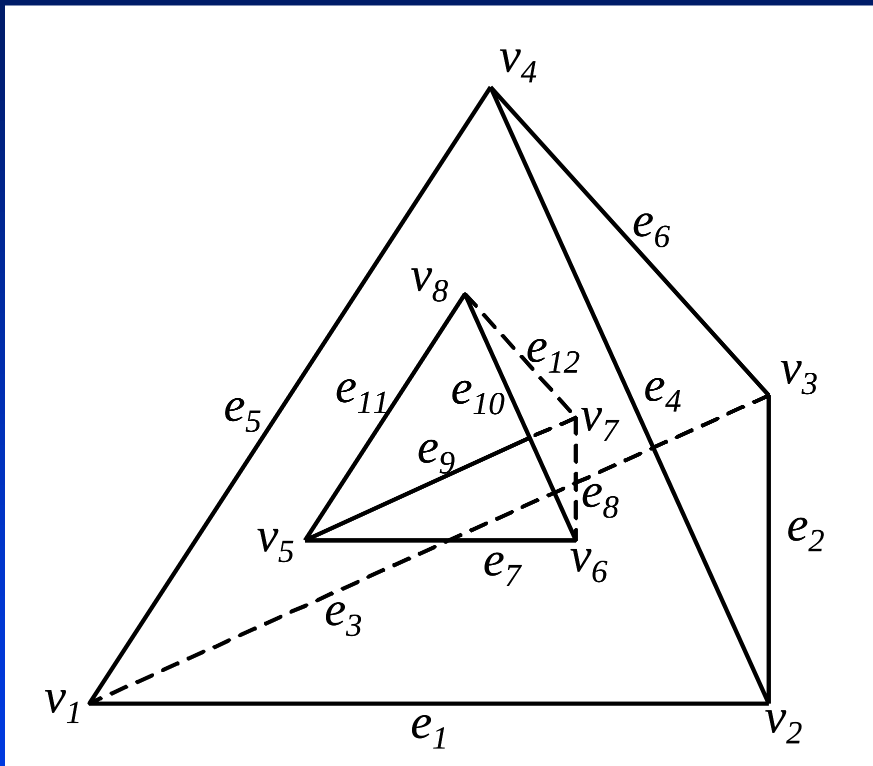
B-REP Example



Edges:

e_1	v_1	v_2
e_2	v_2	v_3
e_3	v_3	v_1
e_4	v_2	v_4
e_5	v_1	v_4
e_6	v_3	v_4
e_7	v_5	v_6
e_8	v_6	v_7
e_9	v_7	v_5
e_{10}	v_6	v_8
e_{11}	v_5	v_8
e_{12}	v_7	v_8

B-REP Example



Faces:

f_1	l_1 l_2
f_2	l_3
f_3	l_4
f_4	l_5
f_5	l_6
f_6	l_7
f_7	l_8

Loops:

l_1	$+e_1$	$+e_4$	$-e_5$
l_2	$-e_7$	$+e_{11}$	$-e_{10}$
l_3	$+e_2$	$+e_6$	$-e_4$
l_4	$+e_5$	$-e_6$	$+e_3$
l_5	$-e_1$	$-e_3$	$-e_2$
l_6	$+e_7$	$+e_8$	$+e_9$
l_7	$+e_{10}$	$-e_{12}$	$-e_8$
l_8	$-e_{11}$	$-e_9$	$+e_{12}$

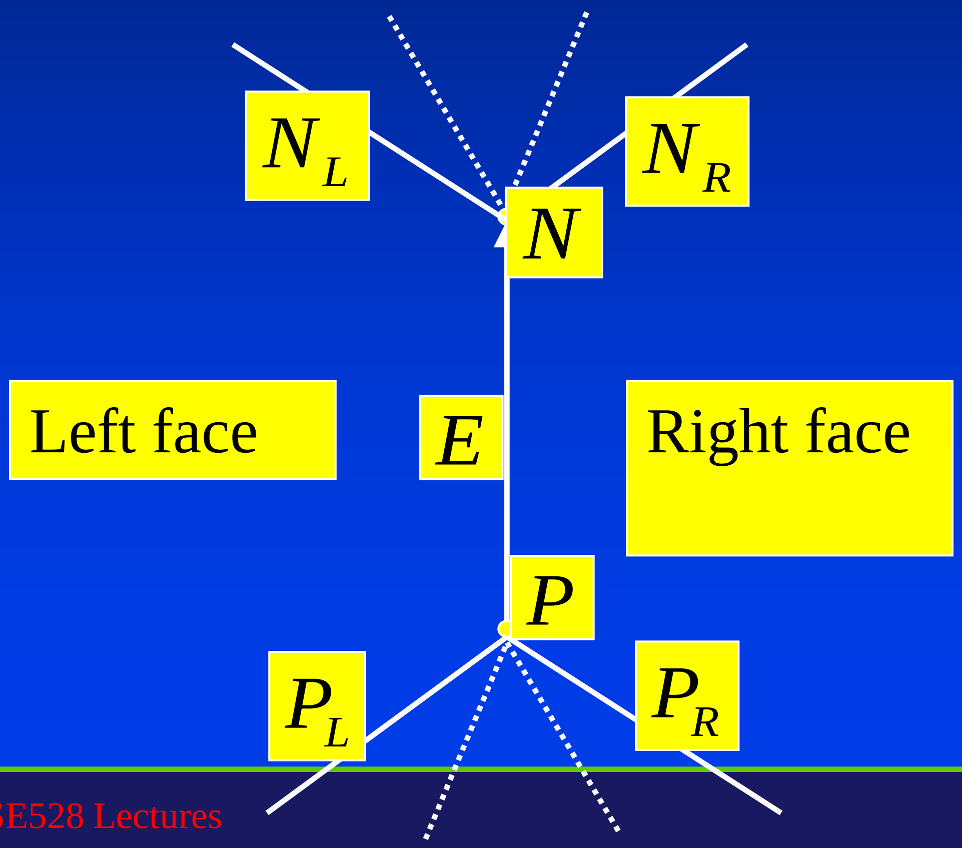
Winged Edge Data Structure

- Efficient implementation of frequently-used algorithms

- Area of face
- Hidden surface removal
- Find neighbor-faces of a face

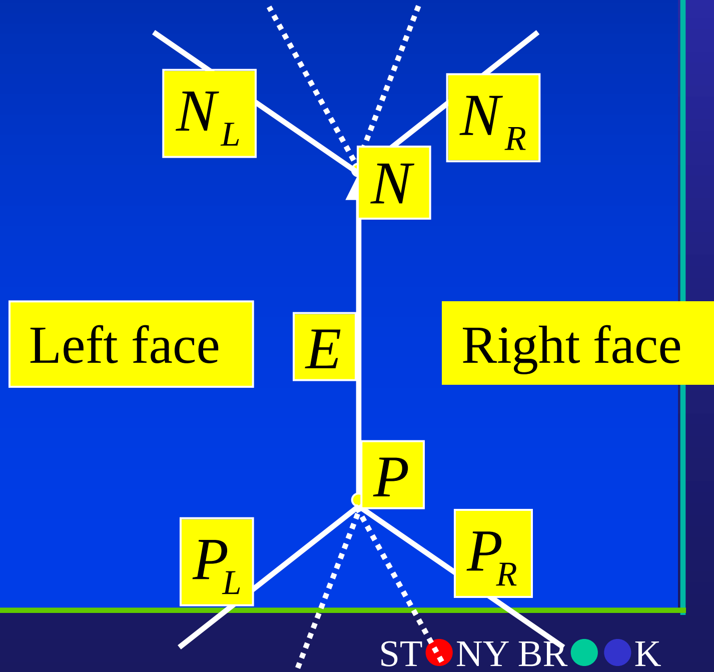
Winged Edge Data Structure

- Each vertex/face points to a single edge containing that vertex/face



Winged Edge Data Structure

- Given a face, find all vertices touching that face
- Given a vertex, find all edge-adjacent vertices
- Given a face, find all adjacent faces

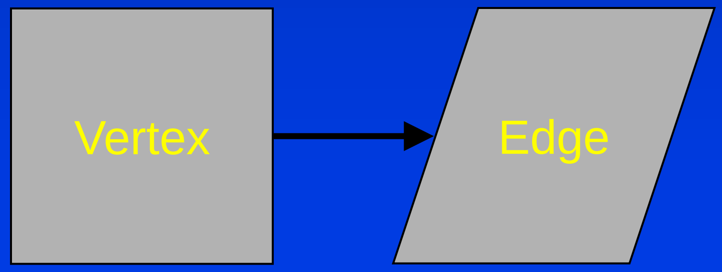


Winged Edge Data Structure

- Used to store information regarding the mesh.
- Provides efficient means to find all faces that include a given vertex.
- Network with 3 types of records - vertex, edge, and face records.
- All faces using a vertex can be found in time proportional to the number of faces that include the vertex.
- All vertices around a face can be found in time proportional to the number of vertices around the face.
- Can handle polygons with many sides; not all polygons in the mesh necessarily need to have the same size / same number of sides.
- Compact data structure that allows for very efficient algorithms.
- WEDS includes pointers that can be followed to find all neighboring elements without searching the entire mesh or storing a list of neighbors in the record for each element.
- There is 1 vertex record for every vertex in the polygonal mesh, **etc**.

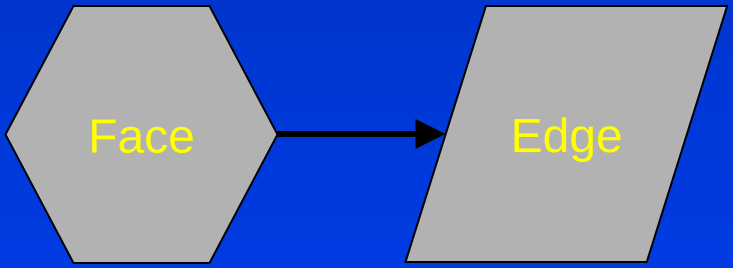
Winged Edge Data Structure

- Vertex record
  - Contains the vertex coordinates
  - Contains a unique number for the vertex
  - Contains a pointer to the record for an edge that ends at that vertex.



Winged Edge Data Structure

- Face record contains a pointer to the edge record of one of its edges

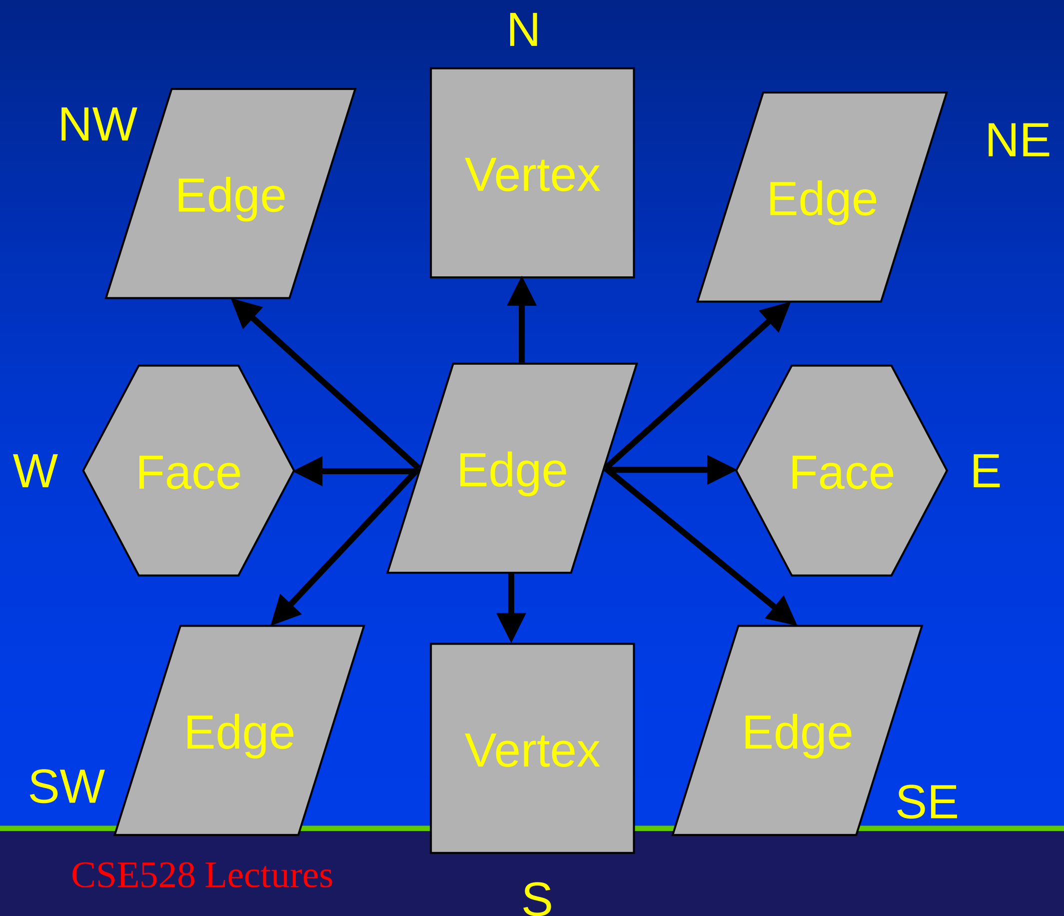


Winged Edge Data Structure

- Edge record
  - Provides most of the connectivity for the mesh
  - Contains a pointer to each of the vertices at its ends
  - Contains a pointer to each face on either side of the edge
  - Contains pointers to the four wing edges that are neighbors in the polygonal mesh
  - These pointers connect the faces and vertices into a polygonal mesh and allow the mesh to be traversed efficiently, i.e., efficient traversal from edge to edge around a face

Winged Edge Data Structure

- Edge record - Notation of compass directions is just for convenience; in a polygonal mesh, there is no global sense of direction



Traversing a Face

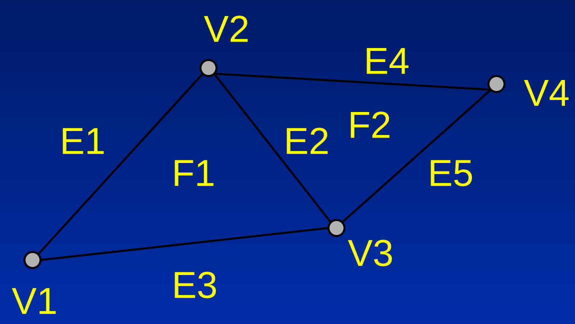
- Start at the edge pointed to by the face record
- For clockwise traversal, follow the northeast wing if the face is east of the edge, follow the southwest wing if the face is west of the edge.
- For each edge, a check must be performed to determine if the face is east or west of the edge
- Continue until the starting edge is reached

Adding a Face to a Mesh

Input: A clockwise list of vertices for the face, each consisting of a vertex number and coordinates. Use the left-hand rule to determine the clockwise direction.

1. For each vertex in the list, add a record for the vertex to the WEDS if one does not already exist.
2. For each pair of successive vertices (including first and last), add a corresponding edge record to the WEDS if it does not already exist. If any of the two vertices does not yet point to an edge, set the edge pointer of the vertex to the new edge.
3. Create a record for the face in the WEDS and add a pointer to any of the face edges.
4. For each record of an edge of the face, add the wings for traversal and update the face pointers. This depends on whether the face is east or west of each edge record

Example – Adding a Face



Input 1: V1, V2, V3

Input 2: V2, V4, V3

1. *Add each vertex to the WEDS.*
2. *Add an edge for each pair of vertices and set the edge pointers for the vertices.*
3. *Create a record for the face in the WEDS and add a pointer to any of the face edges.*
4. *For each record of an edge of the face, add the wings for traversal and update the face pointers. This depends on whether the face is east or west of each edge record.*

Vertices

V1-> E1

V2-> E1

V3-> E2

V4-> E4

Faces

F1-> E1

F2-> E4

Edges

	V2	E2
	E1	F1
	V1	E3

E2	V2	
F2	**E4**	
E5	V4	

E1	V2	E4
F1	**E2**	F2
E3	V3	E5

E4	V4	
F2	**E5**	
E2	V3	

E2	V3	
F1	**E3**	
E1	V1	

B-REP vs. CSG ?

- Using: CSG is more intuitive
- Computing: B-REP is more convenient
- Modern CAD Systems:
- CSG for GUI (feature tree)
- B-REP for internal storage and API's

B-REP: Non-Polyhedral Models

Same Data Structure, plus

For each edge, store equation

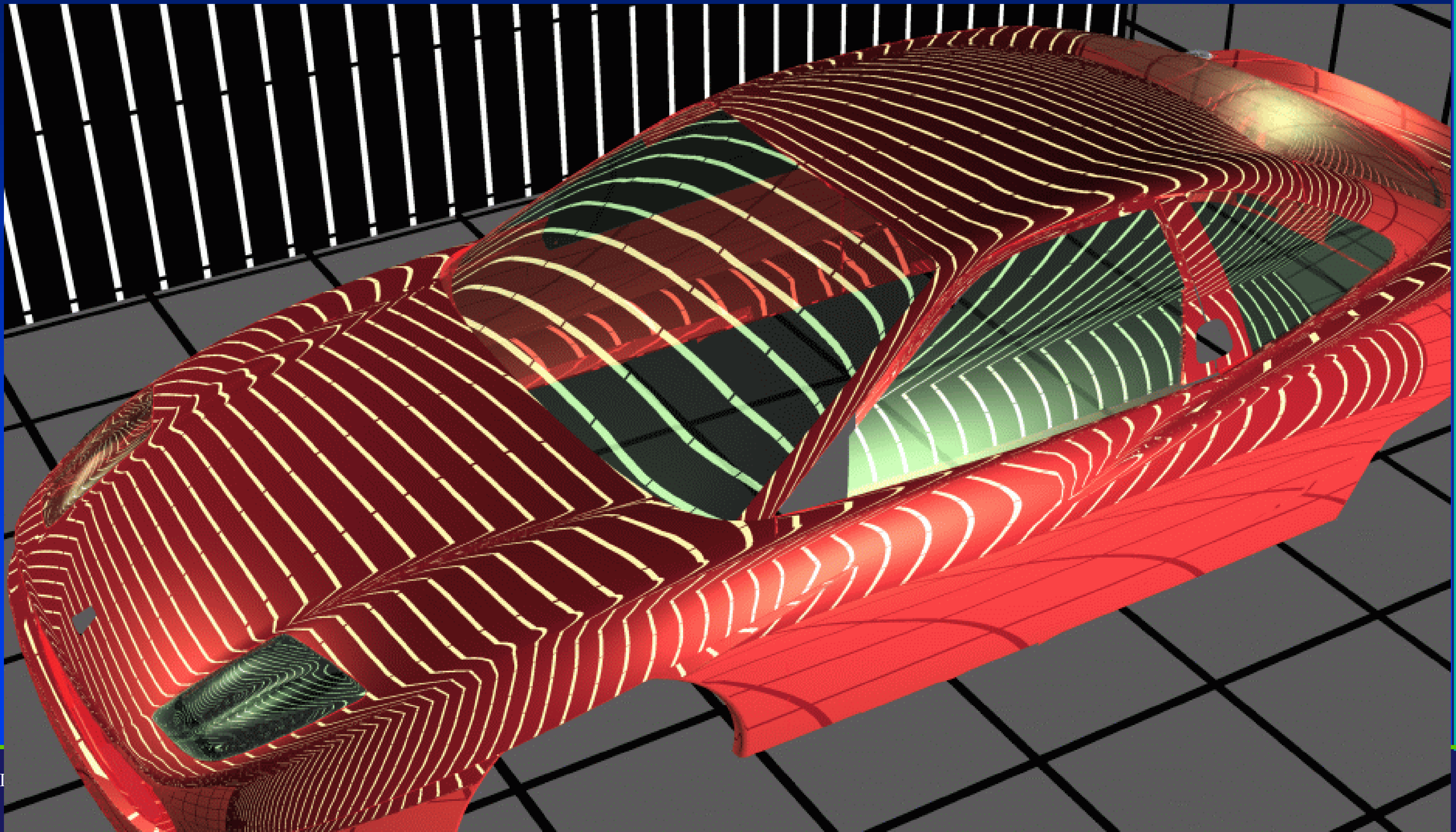
For each curved face, store equation

Why do we need to learn all of these ?

(a) To anticipate when an operation will fail

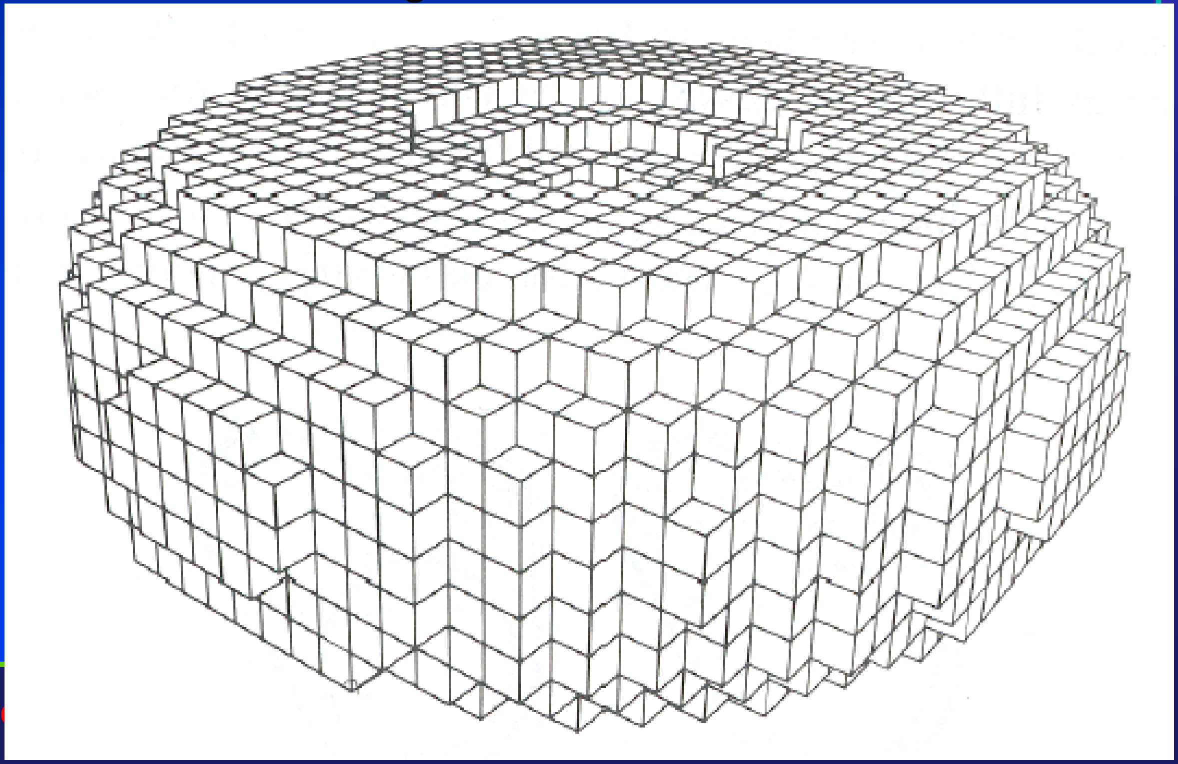
(b) To allow us to write API's

Surface Modeling



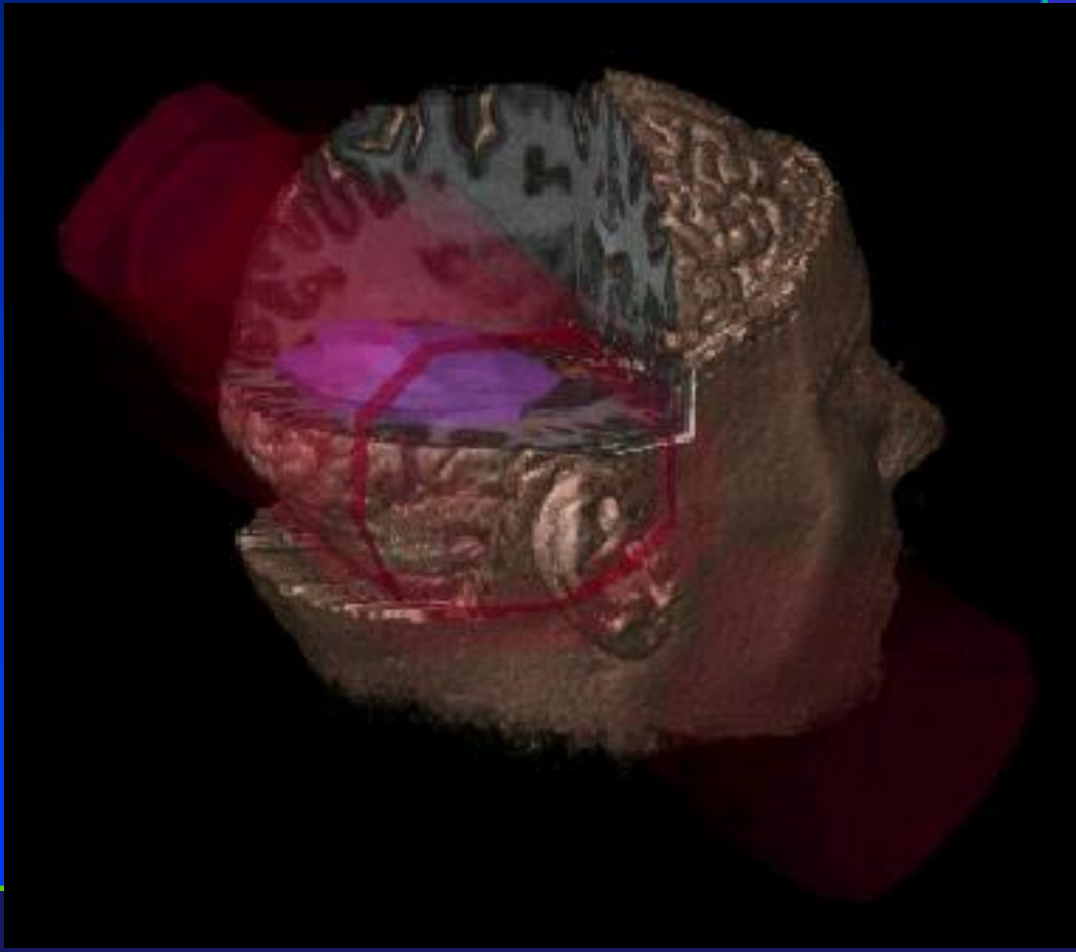
Voxel Representation

- **Partition space into uniform grid**
  - Grid cells are called *voxels* (like pixels)
- **Store properties of solid object with each voxel**
  - Occupancy
  - Color
  - Density
  - Temperature
  - Etc.



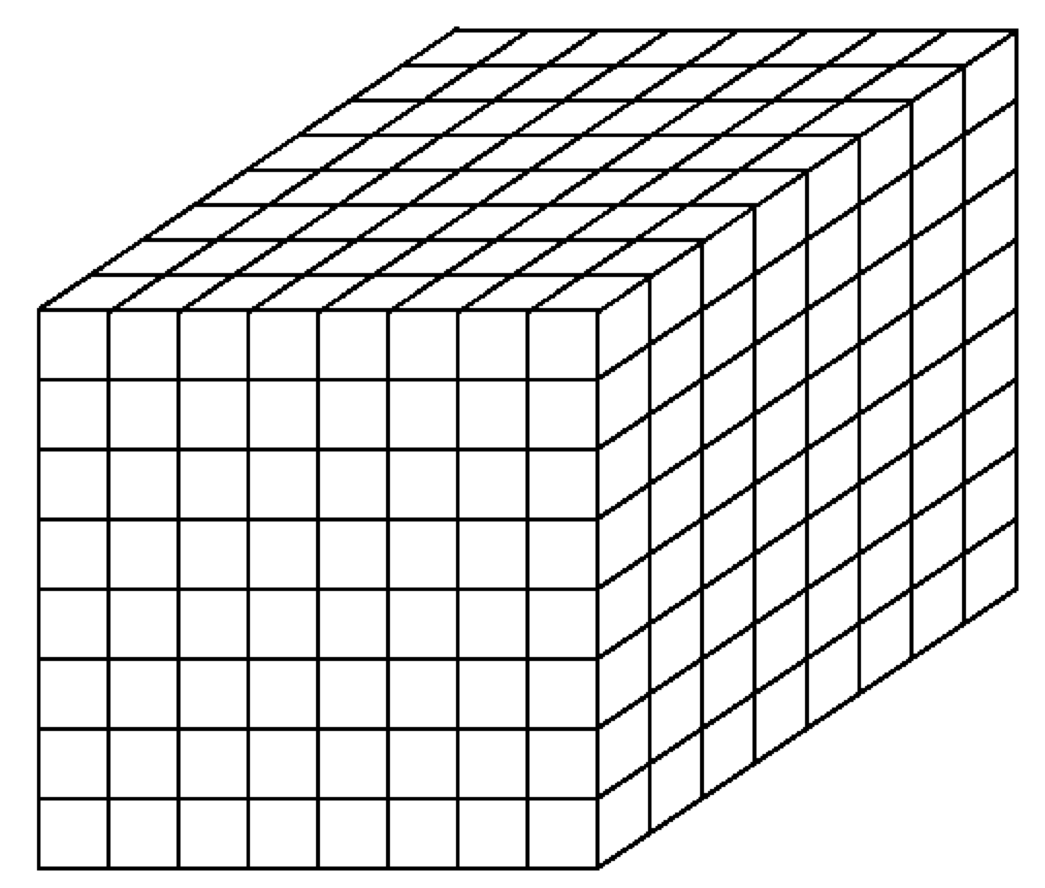
Voxel Acquisition

- **Scanning devices using different medical imaging modalities**
  - MRI
  - CAT
- **Simulation**
  - FEM



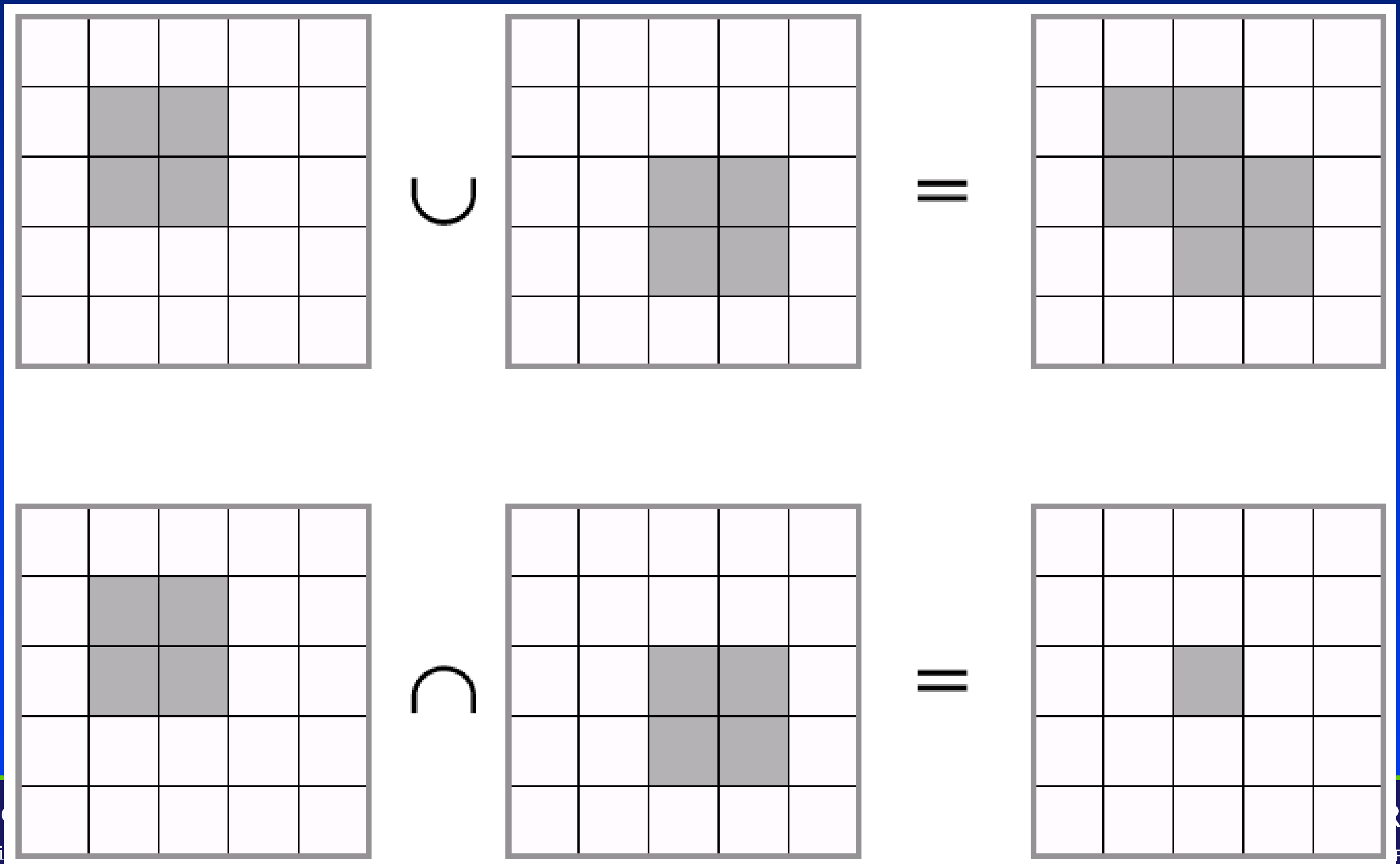
Voxel Storage

- **O(n^3) storage for *n* x *n* x *n* grid**
  - 1 billion voxels for 1000 x 1000 x 1000



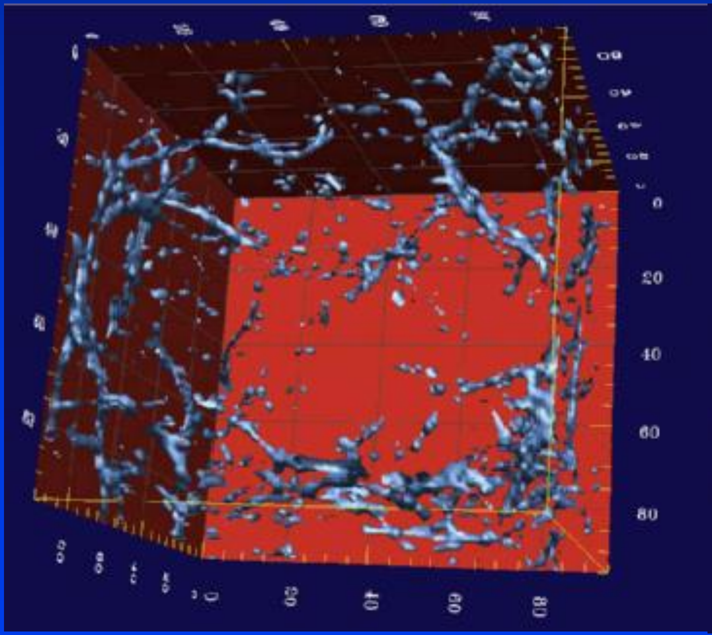
Voxel Boolean Operations

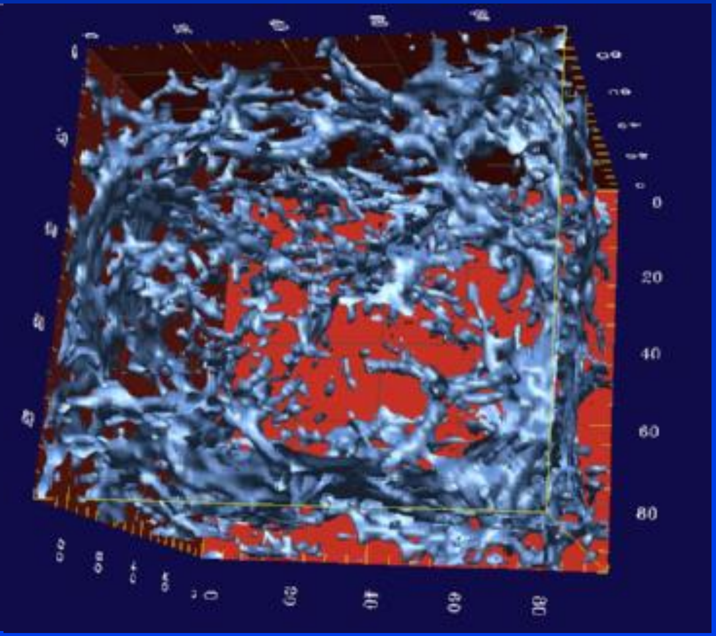
- **Compare objects voxel by voxel**



Voxel Display

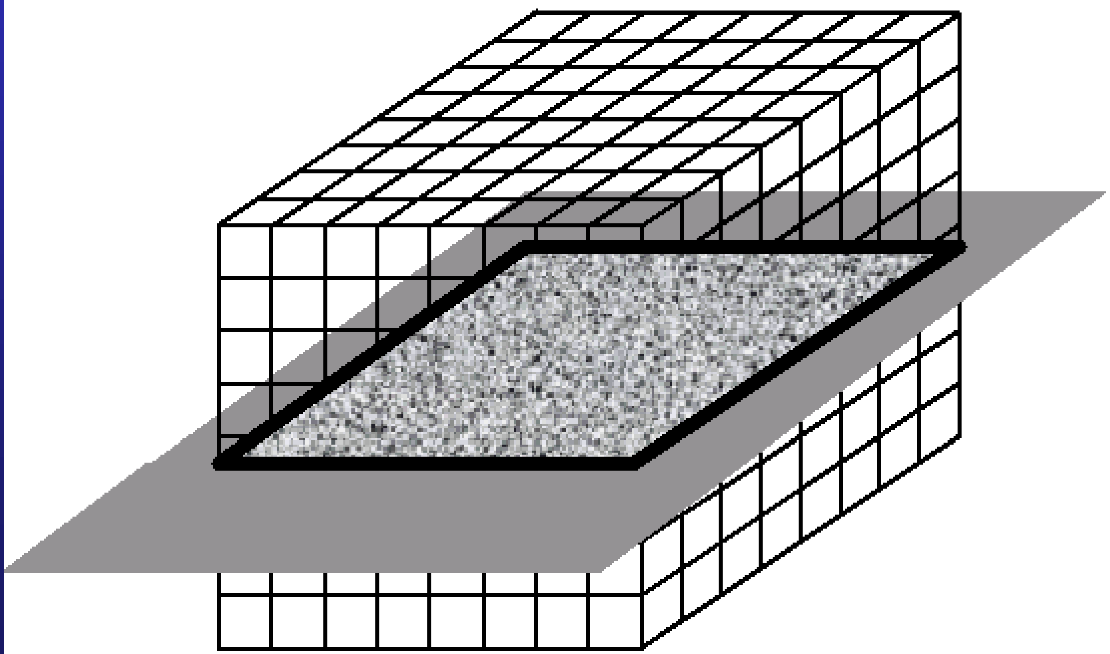
- **Isosurface rendering**
  - Render surfaces bounding volumetric regions of constant value (e.g., density)

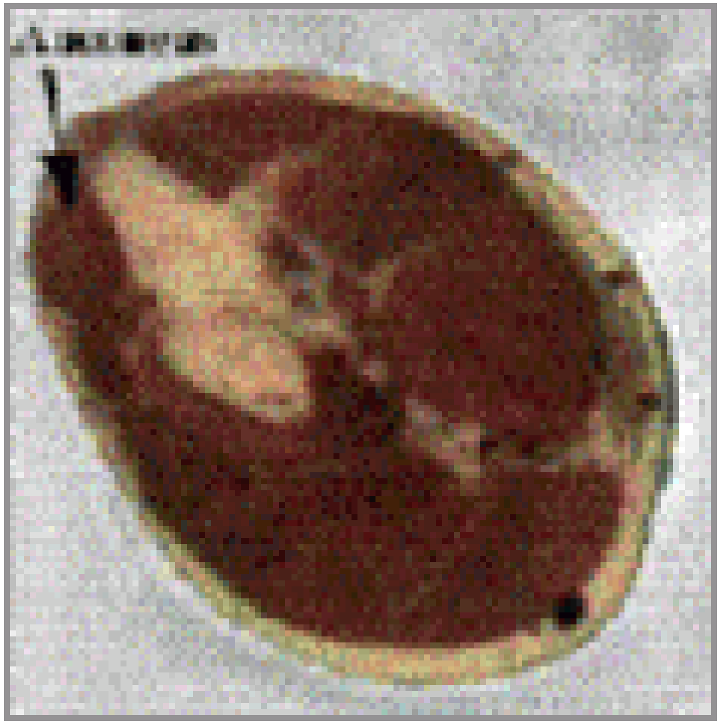




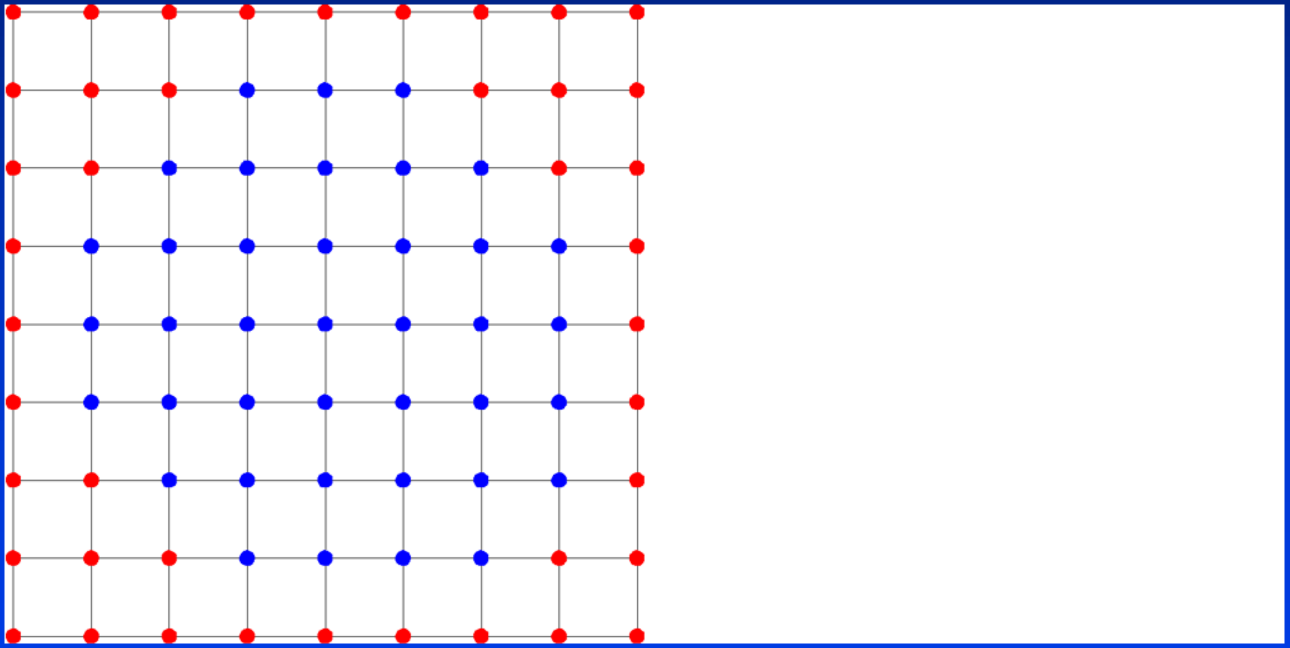
Voxel Display

- **Slicing**
  - Draw 2D image resulting from intersecting voxels with a plane

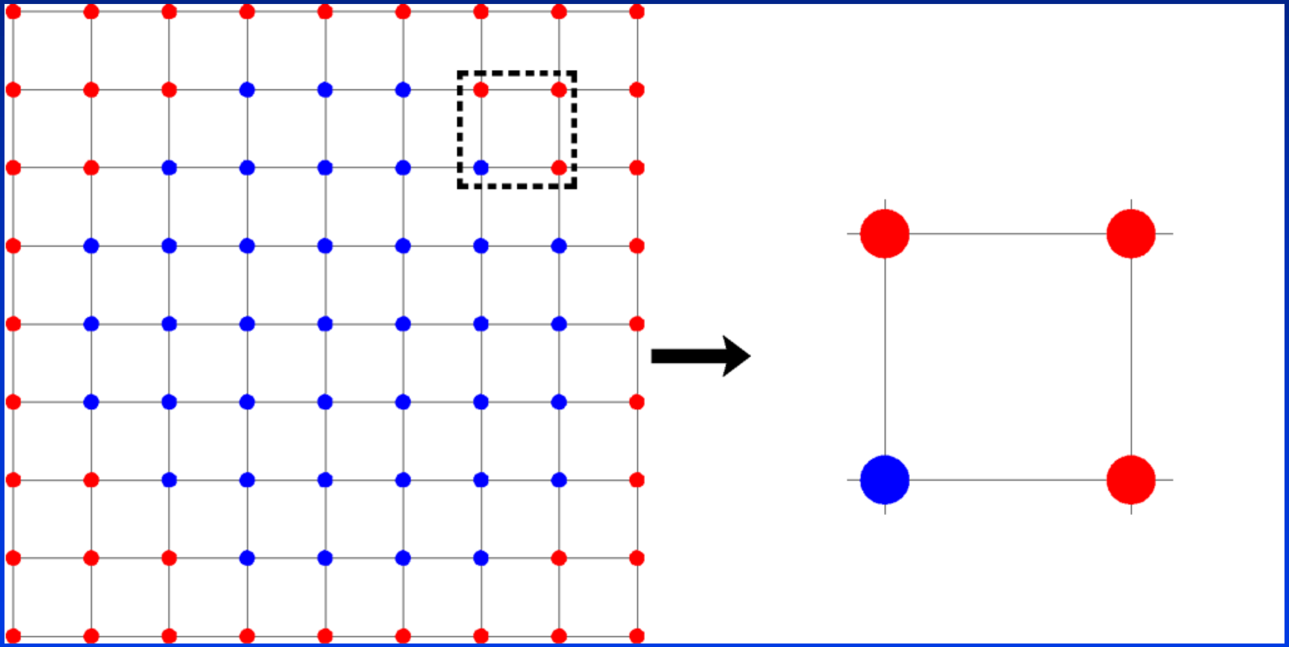




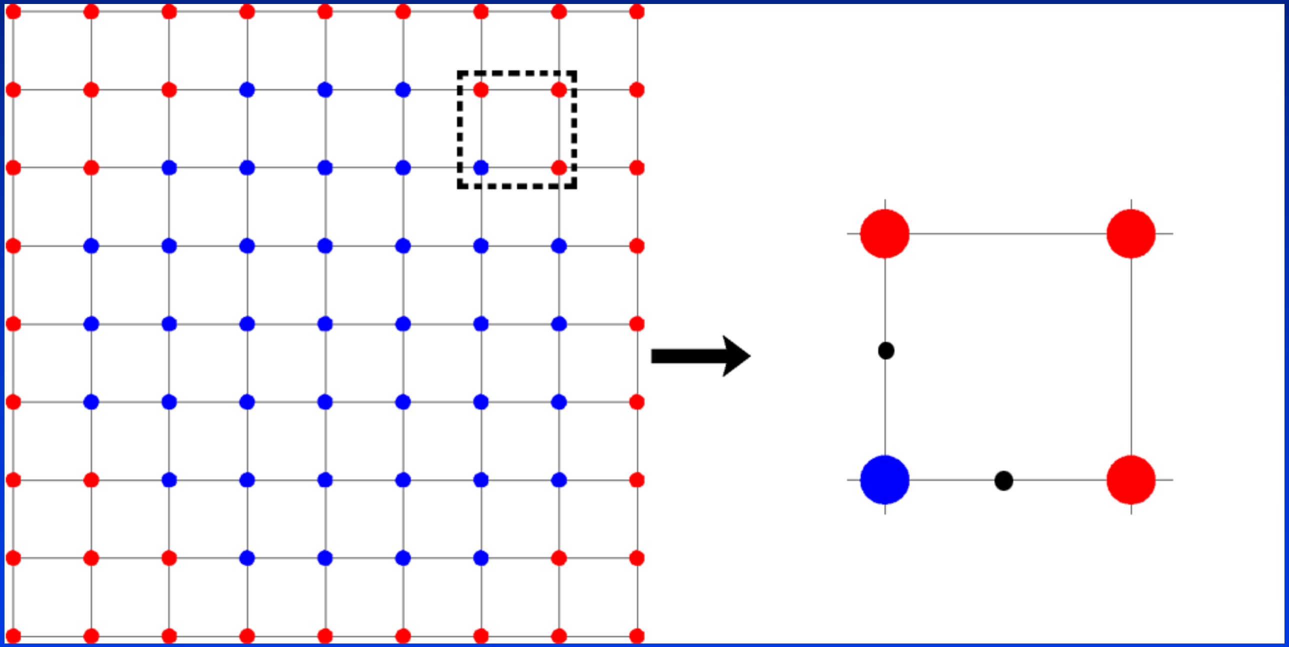
2D Polygon Generation



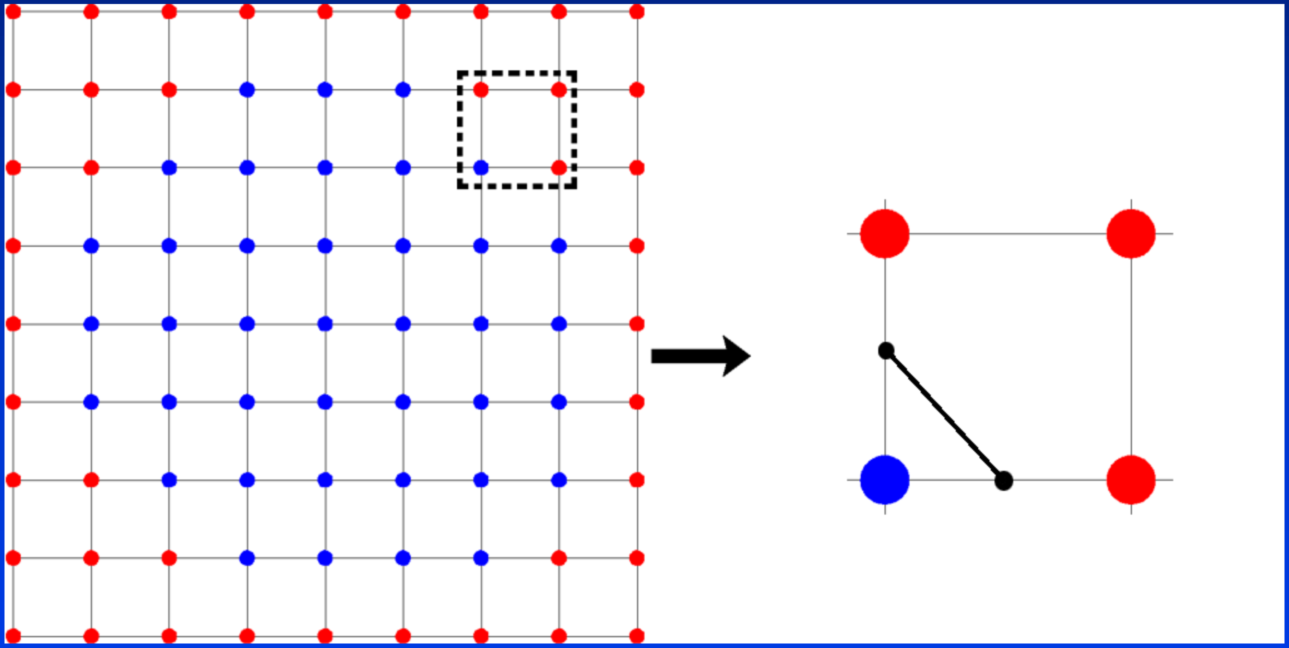
2D Polygon Generation



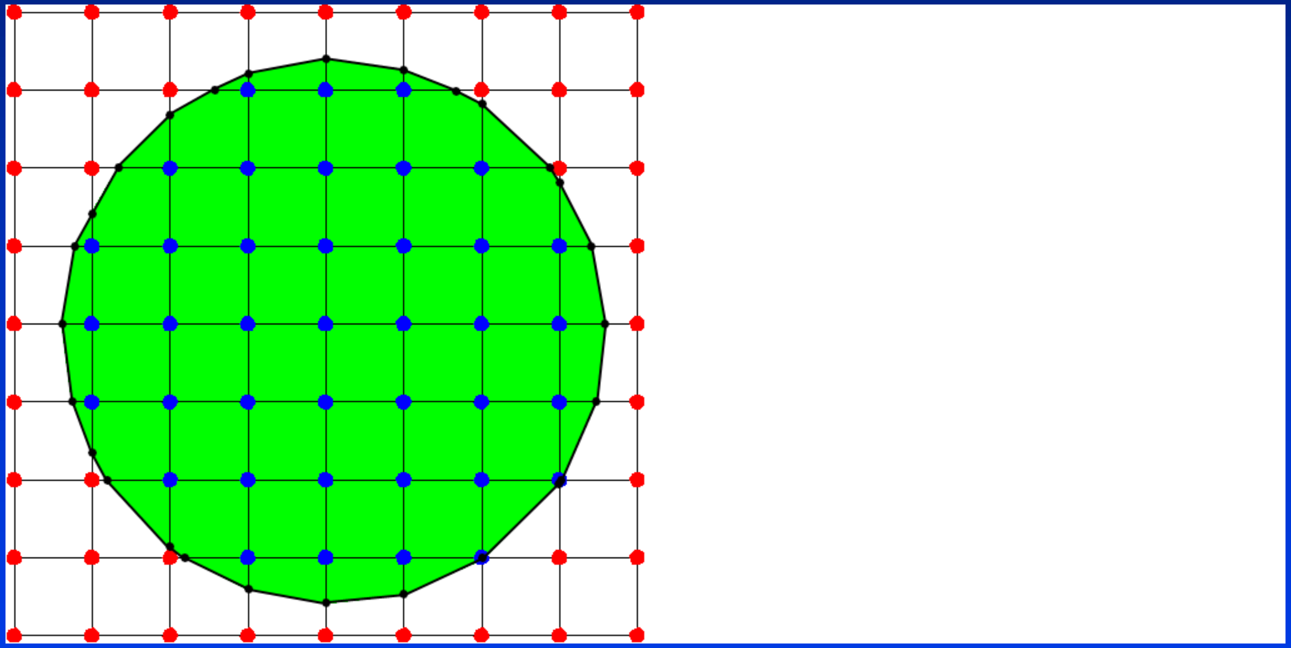
2D Polygon Generation



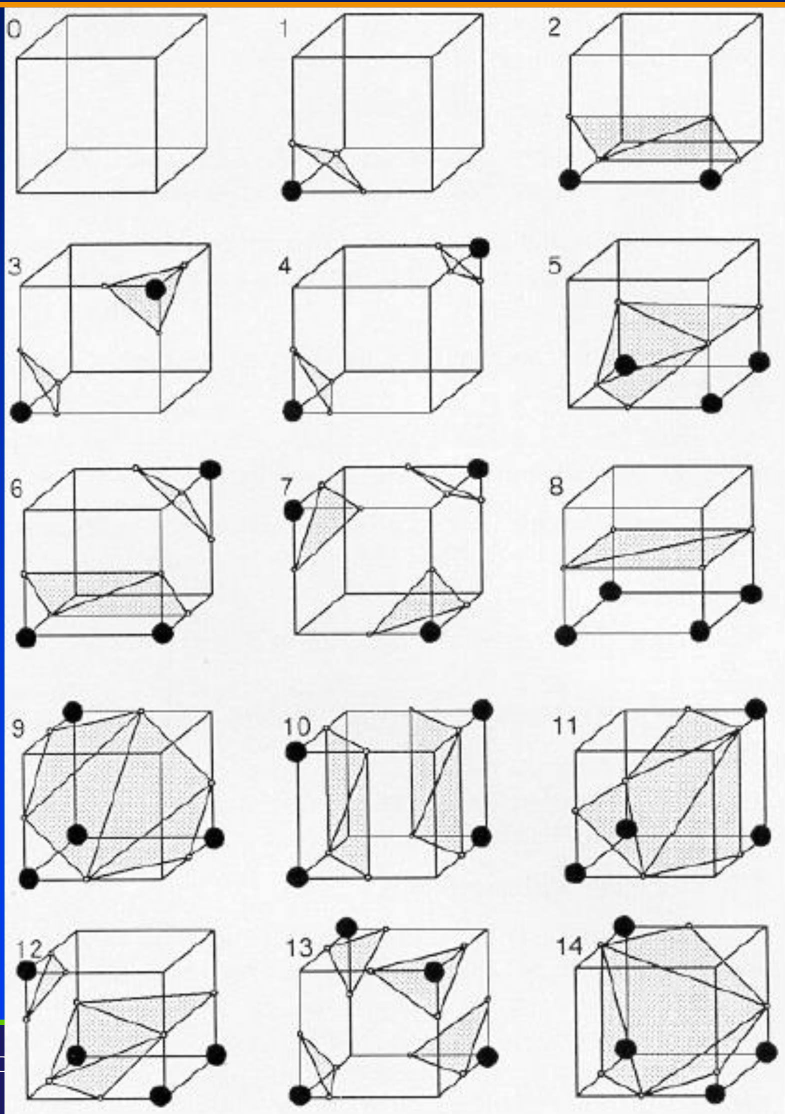
2D Polygon Generation

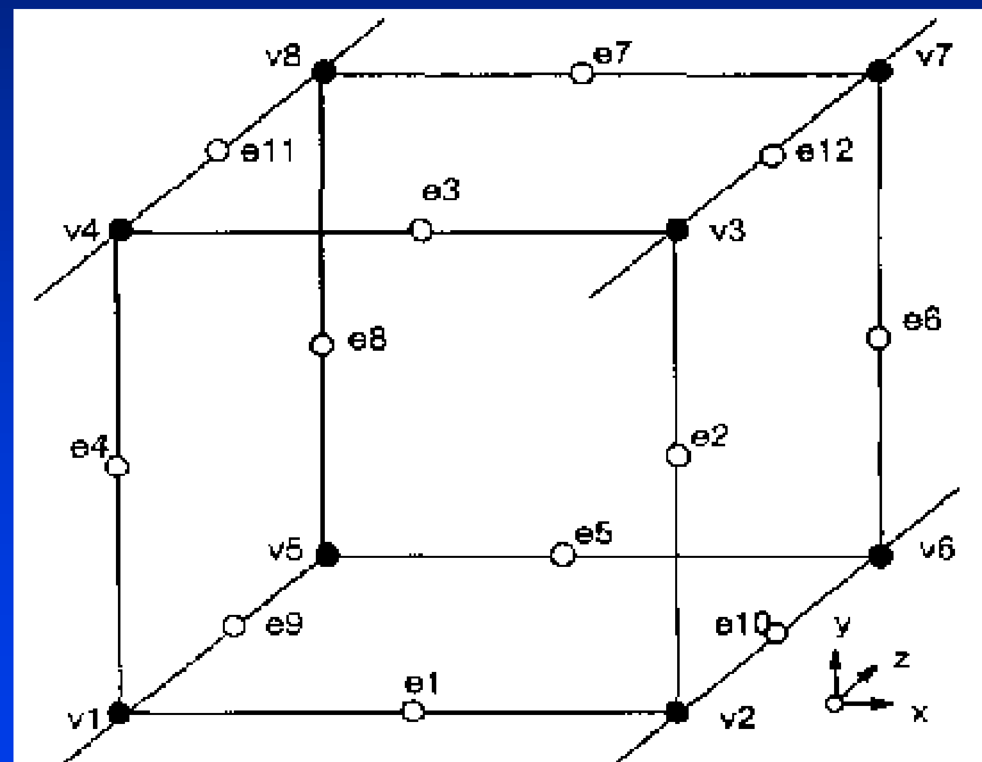


2D Polygon Generation

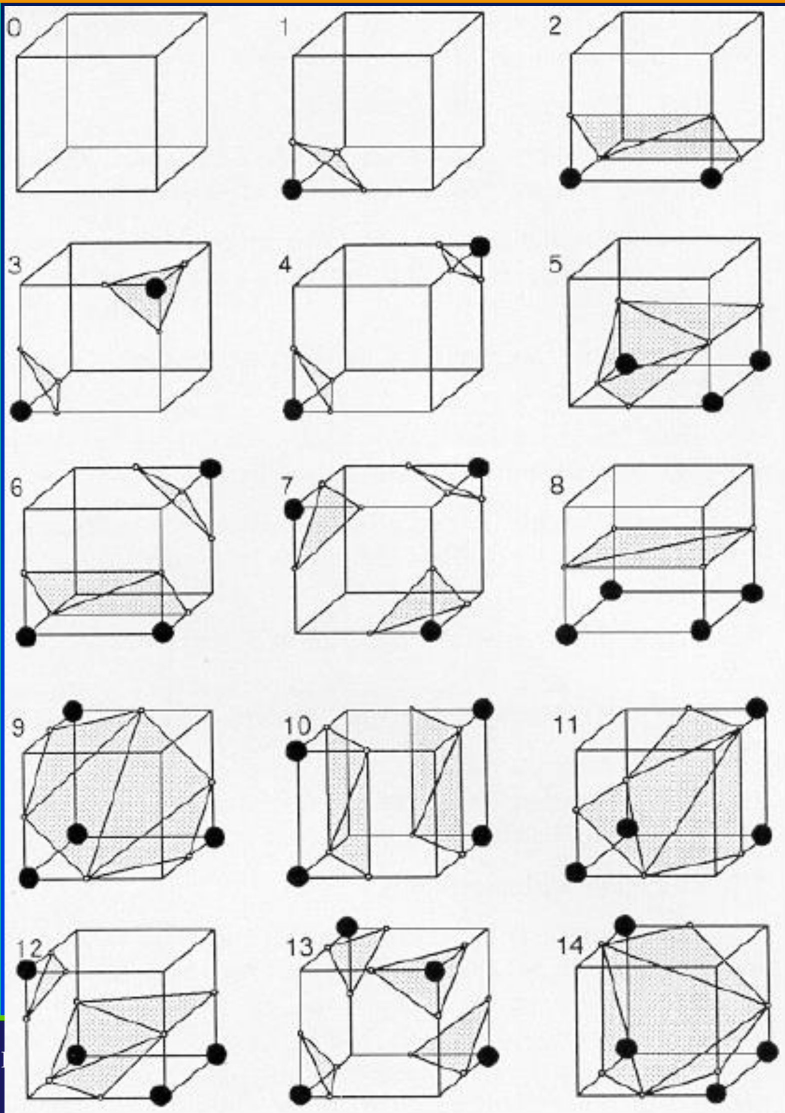


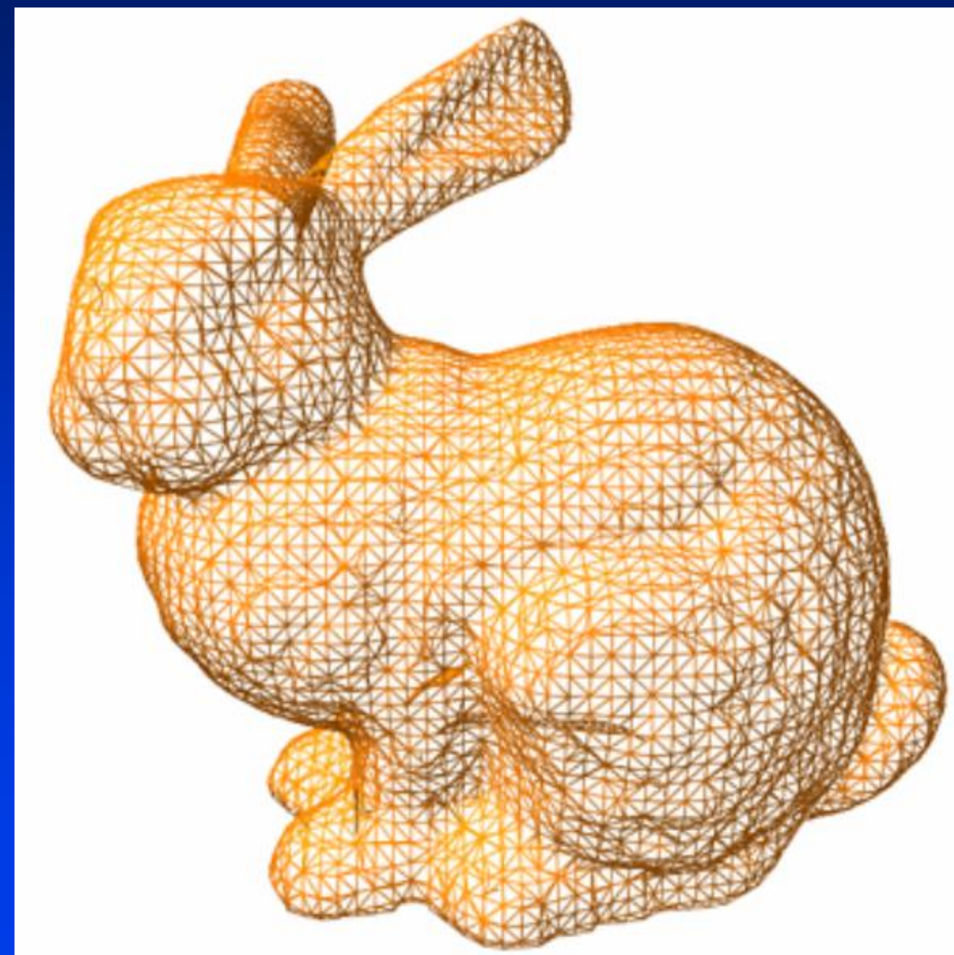
3D Polygon Generation





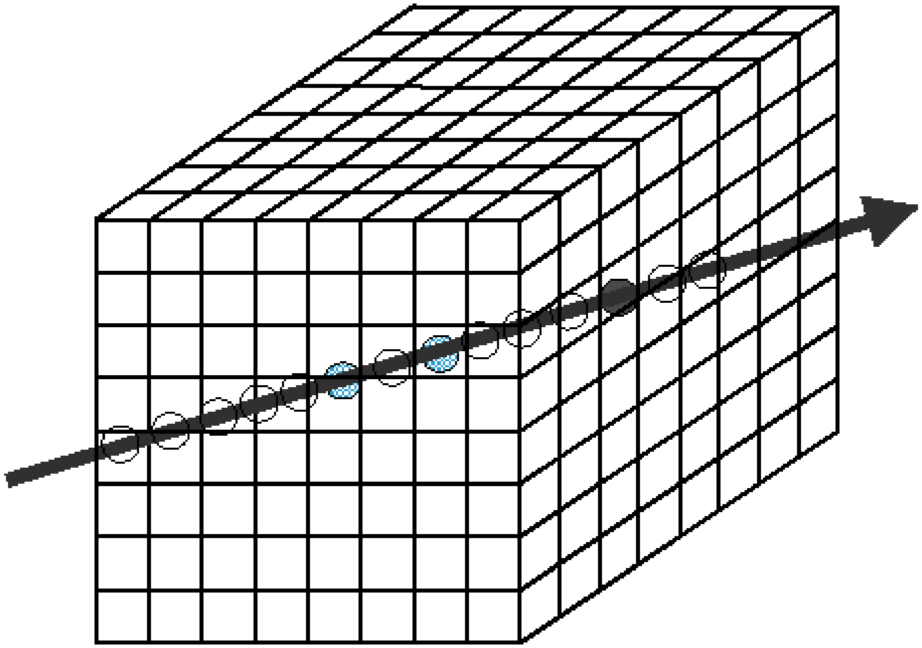
3D Polygon Generation

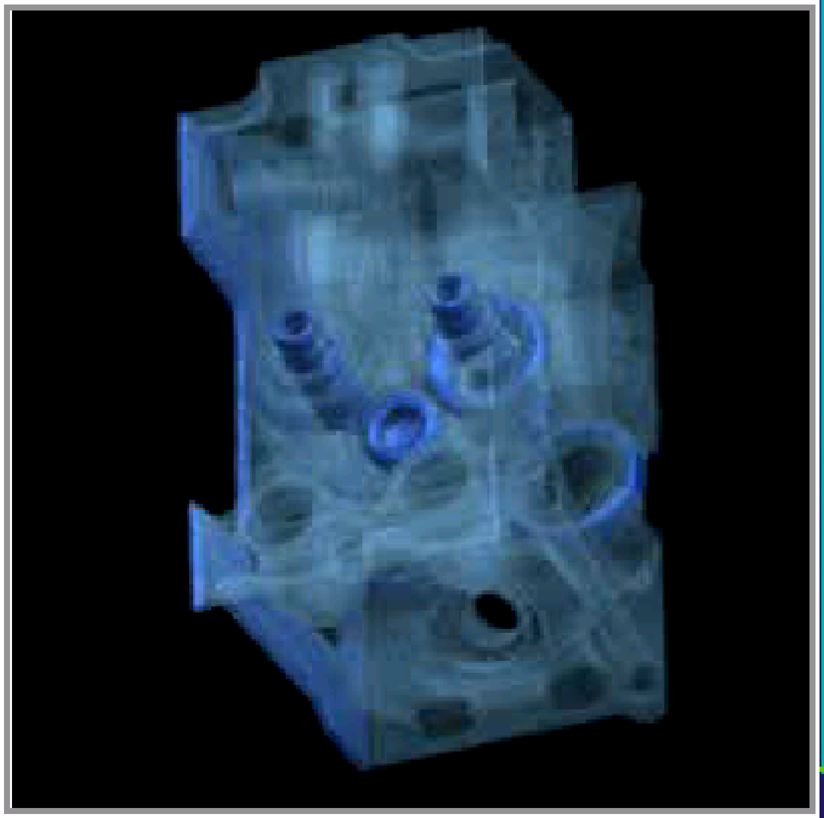




Voxel Display

- **Ray-casting**
  - Integrate density along rays through pixels





Voxels

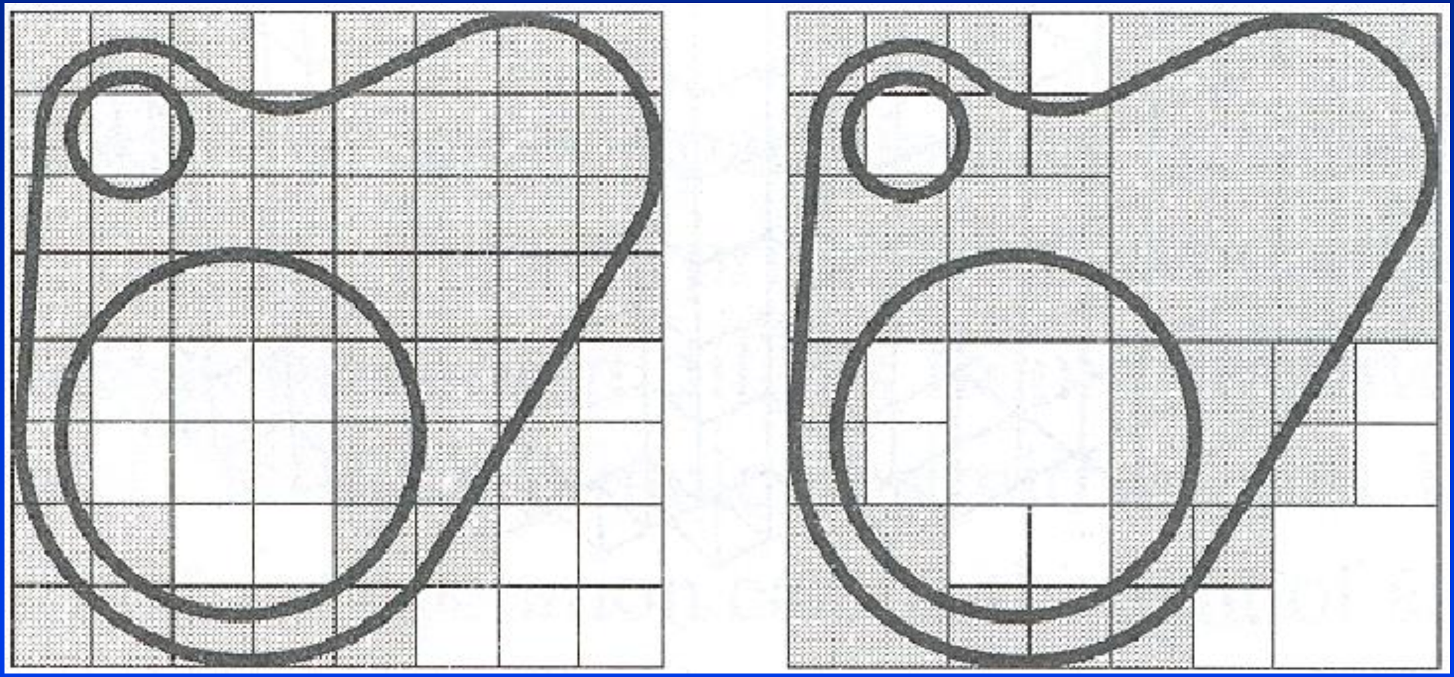
- **Advantages**
  - Simple, intuitive, unambiguous
  - Same complexity for all objects
  - Natural acquisition for some applications
  - Trivial Boolean operations
- **Disadvantages**
  - Approximation, not accurate
  - Large storage requirements
  - Expensive display

Solid Modeling Representation

- **Quadtrees & Octrees**

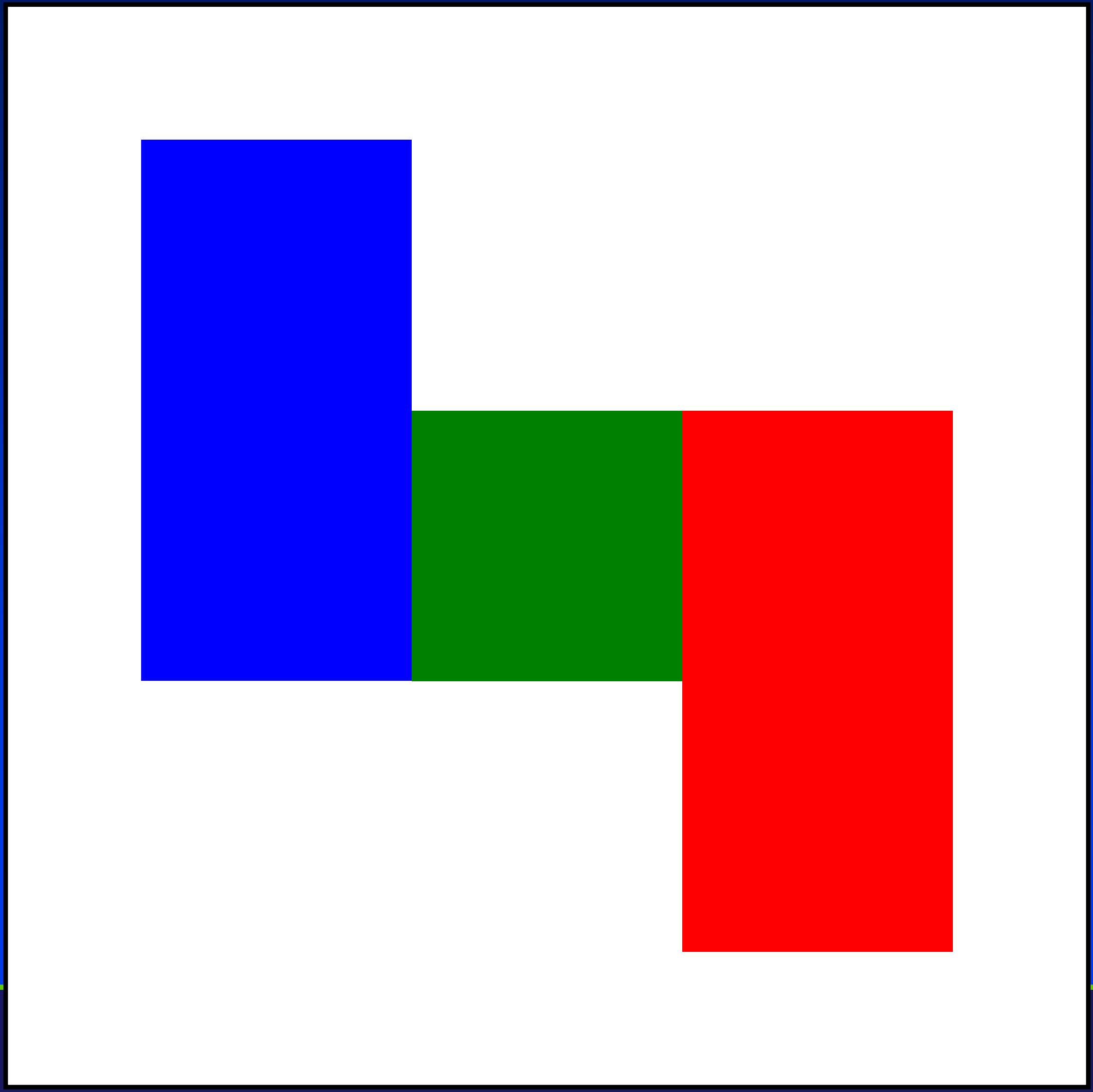
Quadtrees & Octrees

- **Refine resolution of voxels hierarchically**
  - More concise and efficient for non-uniform objects

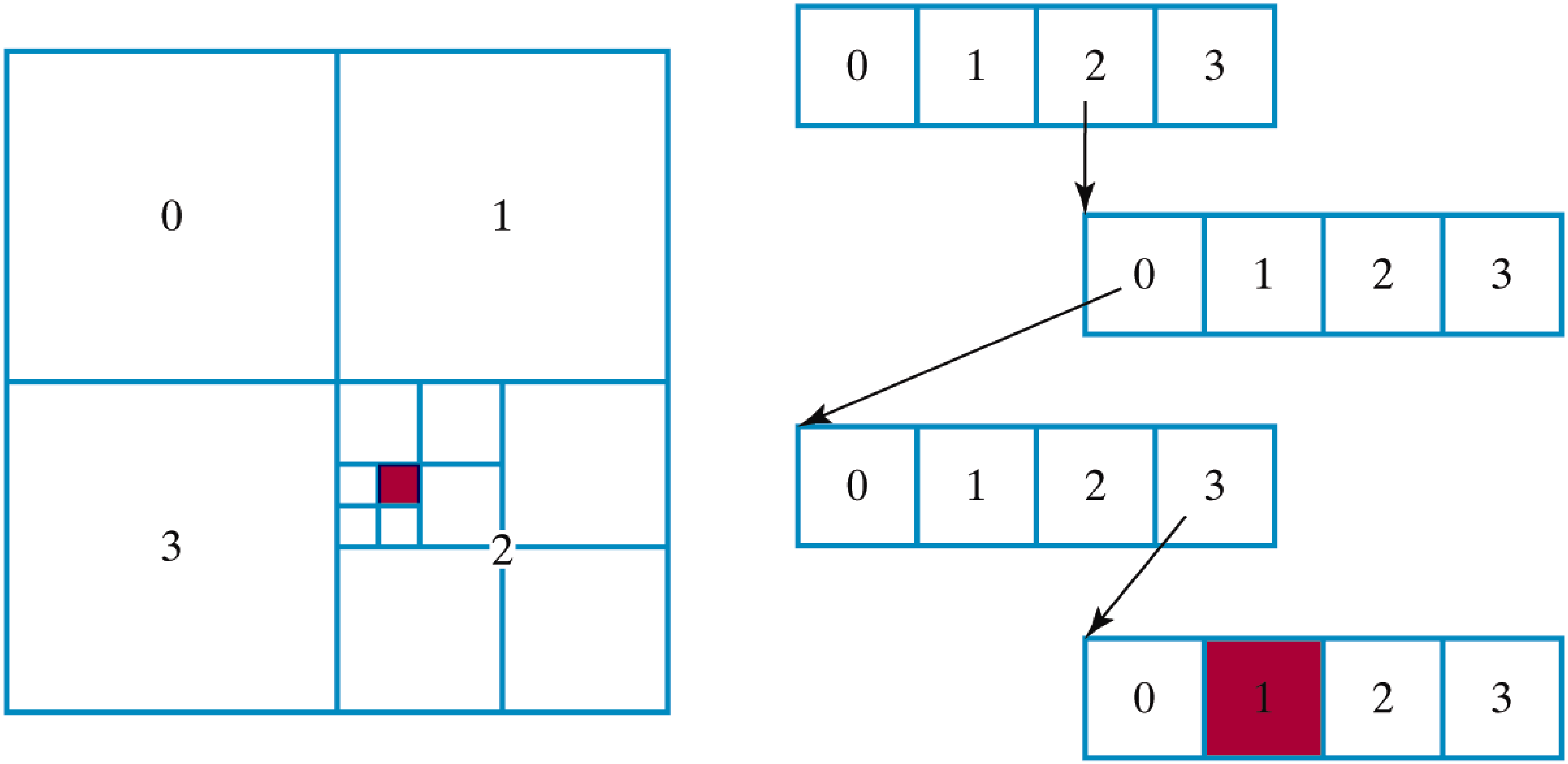


Uniform Voxel Quadtree

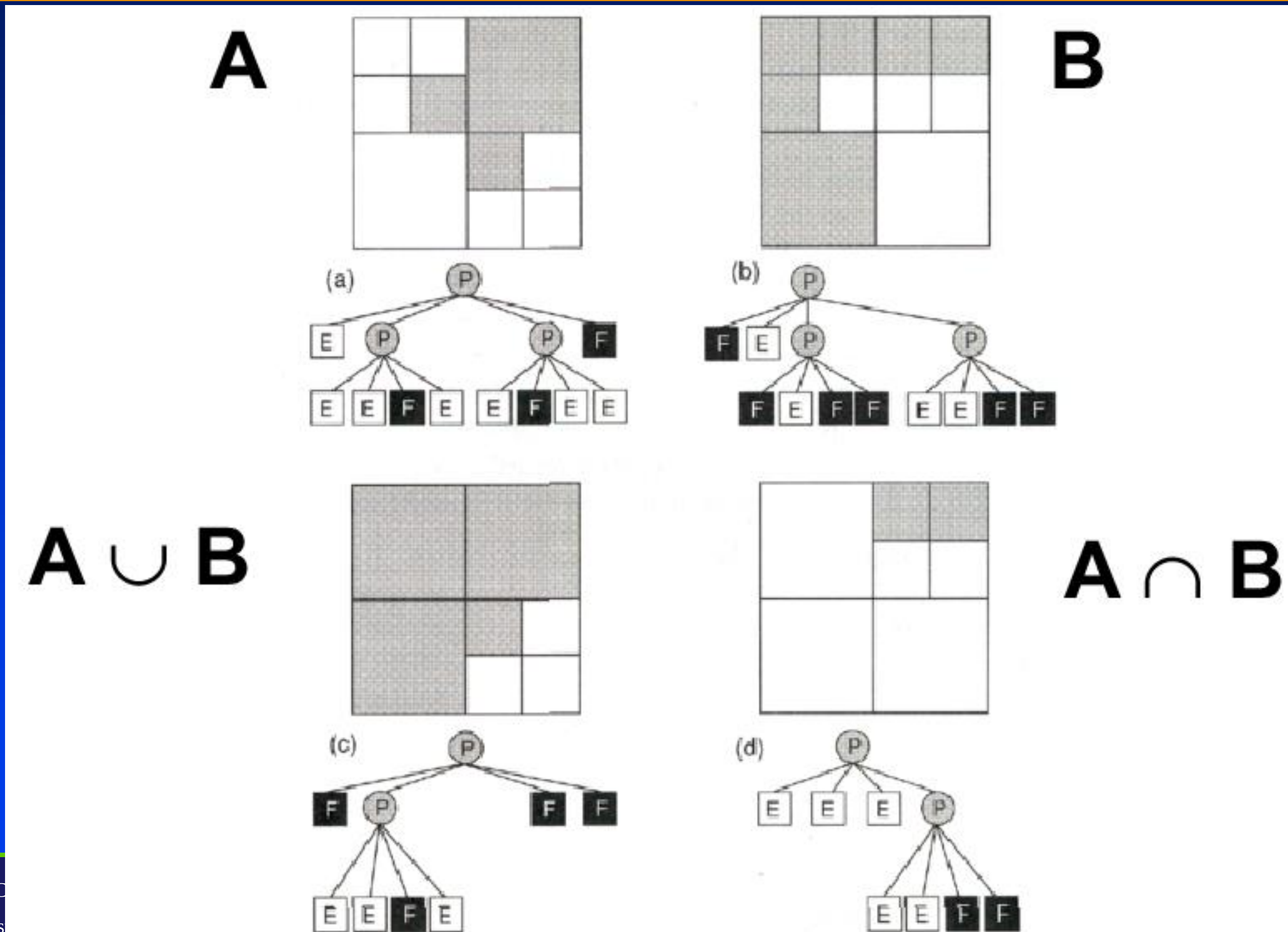
Quadtree



Quadtree



Quadtree Boolean Operations

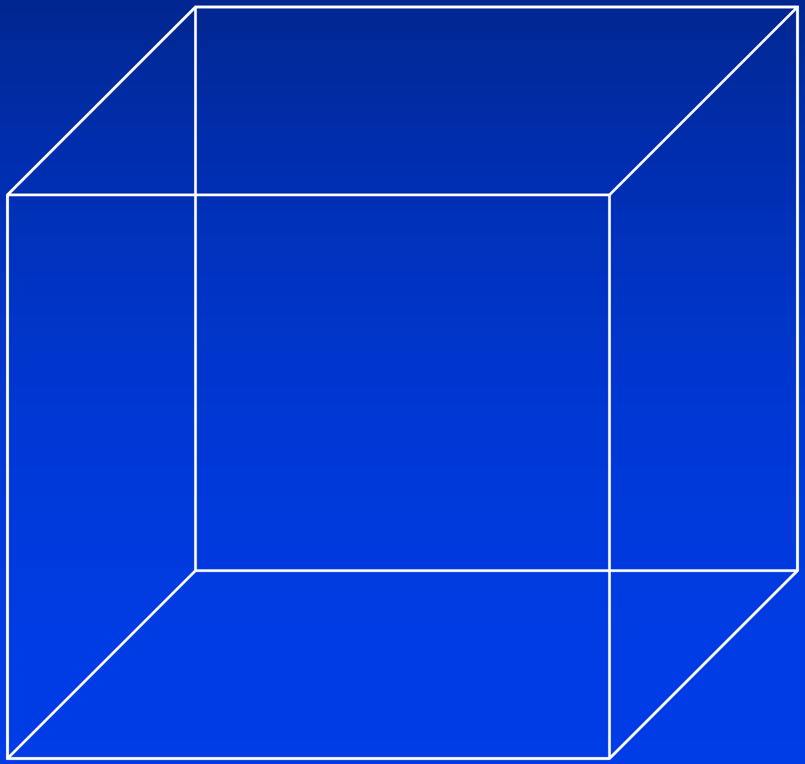


Octrees & Quadtrees

- Octrees are based on a two-dimensional representation scheme called **quadtree** encoding
- Quadtree encoding divides a square region of space into four equal areas until *homogeneous regions* are found
- These regions can then be arranged in a tree

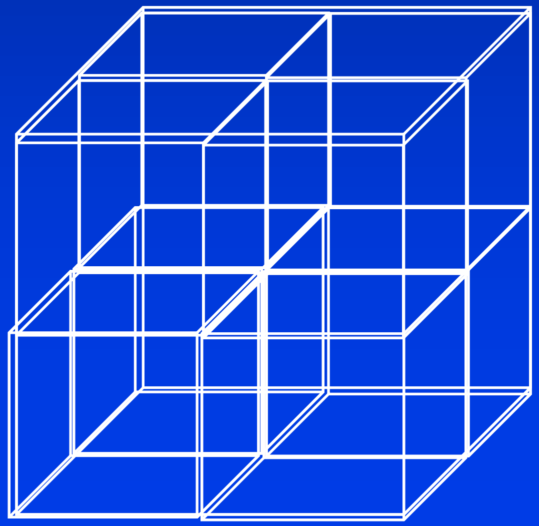
Octrees

- Model space as a tree with 8 children
- Nodes can be 3 types
  - Interior Nodes
  - Solid
  - Empty

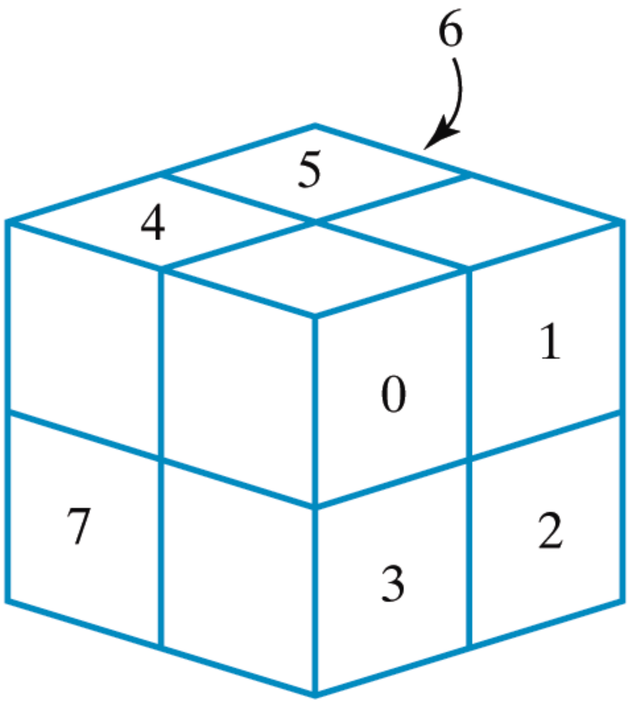


Octrees

- Model space as a tree with 8 children
- Nodes can be 3 types
  - Interior Nodes
  - Solid
  - Empty



Octrees

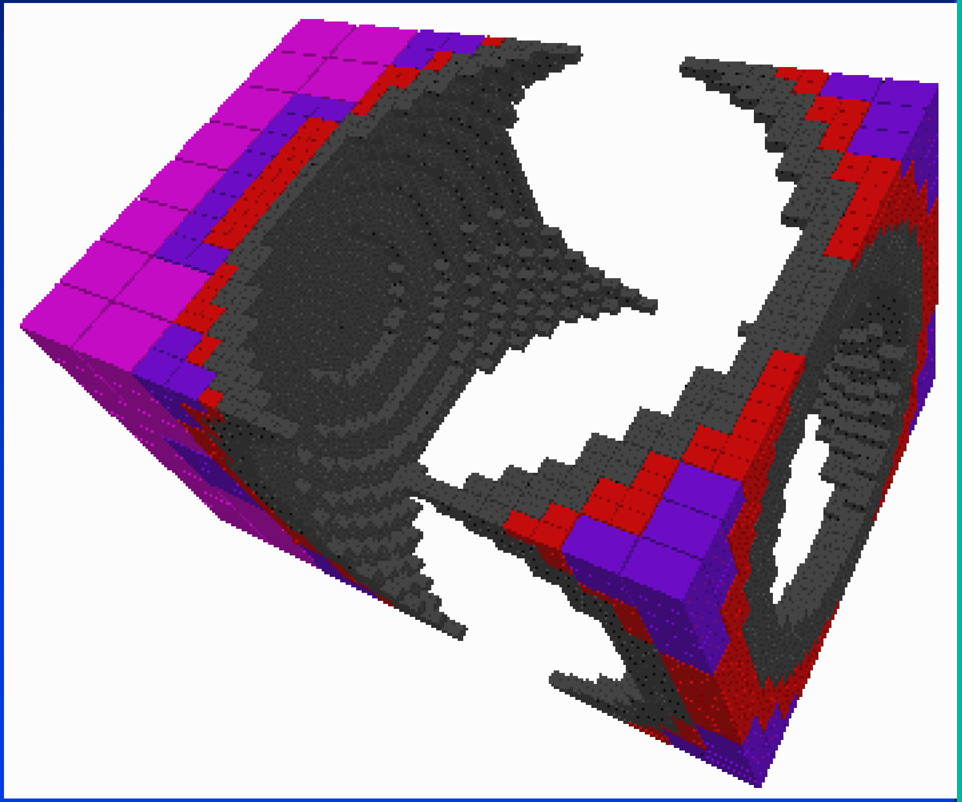


Region of a
Three-Dimensional
Space

Data Elements
in the Representative
Octree Node

Octrees

- Octrees are hierarchical tree structures used to represent solid objects
- Octrees are particularly useful in applications that require cross sectional views – for example medical applications
- Octrees are typically used when the interior of objects is important



Octrees

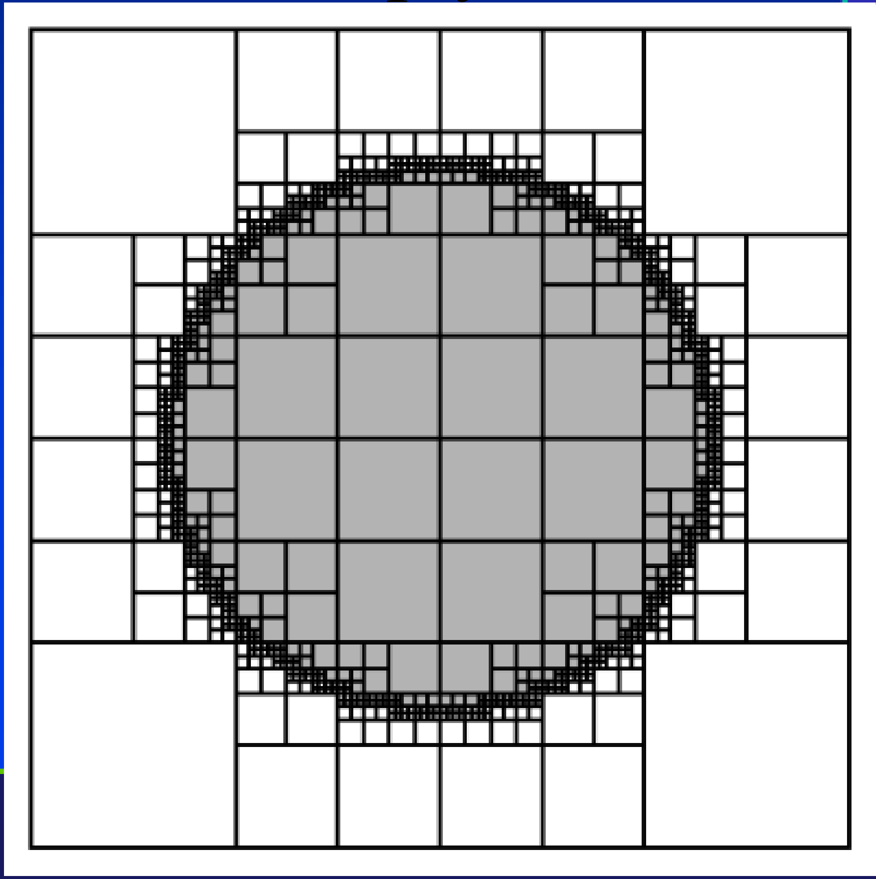
- Quadtree encodings provide considerable savings in storage when large colour areas exist in a region of space
- An octree takes the same approach as quadtrees, but divides a cube region of 3D space into octants
- Each region within an octree is referred to as a **volume element** or **voxel**
- Division is continued until homogeneous regions are discovered

Octrees

- In 3 dimensions regions can be considered to be homogeneous in terms of color, material type, density or any other physical characteristics
- Voxels also have the unique possibility of being *empty*

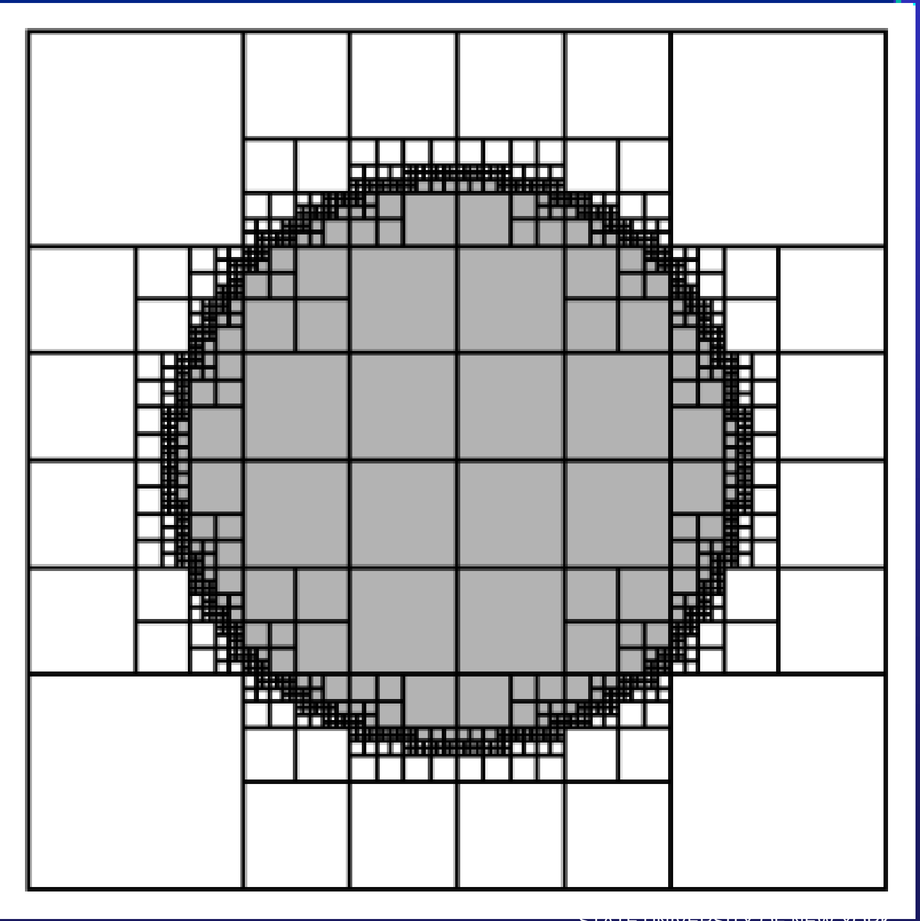
Building Octrees

- If cube completely inside, return solid node
- If cube completely outside, return empty node
- Otherwise recursion until maximum depth reached



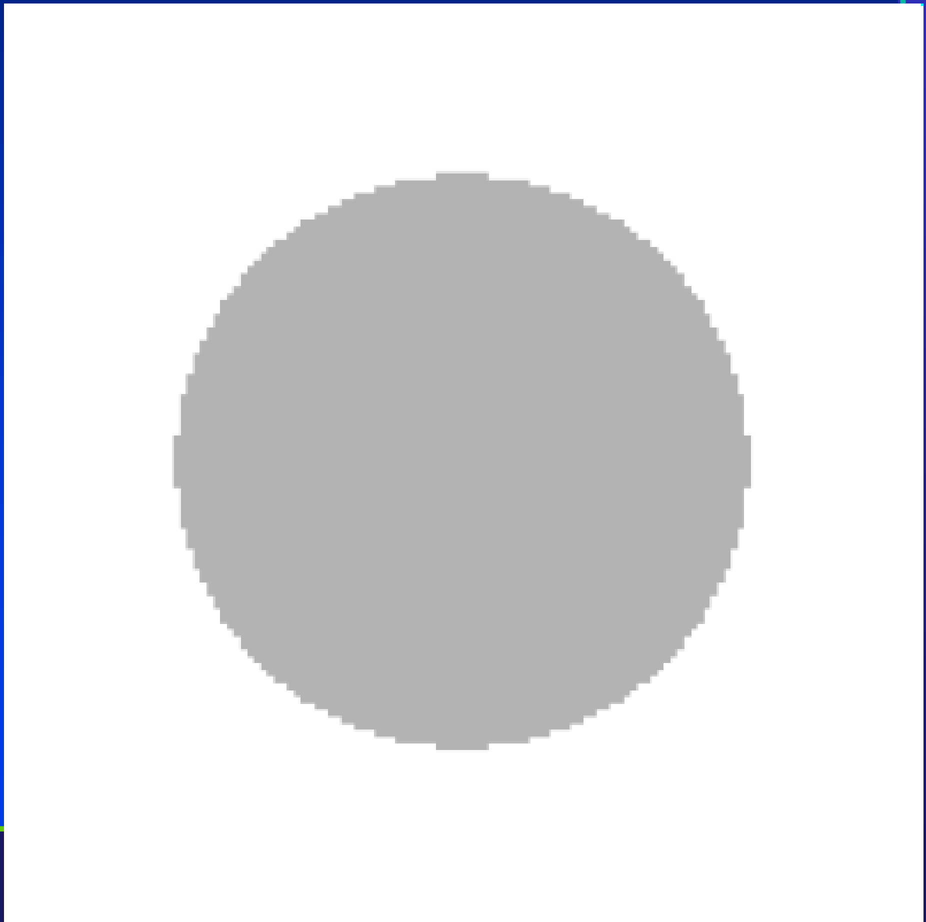
Octrees

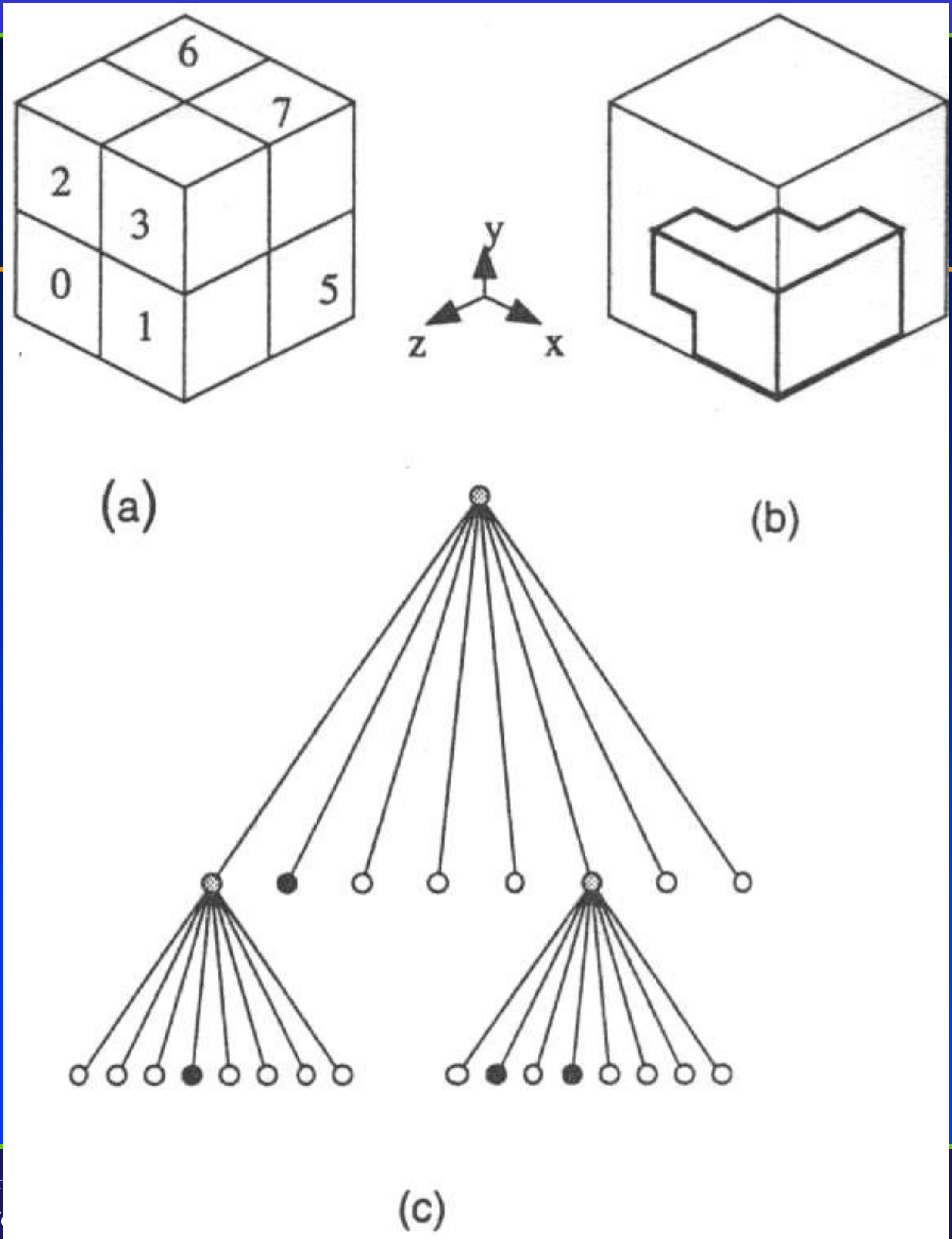
- Advantages
  - Storage space proportional to surface area
  - Inside/Outside trivial
  - Volume trivial
  - CSG relatively simple
  - Can approximate any shape
- Disadvantages
  - Blocky appearance



Octrees

- Advantages
  - Storage space proportional to surface area
  - Inside/Outside trivial
  - Volume trivial
  - CSG relatively simple
  - Can approximate any shape
- Disadvantages
  - Blocky appearance



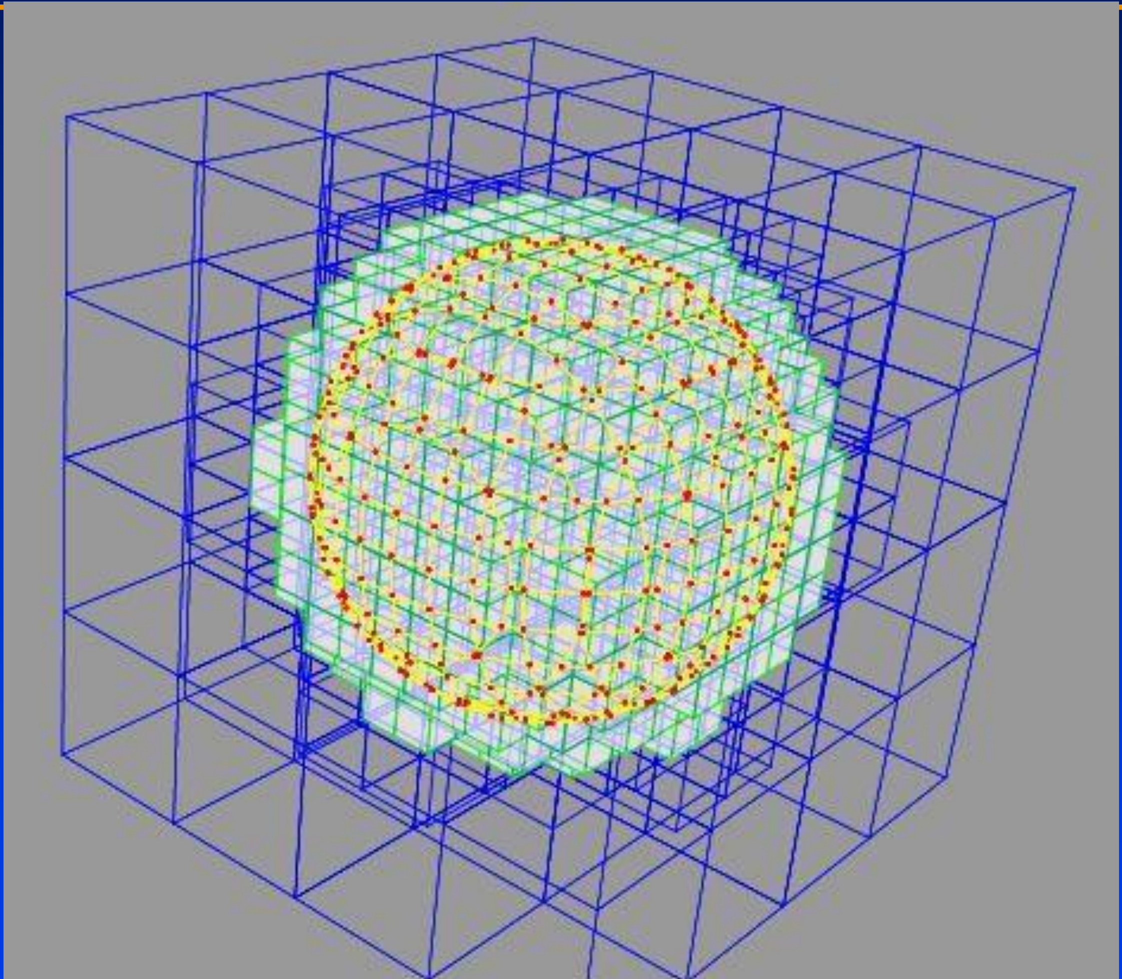


Octree Example

Octree Data Structure

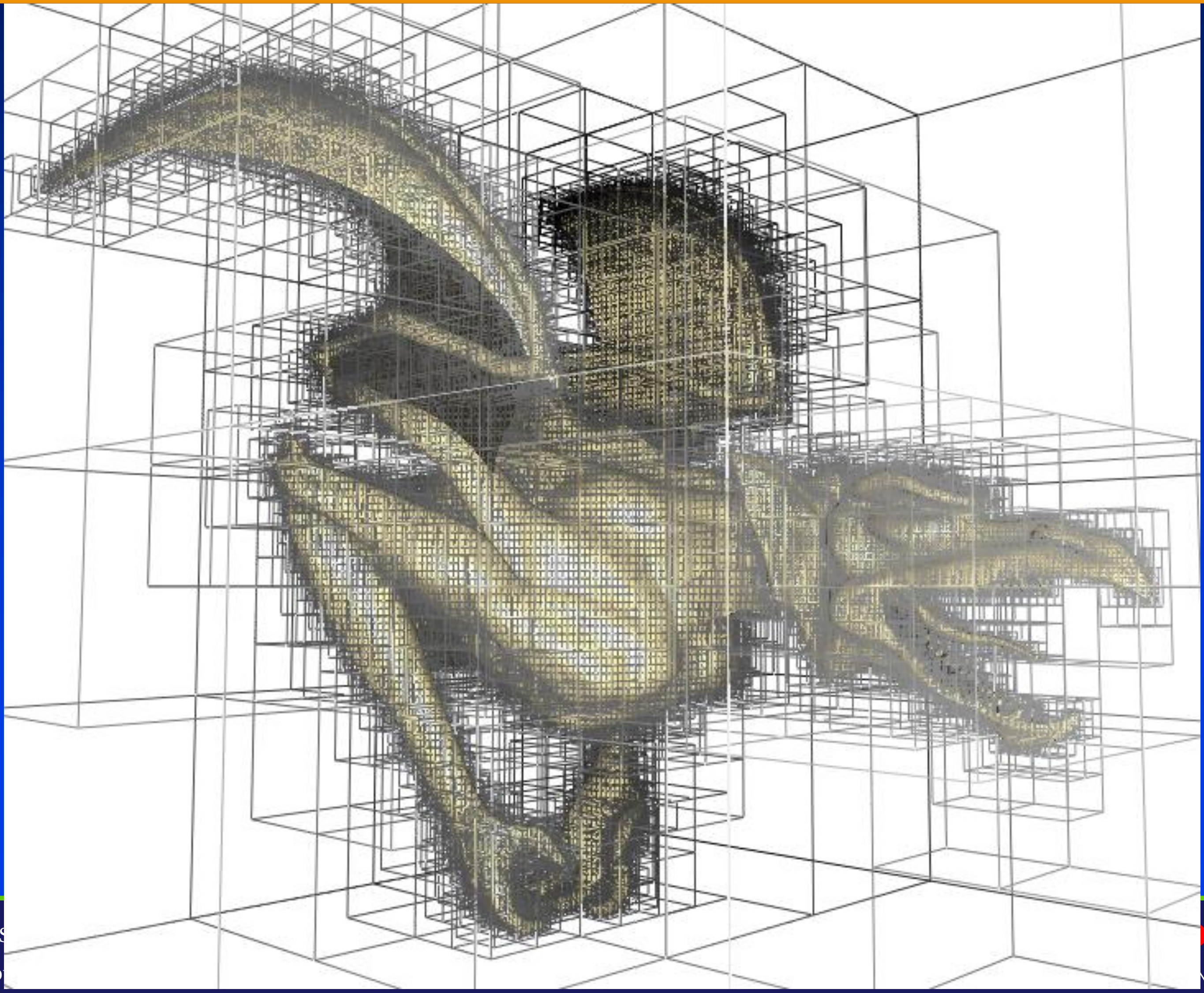
```
struct octreeroot
{
   float xmin, ymin, zmin;     /* space of interest */
   float xmax, ymax, zmax;
   struct octree  *root;/* root of the tree */
};

struct octree
{
   char  code;                 /* BLACK, WHITE, GRAY */
   struct octree  *oct[8];     /* pointers to octants, present if GRAY */
};
```

Octree Examples



Octree containing pieces of an implicitly defined sphere; within each terminal node surface vertices are computed and connected to form a polygon

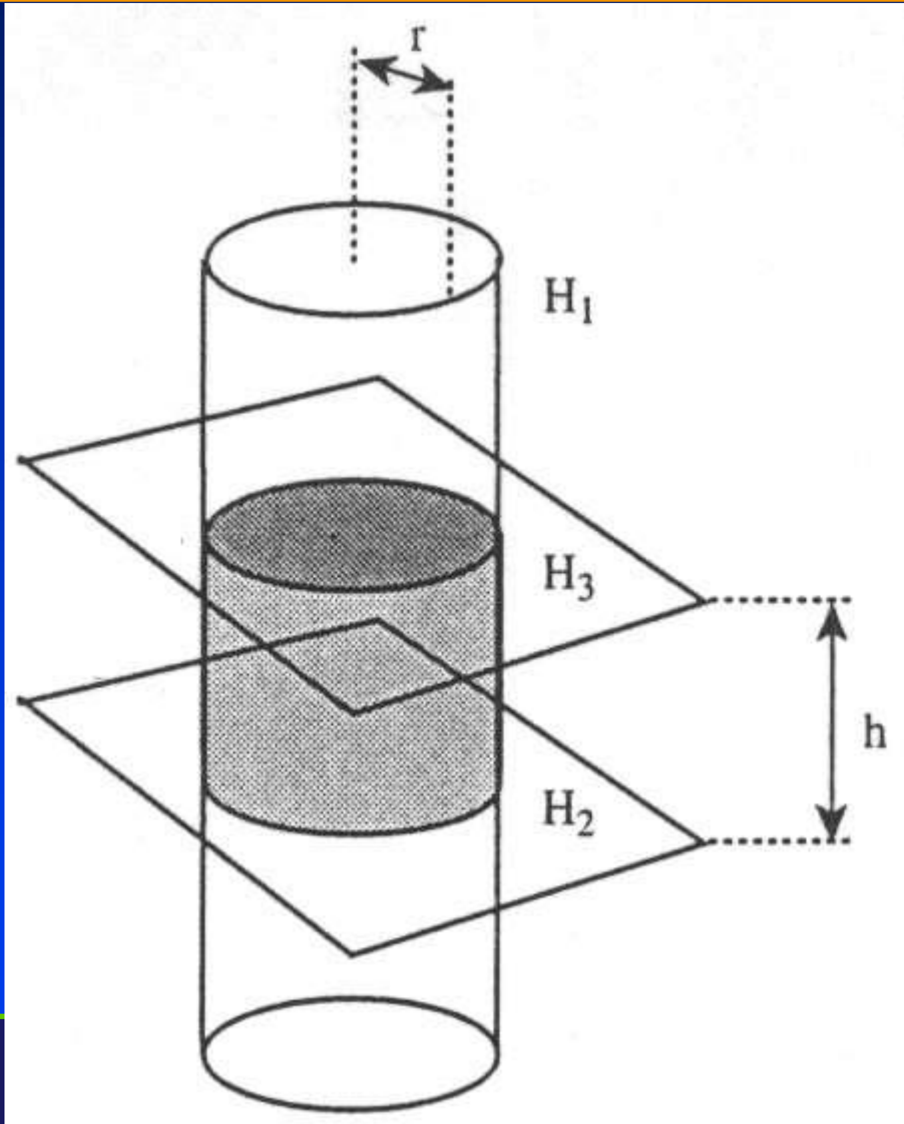
Octree Examples



Solid Modeling Representation

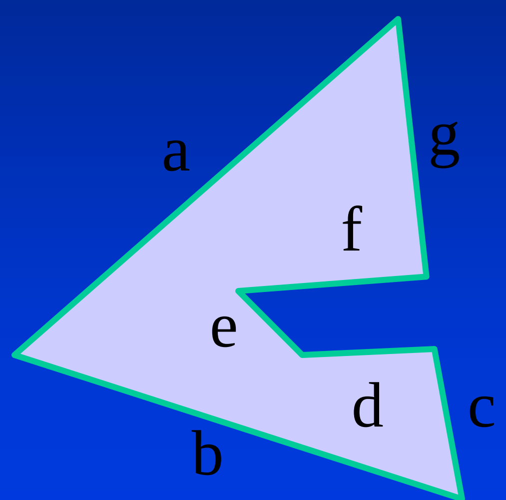
- **Binary Space Partitions**

Half Space Model

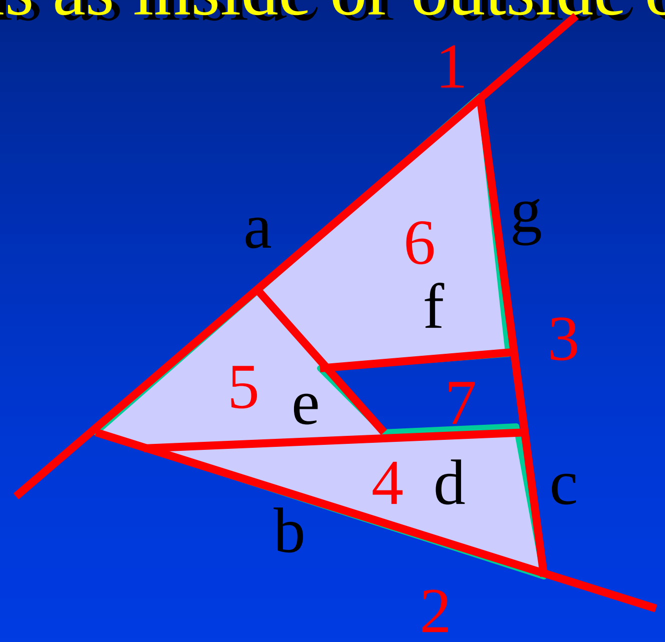


Binary Space Partitions (BSPs)

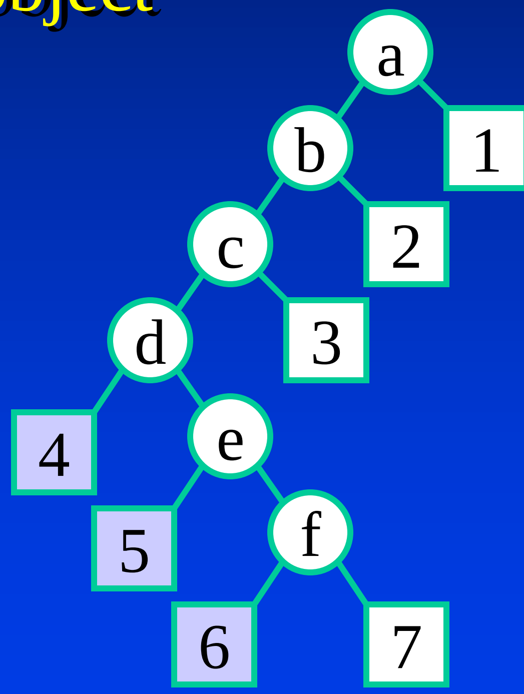
- **Recursive partition of space by Planes**
  - Mark leaf cells as inside or outside object



Object



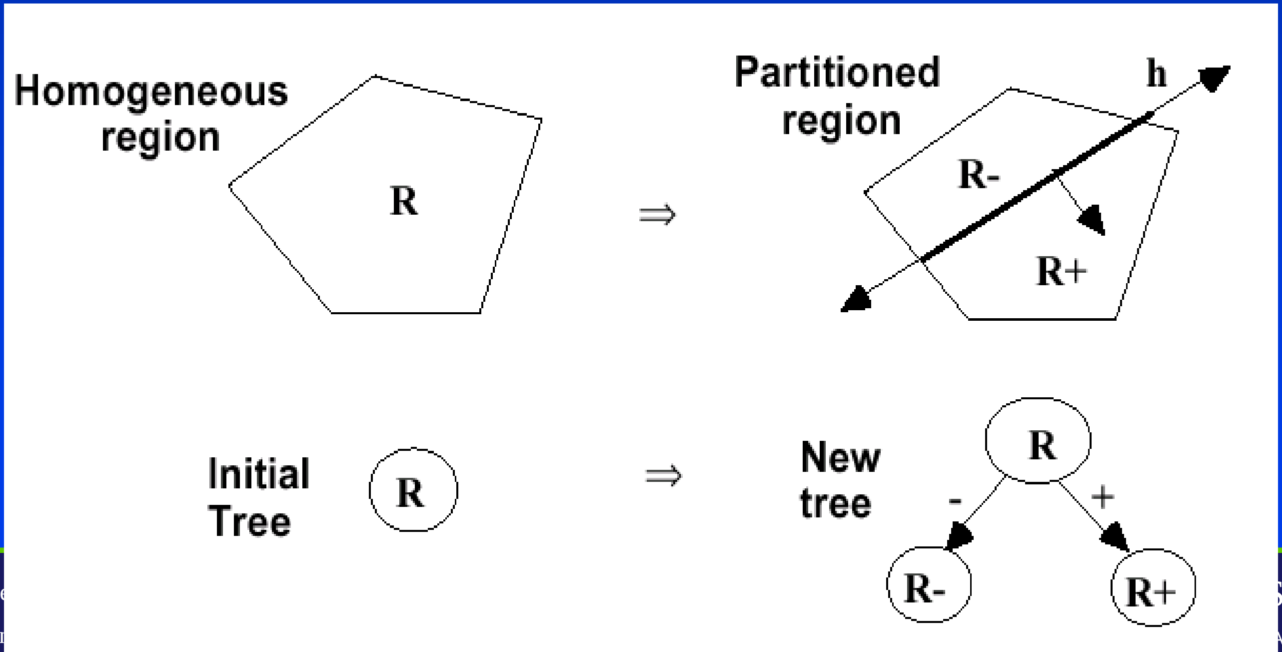
Binary Spatial Partition



BSP Tree

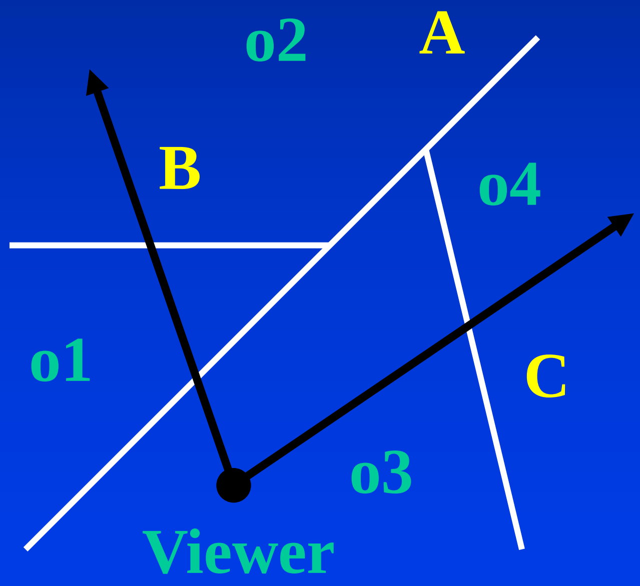
BSP Fundamentals

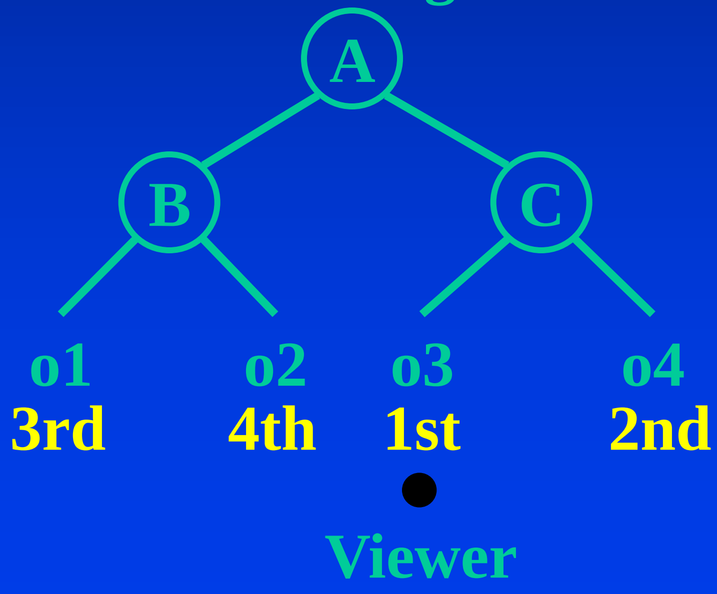
- **Single geometric operation**
  - Partition a convex region by a hyper-plane
- **Single combinatorial operation**
  - Two child nodes added as leaf nodes



BSP Display

- **Visibility Ordering**
  - Determine on which side of plane the viewer lies
    - Near-subtree -> polygons on split -> far-subtree





Summary

	Voxels	Octree	BSP	CSG
Accurate	No	No	Some	Some
Concise	No	No	No	Yes
Affine Invariant	No	No	Yes	Yes
Easy Acquisition	Some	Some	No	Some
Guaranteed Validity	Yes	Yes	Yes	No
Efficient Boolean Operations	Yes	Yes	Yes	Yes
Efficient Display	No	No	Yes	No

New Solid Modeling Techniques:
(Sketch-Based Solid Modeling with BlobTrees)

Implicit Representation of Shape

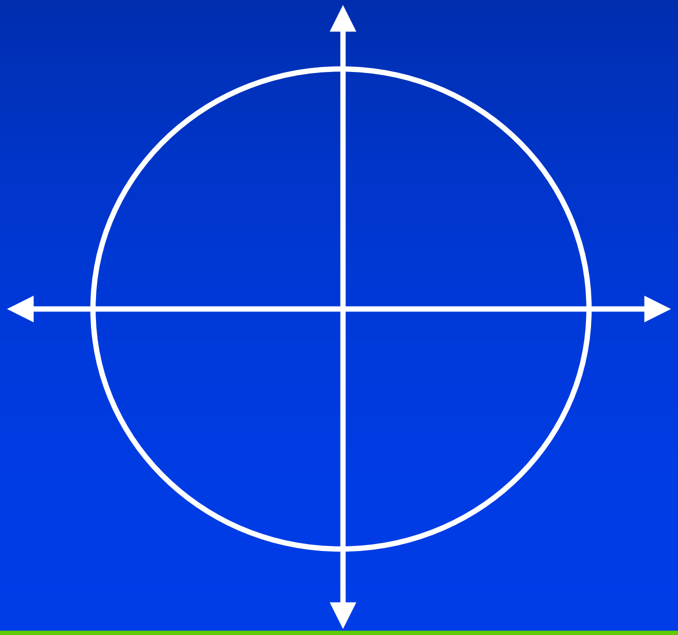
- Shape described by solution to $f(x)=c$

$$f(x,y)=x^2+y^2-9$$

Implicit Representation of Shape

- Shape described by solution to *f*(*x*)=*c*

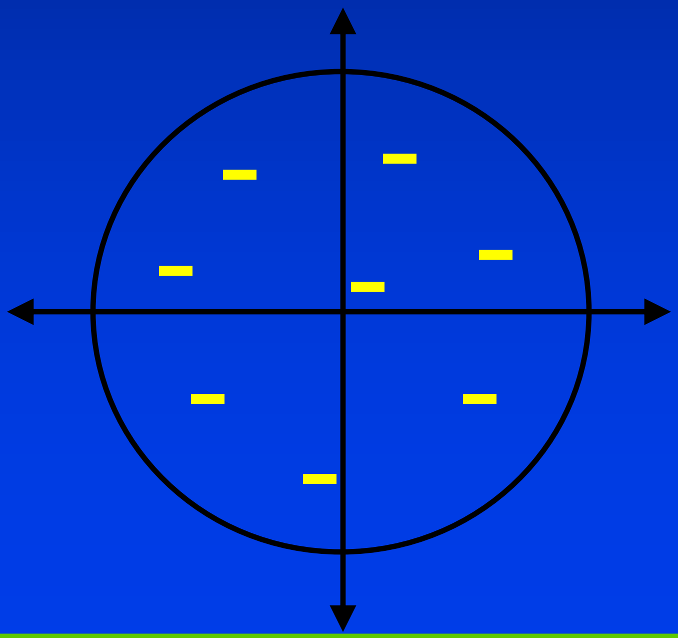
$$f(x,y) = x^2 + y^2 - 9$$



Implicit Representation of Shape

- Shape described by solution to $f(x)=c$

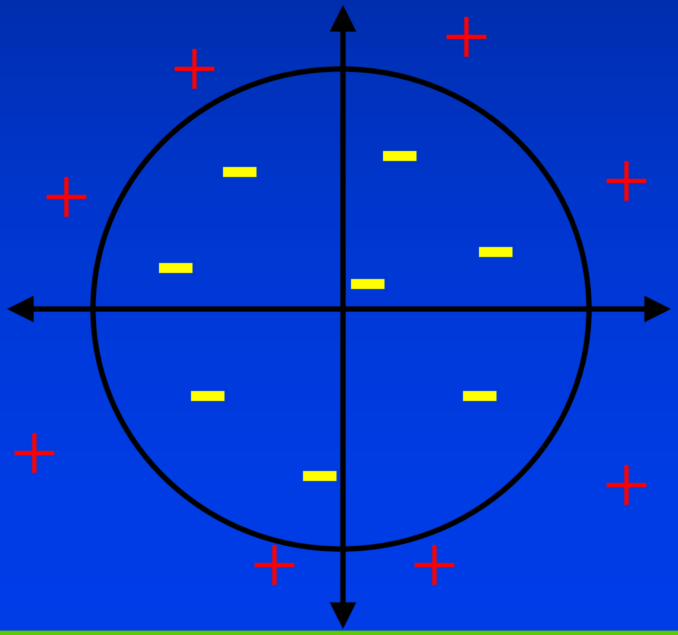
$$f(x,y)=x^2+y^2-9$$



Implicit Representation of Shape

- Shape described by solution to $f(x)=c$

$$f(x,y) = x^2 + y^2 - 9$$

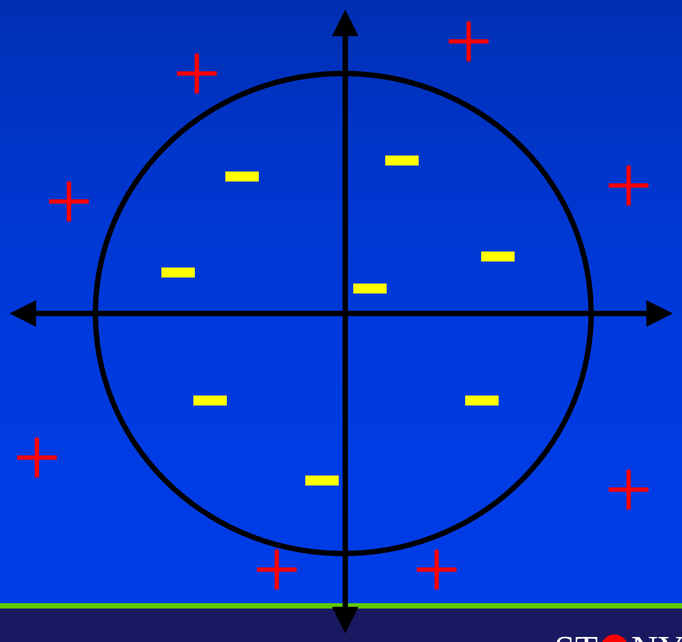


Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations

Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations

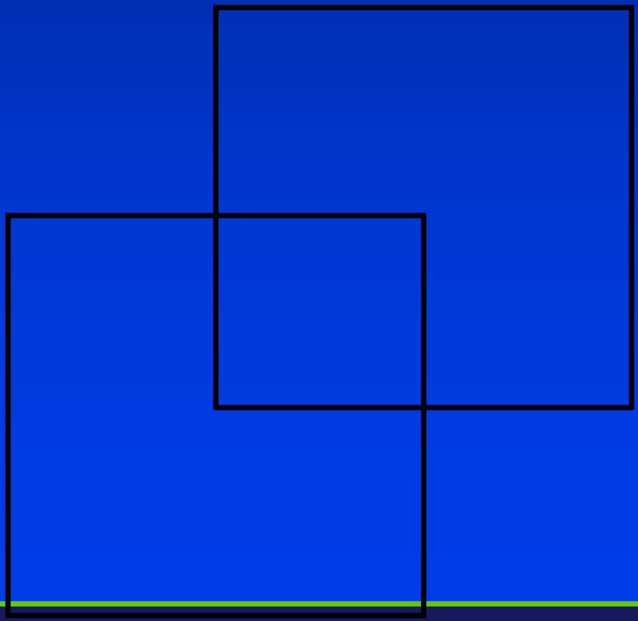


Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations

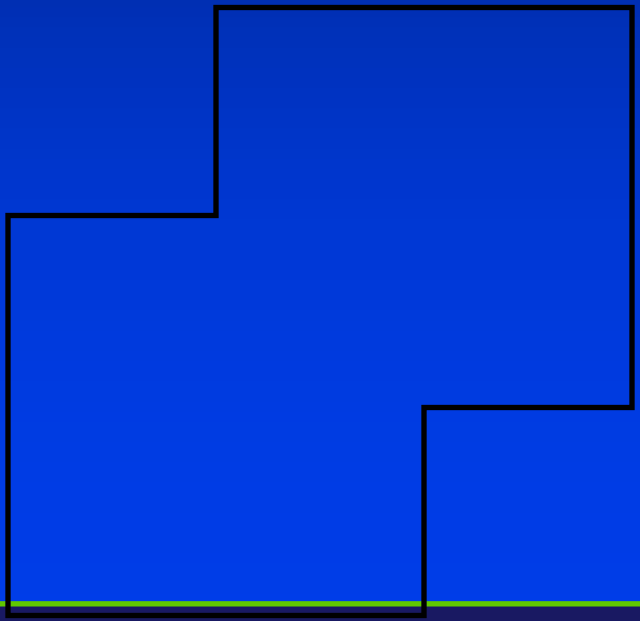
Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations
  - Union



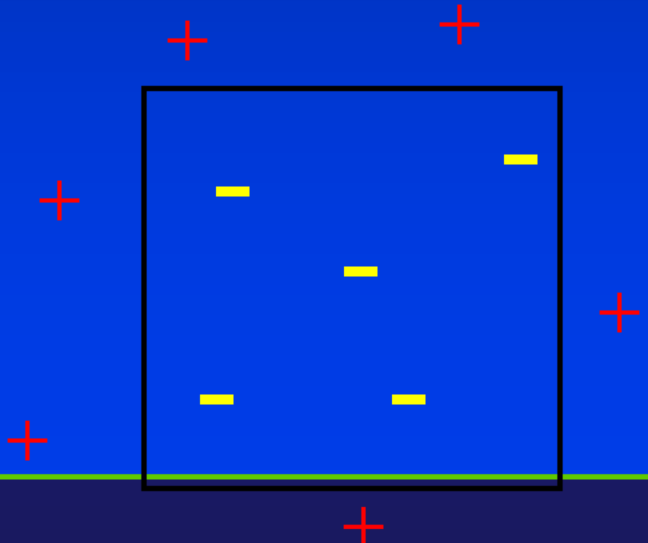
Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations
  - Union



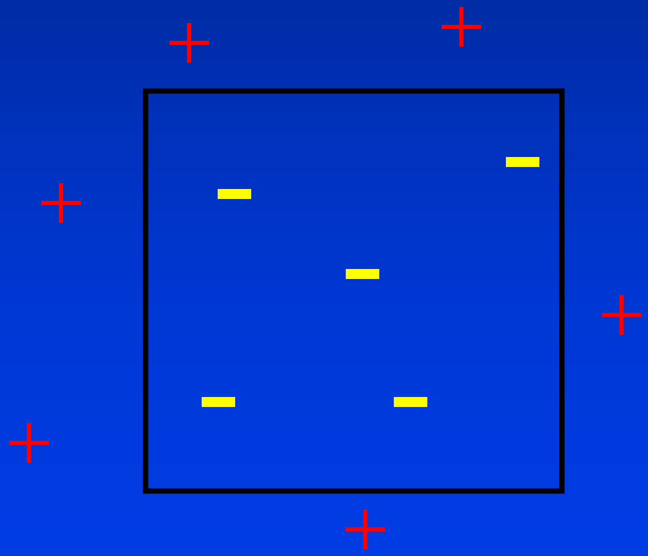
Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations
  - Union



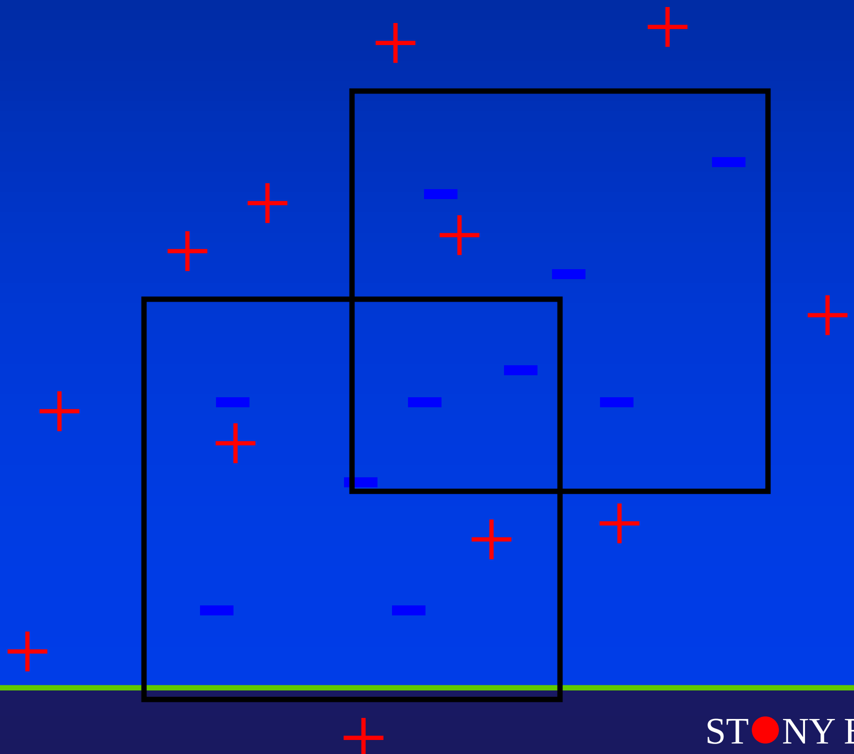
Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations
  - Union



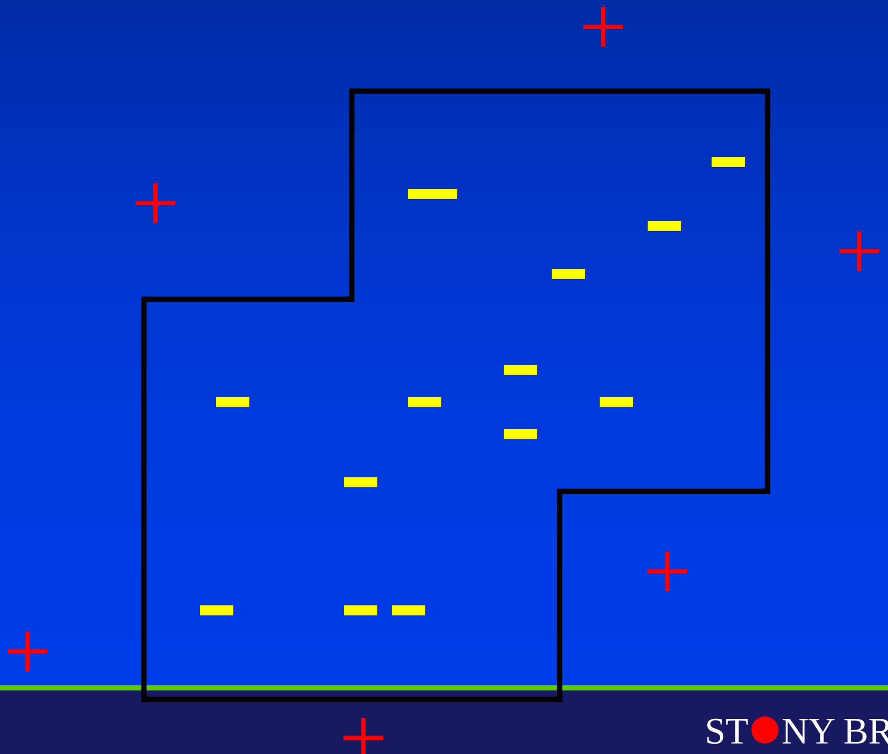
Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations
  - Union



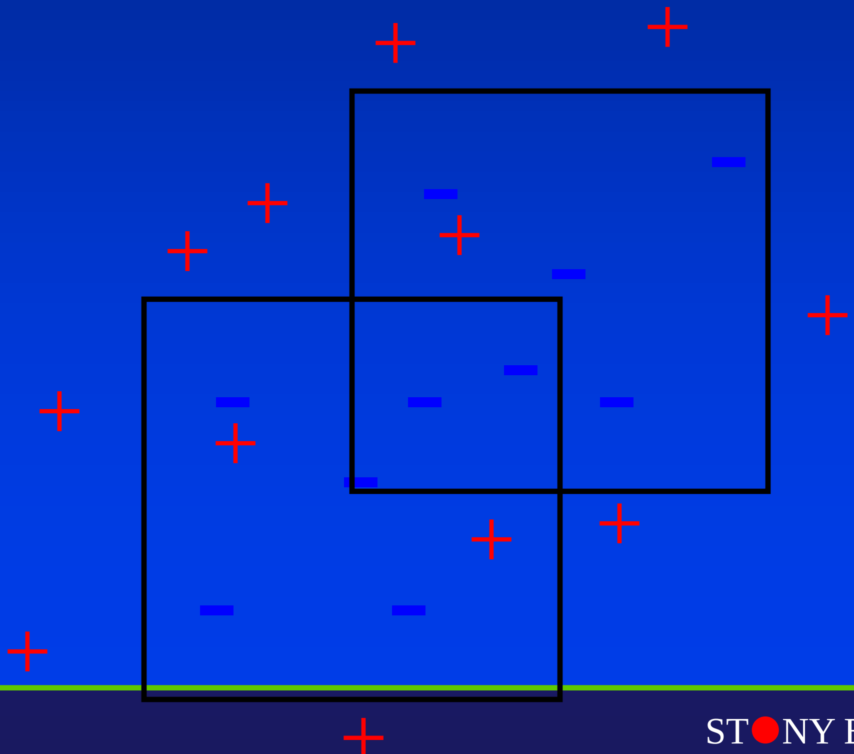
Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations
  - Union



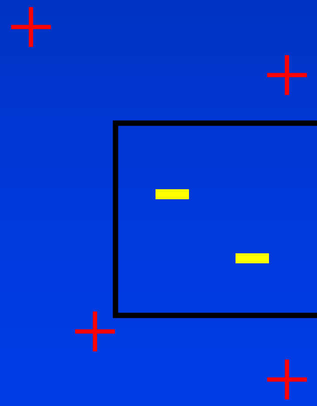
Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations
  - Union
  - Intersection



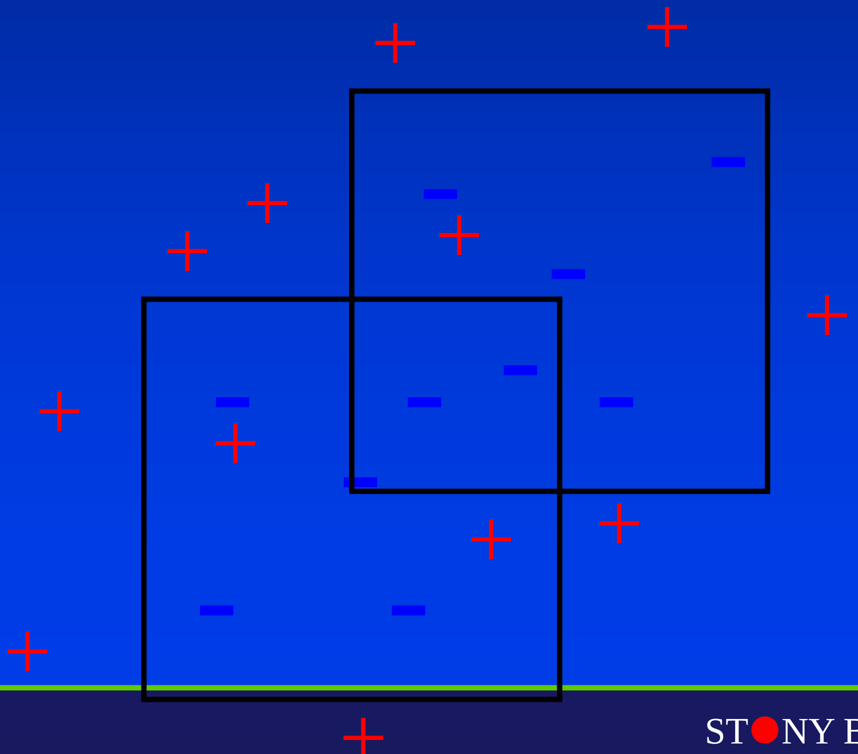
Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations
  - Union
  - Intersection



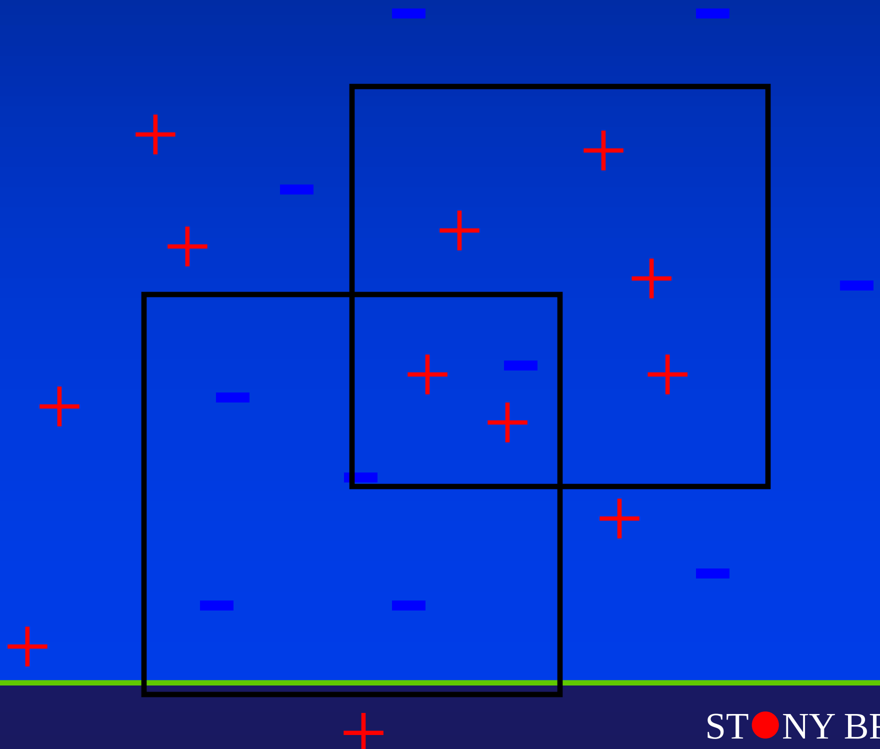
Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations
  - Union
  - Intersection
  - Subtraction



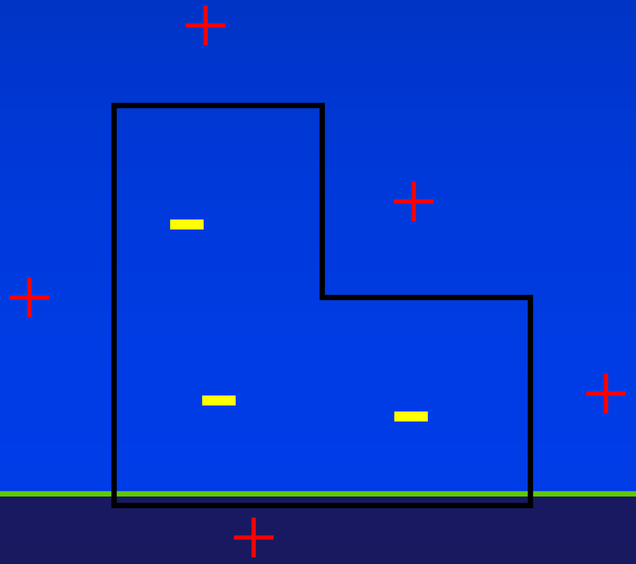
Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations
  - Union
  - Intersection
  - Subtraction



Advantages

- No topology to maintain
- Always defines a closed surface!
- Inside/Outside test
- CSG operations
  - Union
  - Intersection
  - Subtraction

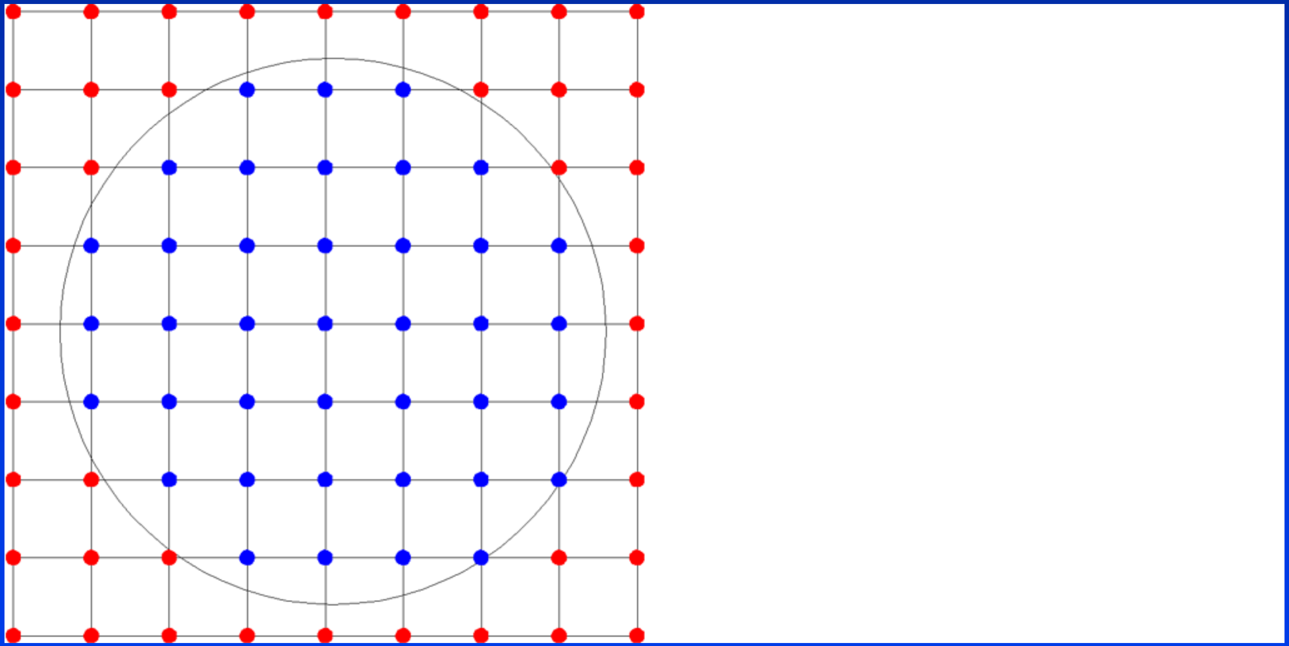


Disadvantages

- Hard to render - no polygons
- Creating polygons amounts to root finding
- Arbitrary shapes hard to represent as a function

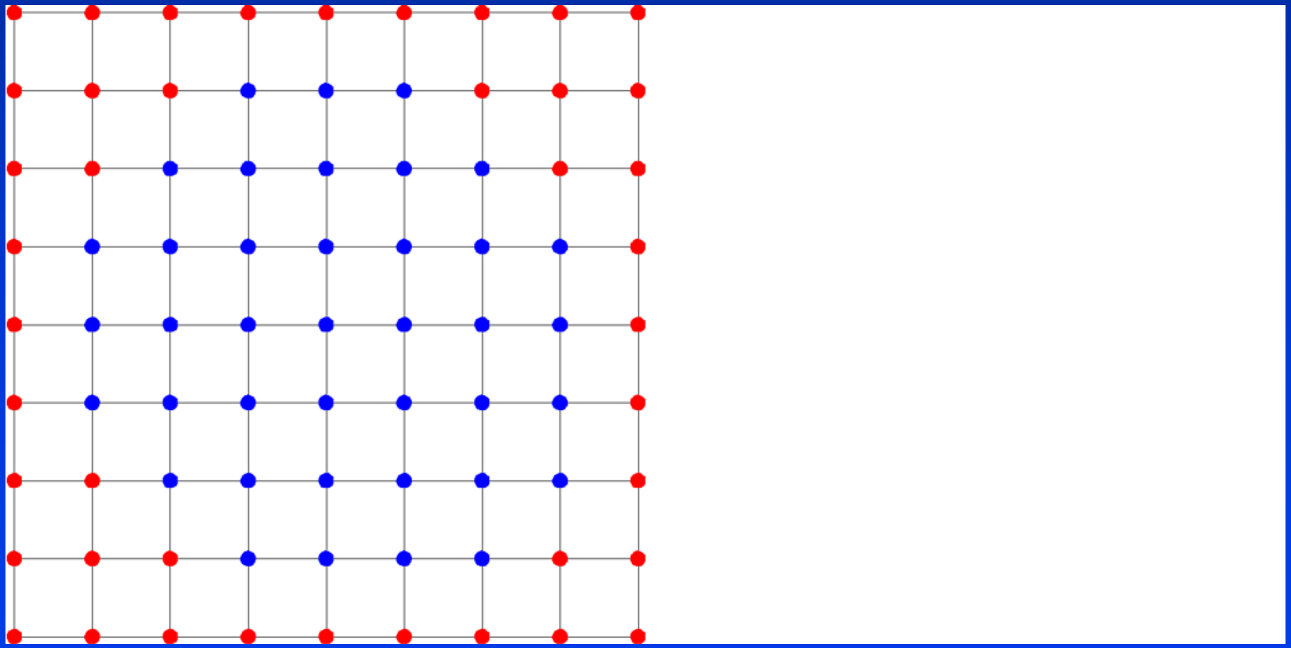
Non-Analytic Implicit Functions

- Sample functions over grids



Non-Analytic Implicit Functions

- Sample functions over grids



Sketch-Based 3D Modeling System ?

Key Concept:

Anyone can create 3D models

Method:

3D modeling from sketched 2D strokes

Technical Challenges

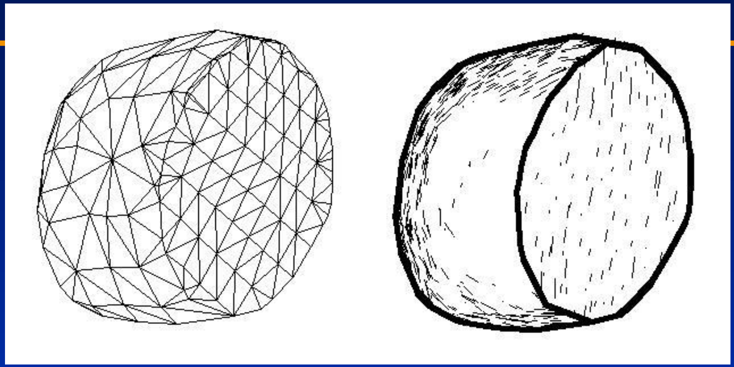
- A sketch-based modeling system
  - Easy
  - Interactive

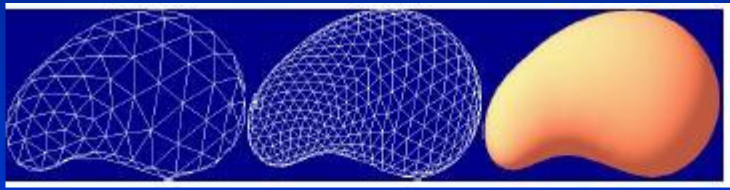
Problem:

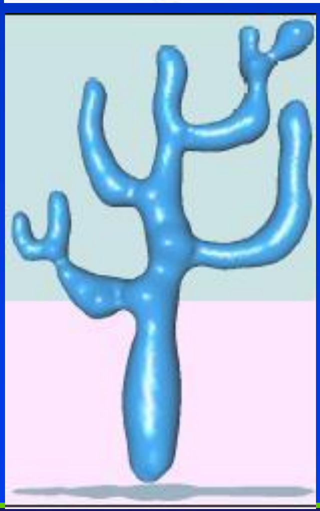
It is difficult to support complex models

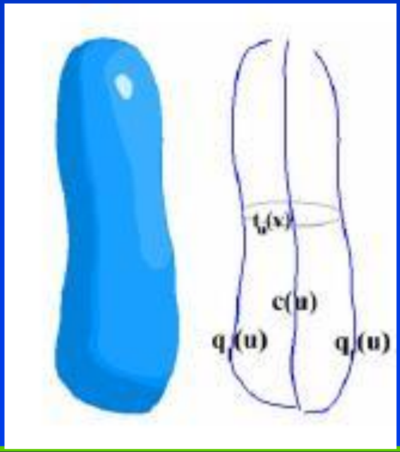
Various Kinds of Sketch-Based Modeling Systems

- Triangle meshes
- Subdivision surfaces
- Implicit surfaces
- Parametric surfaces









Teddy

- Triangle meshes
- Chordal axis

△Low complex models

Implicit Approaches

- Blending operation

△ A Large matrix must be solved

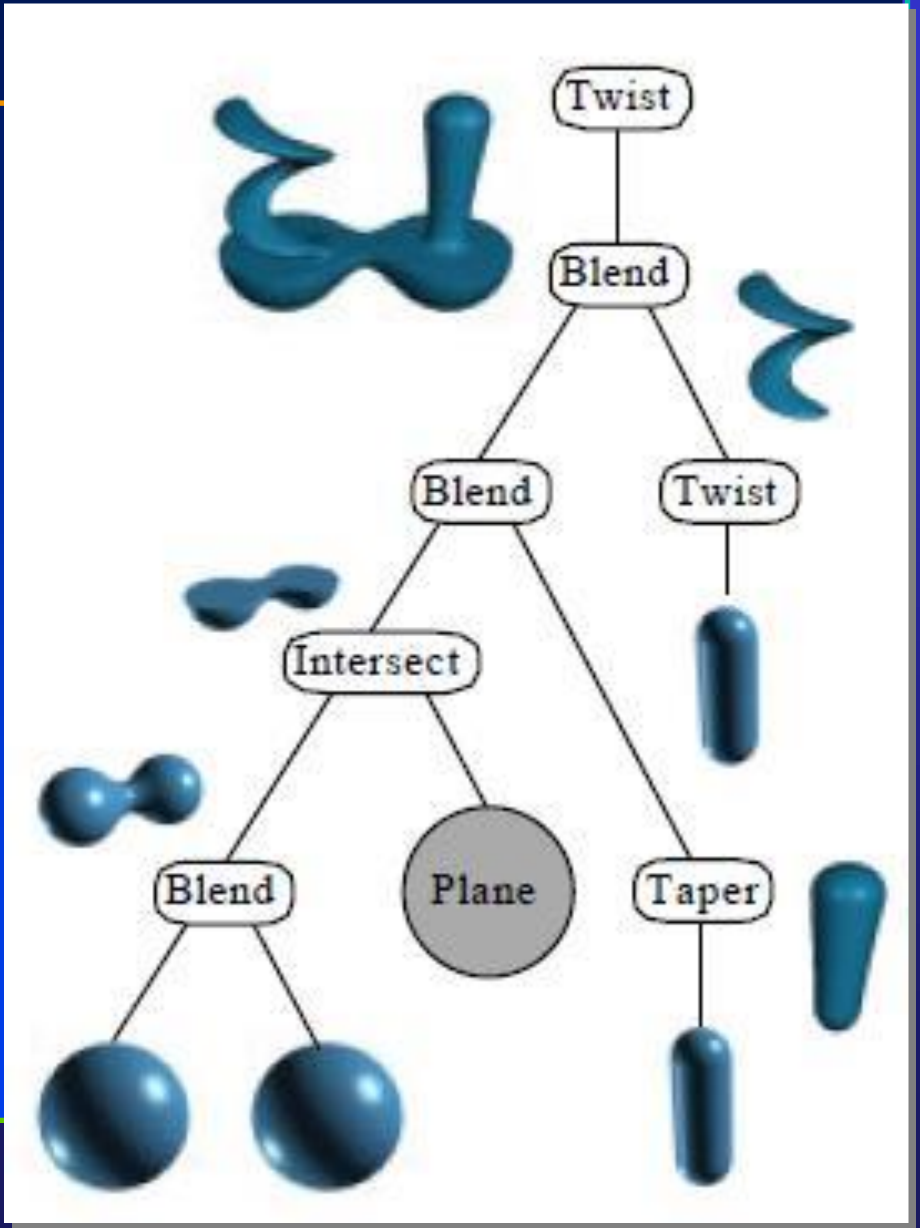
Approach:

BlobTree

(Hierarchical implicit volume Models)

BlobTree

- Leaves:
  Implicit primitives
- Tree nodes: Composition operators
- Complex 3D modeling with skeletal primitives



Why is BlobTree Effective?

- Non-linear editing of primitives

  ⇒ Complex models can be constructed **easily**

- A hierarchical spatial cashing

  ⇒ Complex models can be constructed **Interactively**

Basic Functionalities

- Creating an implicit field from 2D contours defined by sketched strokes
- Converting 2D contours into 3D implicit volumes
- Editing 3D implicit volumes in BlobTree

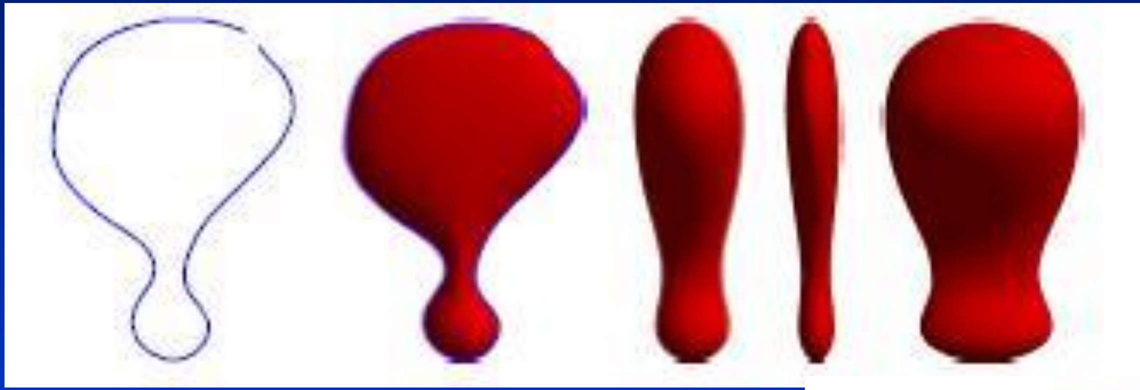
A Sketch-Based Implicit Field

$$g_{wyvill}(x) = (1 - x^2)^3$$

- C^2 Continuity
- $f_M = v_{iso}$ on a 2D stroke

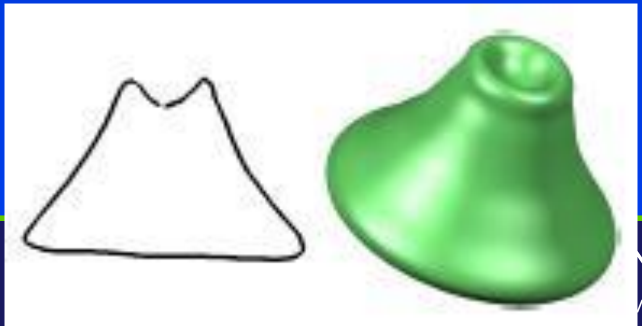
Three Types of Surfaces

- Blobby inflation



- Linear sweeps

- Surfaces of revolution

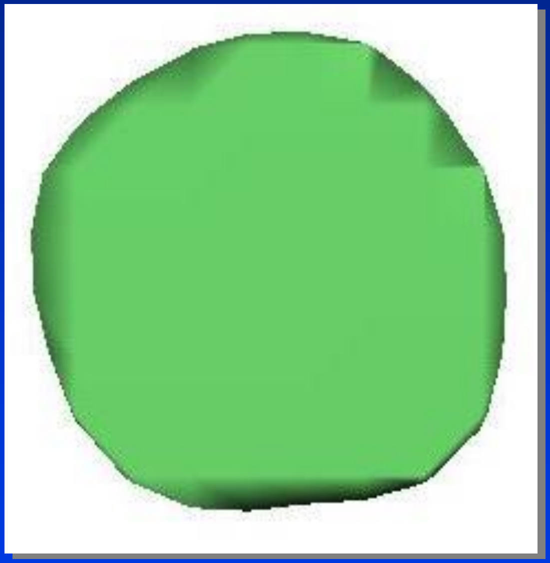


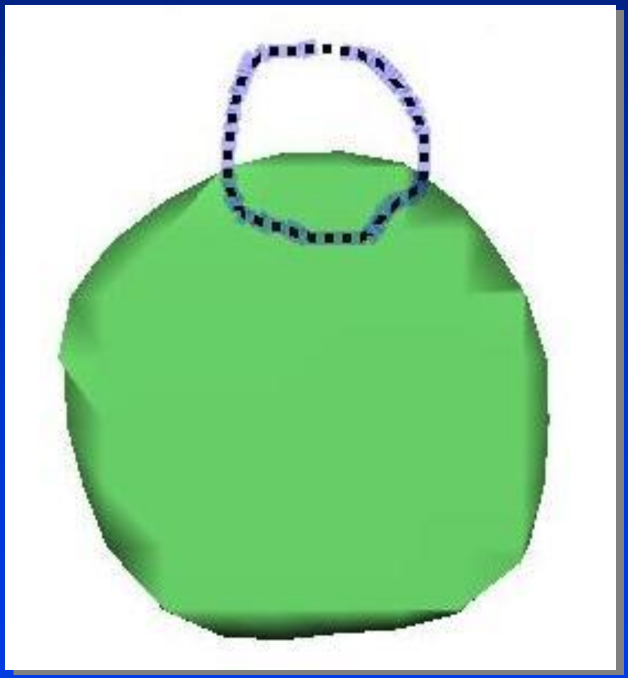
Operations

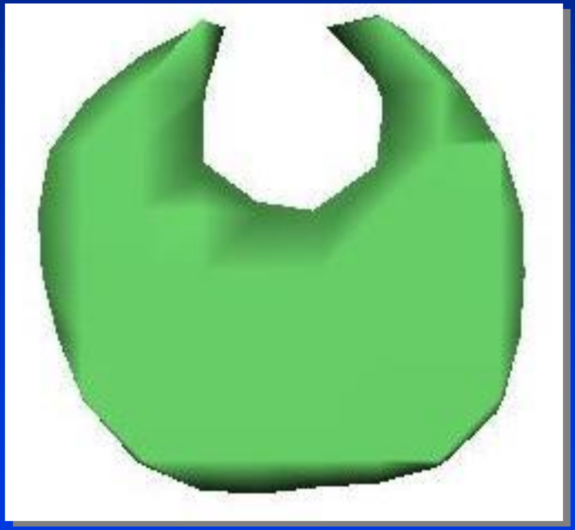
- Cutting (CSG)

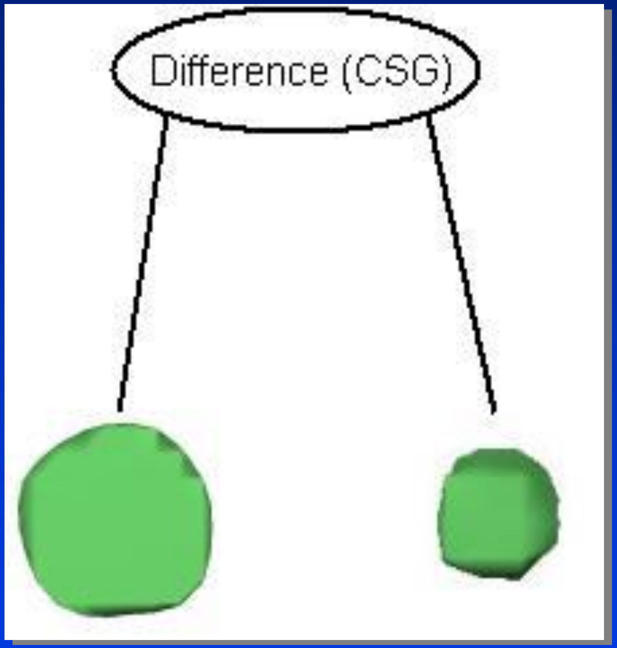
- Blending

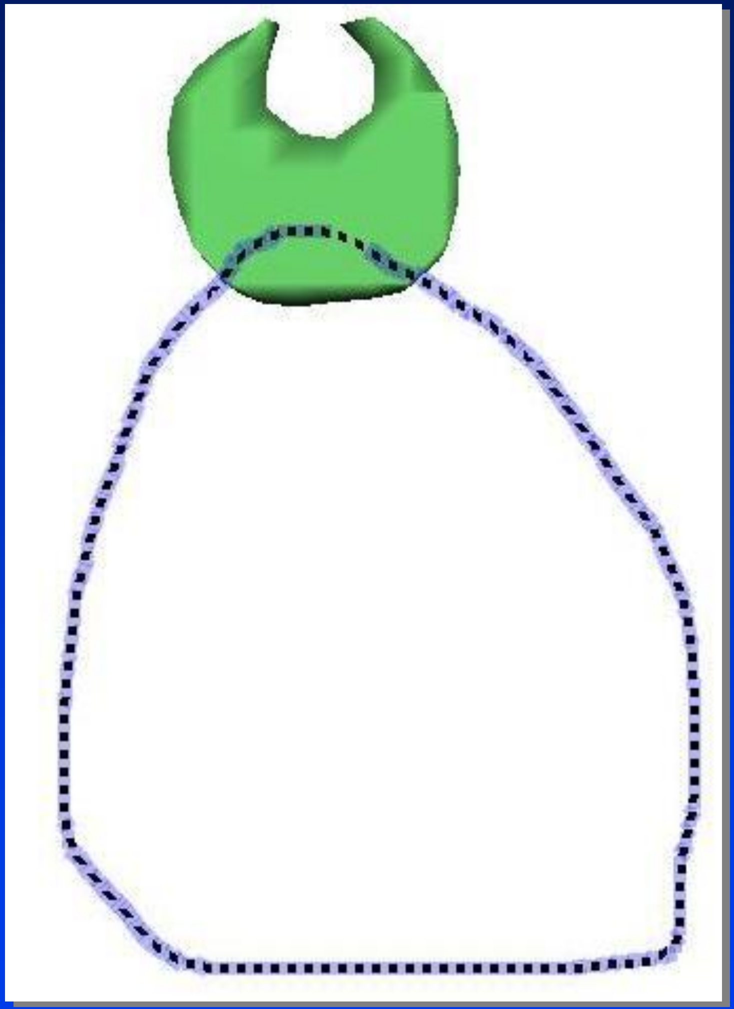
An Example

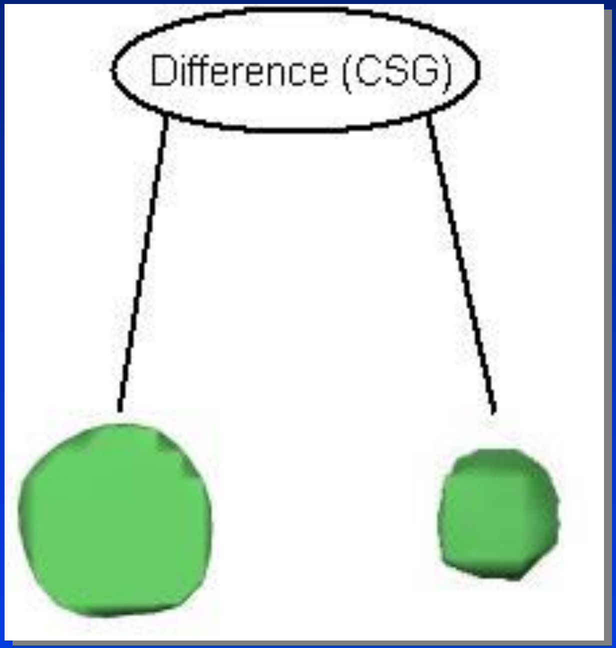


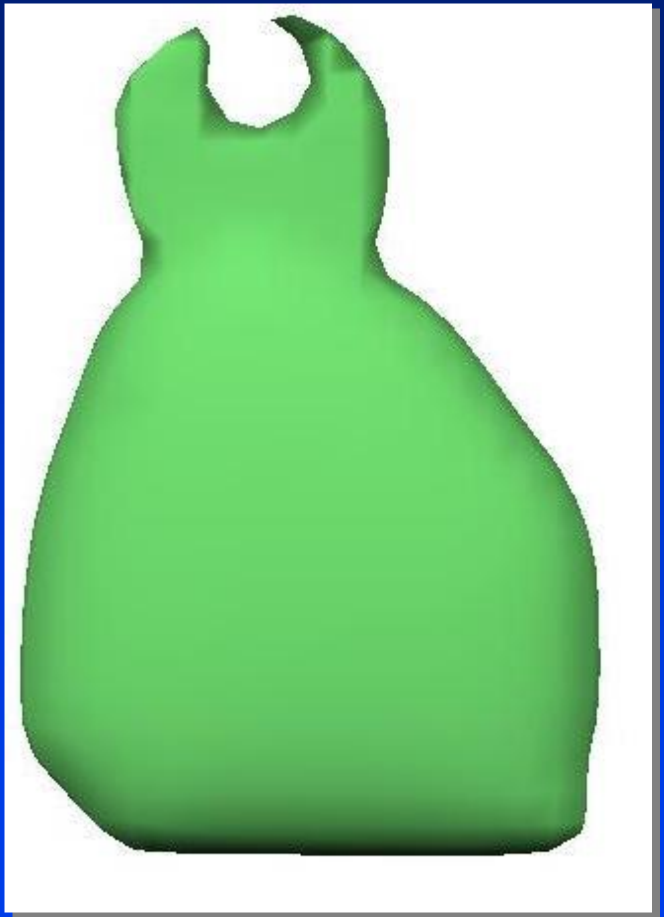


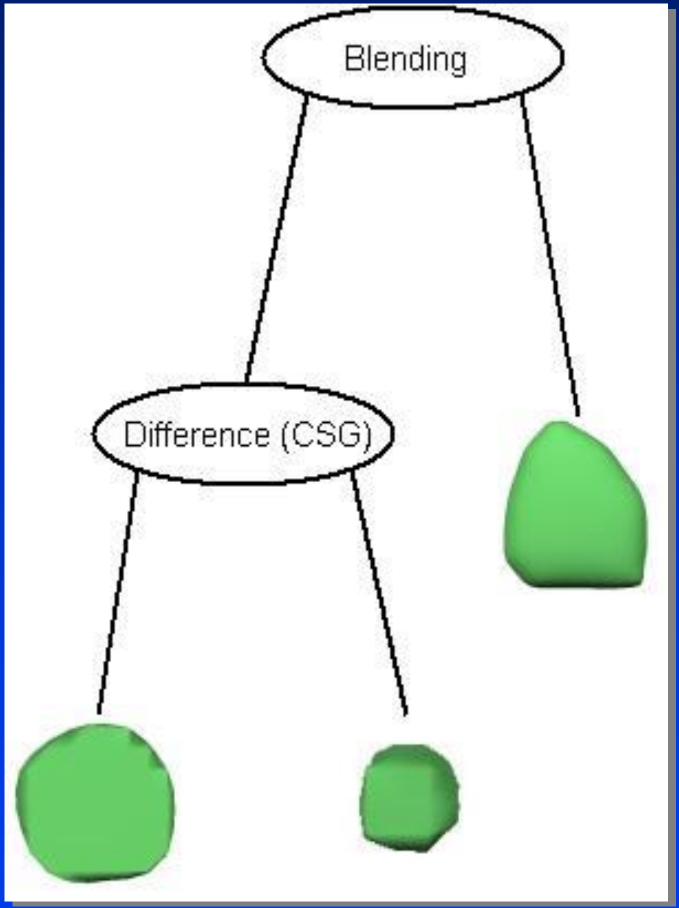


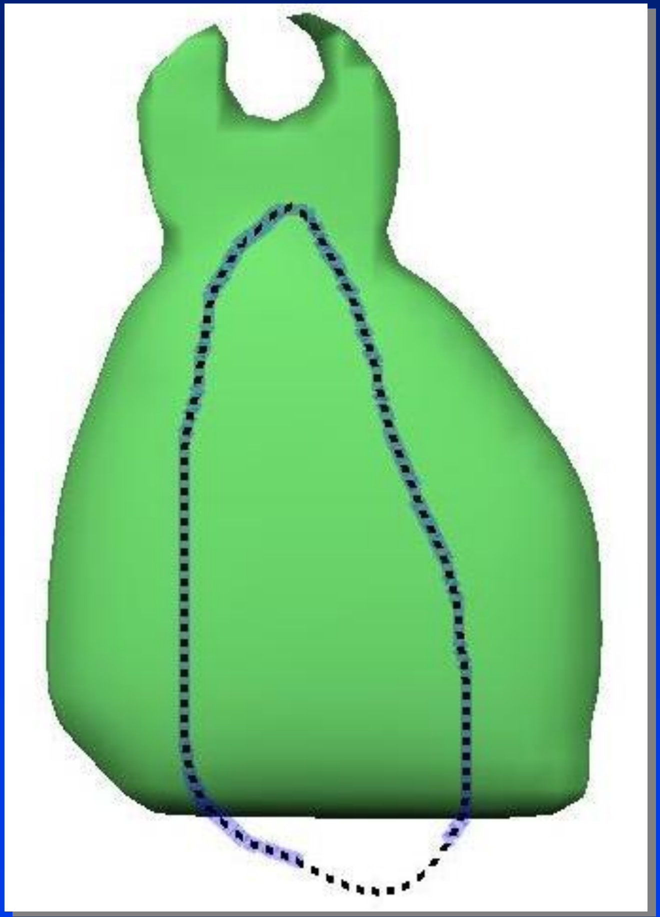

Difference (CSG)

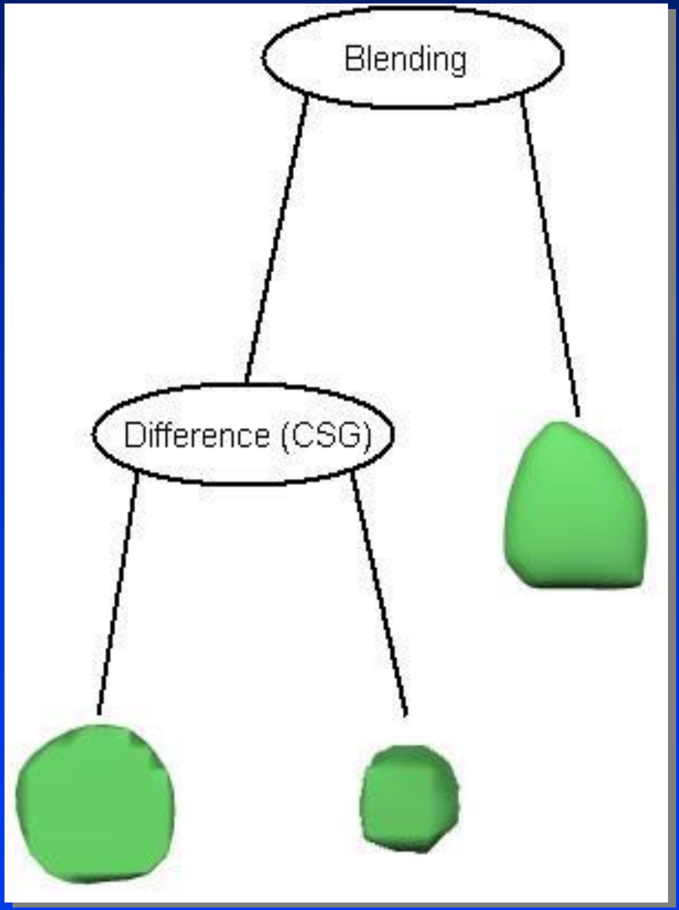


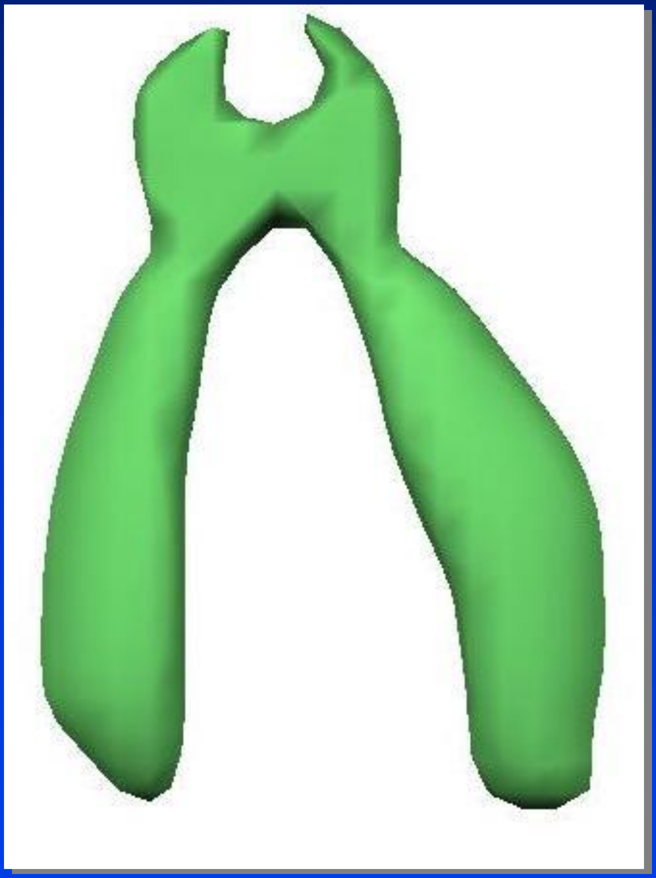

Difference (CSG)

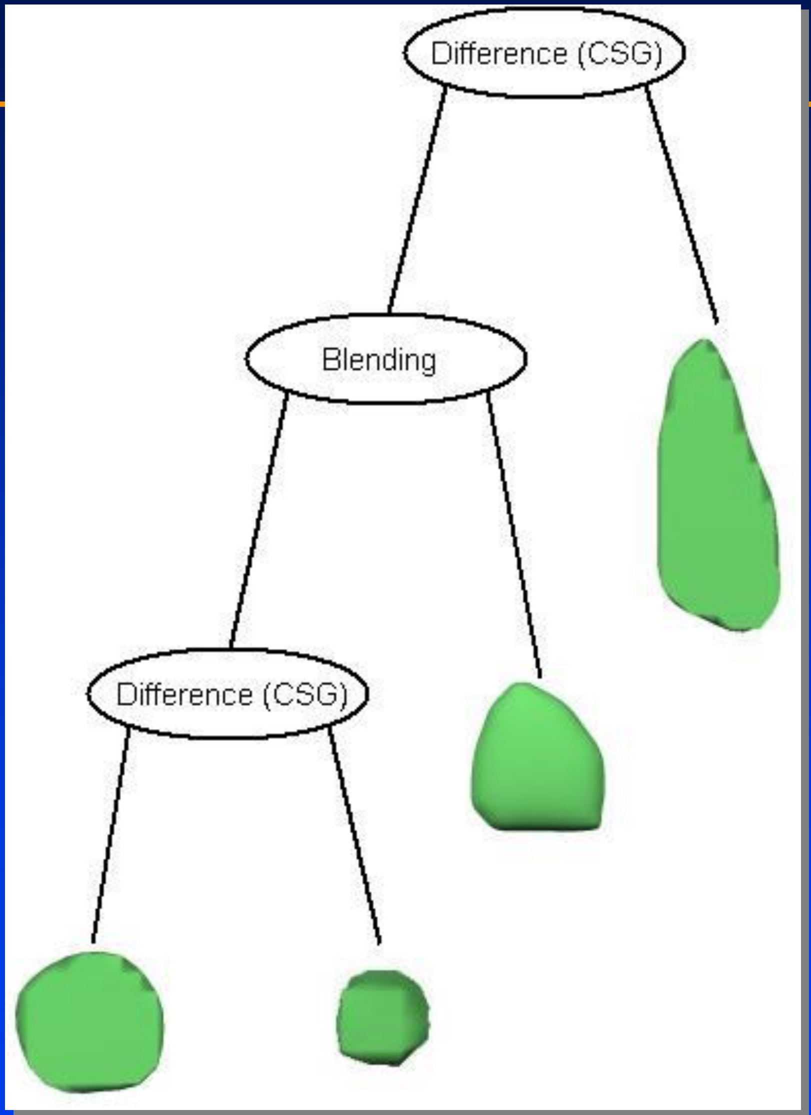


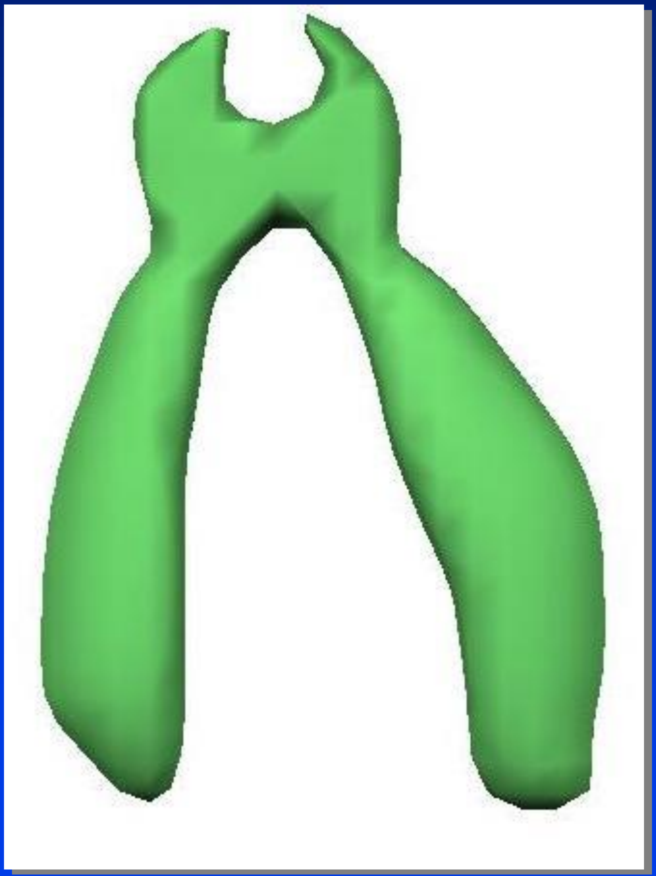

Blending
Difference (CSG)

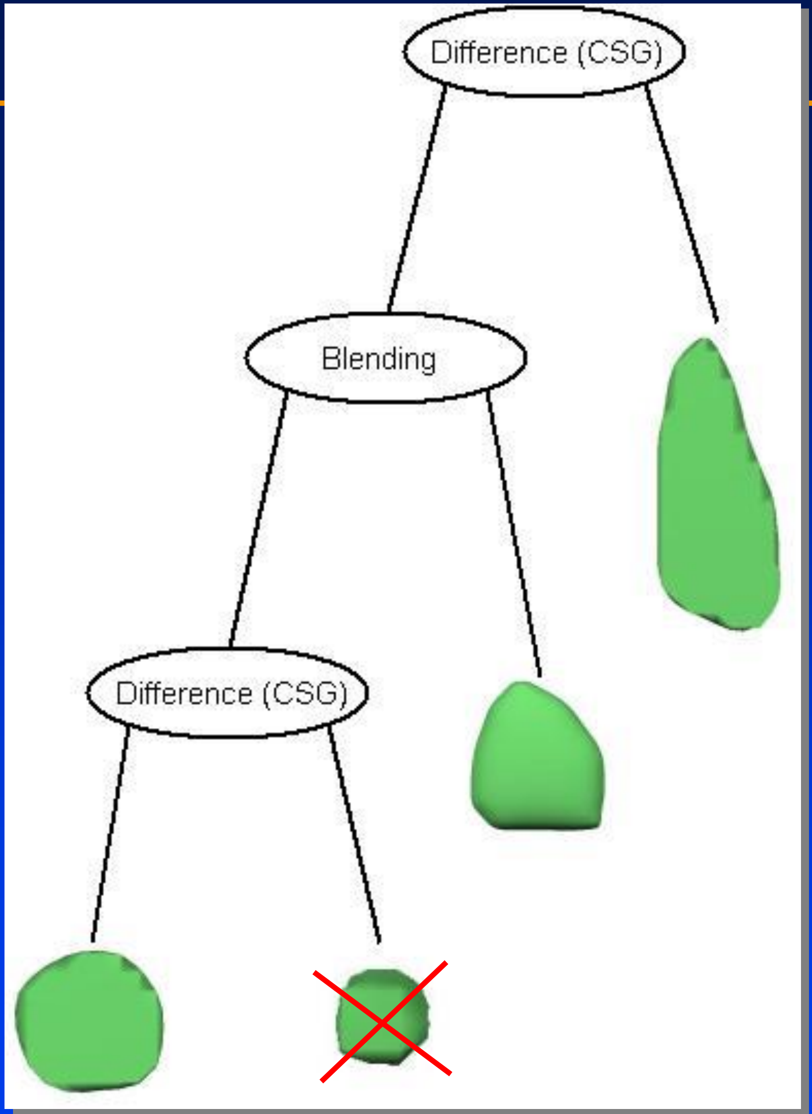


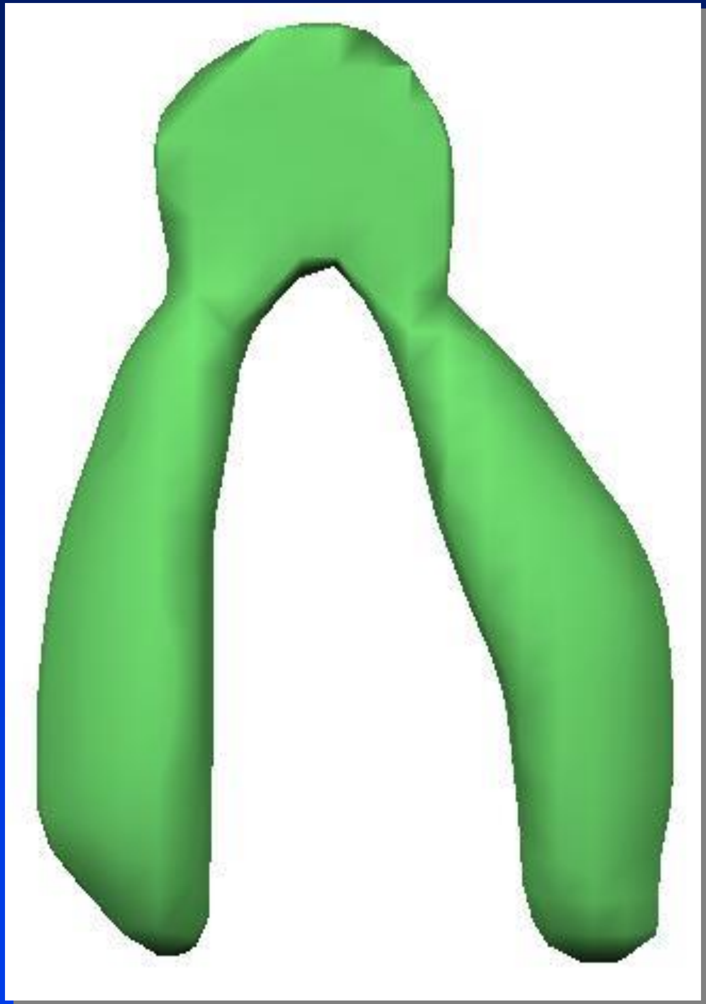

Blending
Difference (CSG)

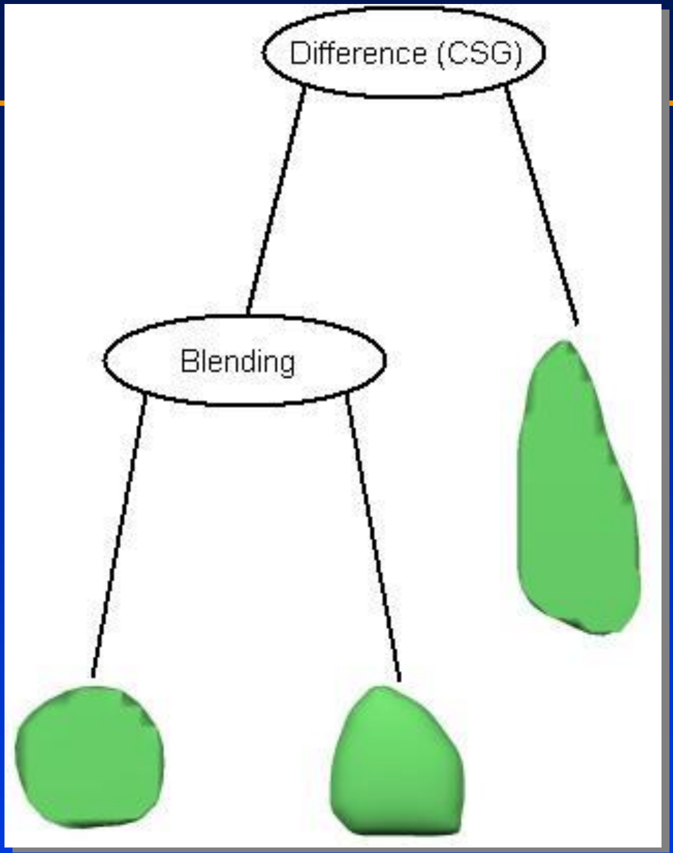


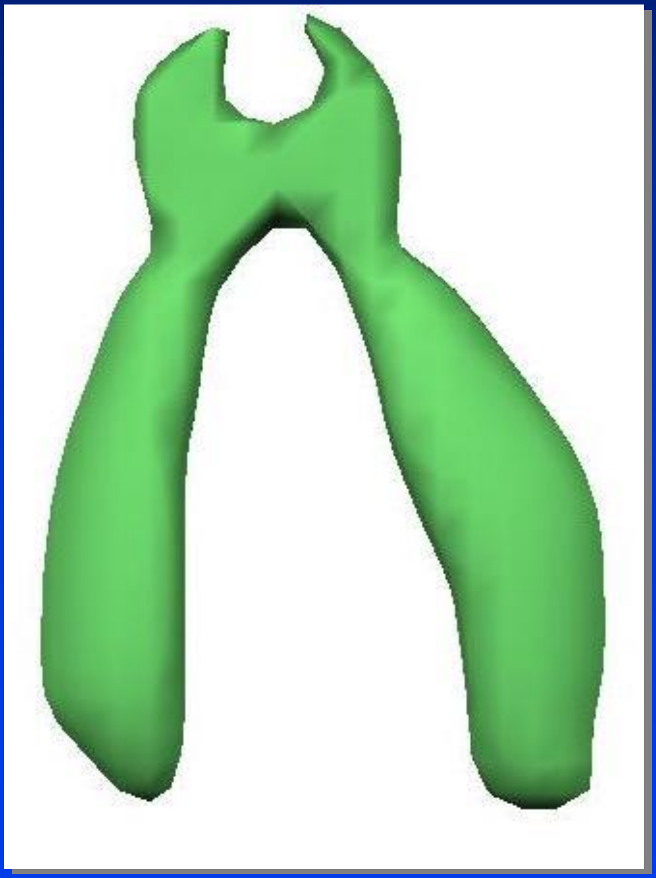

Difference (CSG)
Blending
Difference (CSG)

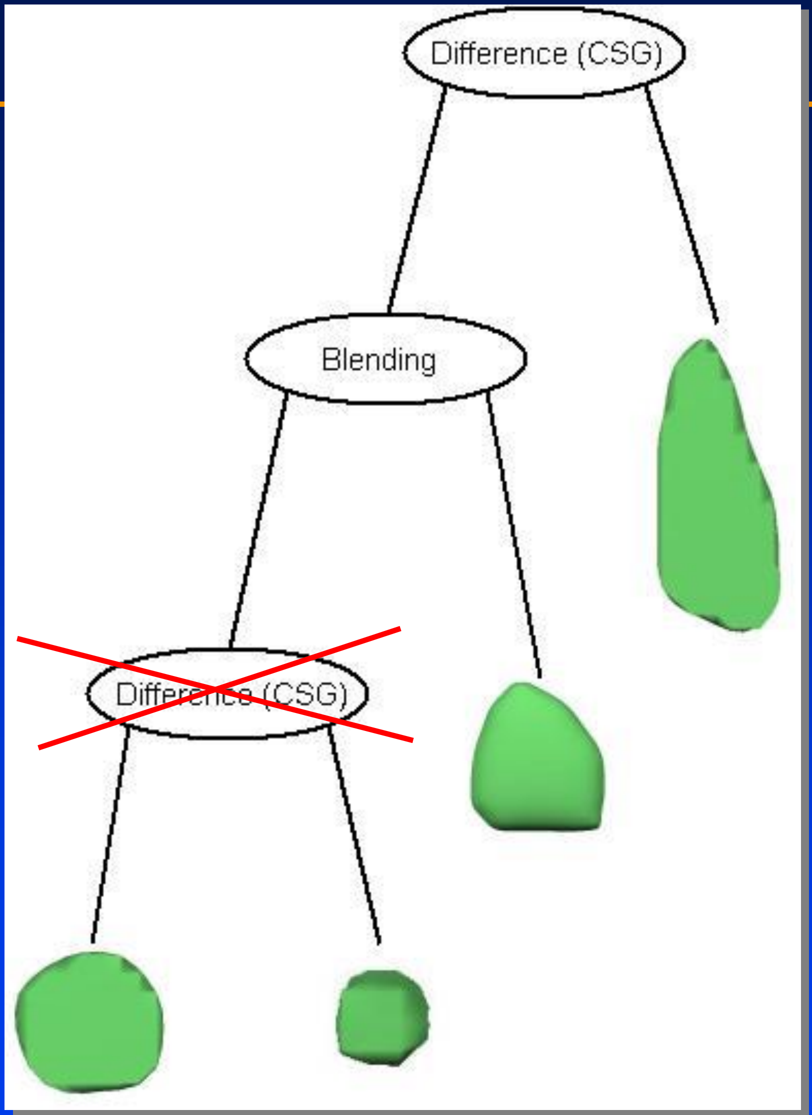


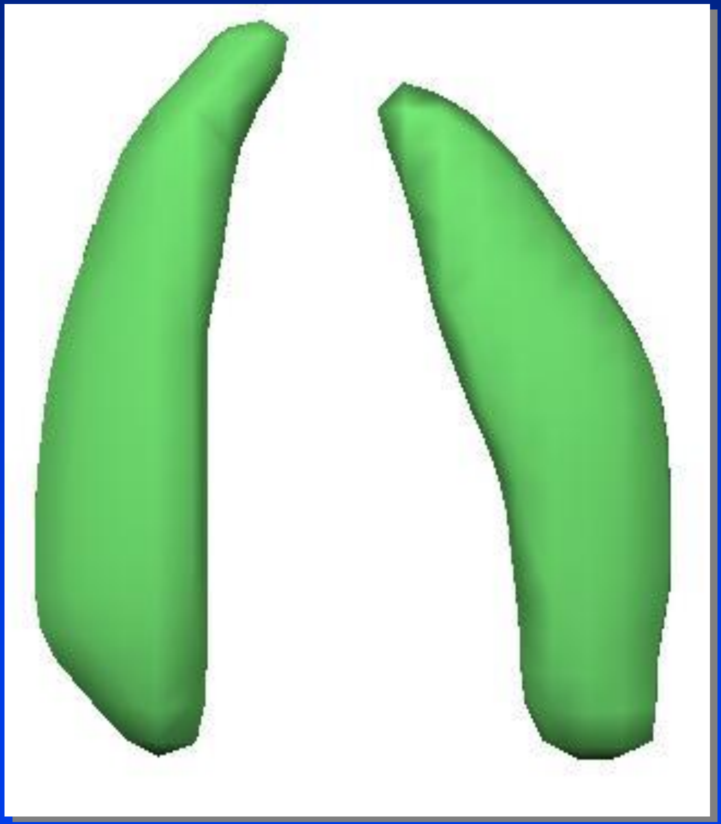

Difference (CSG)
Blending
Difference (CSG)

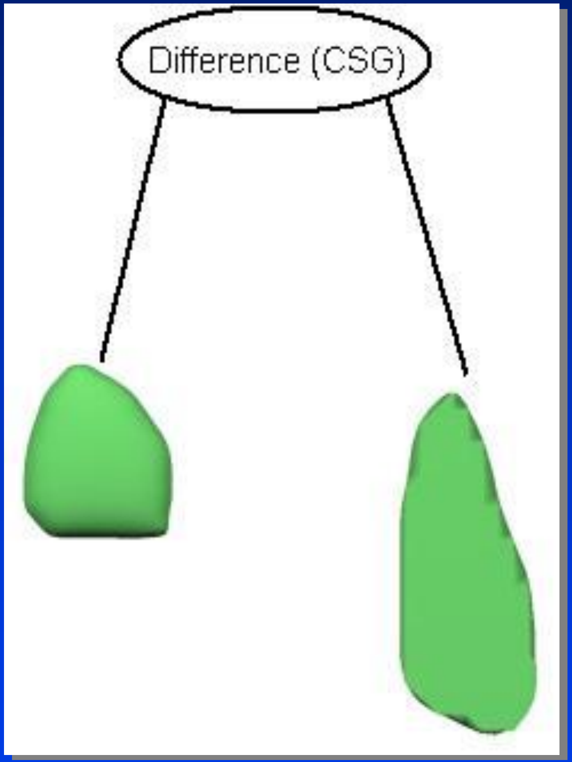



Difference (CSG)
Blending

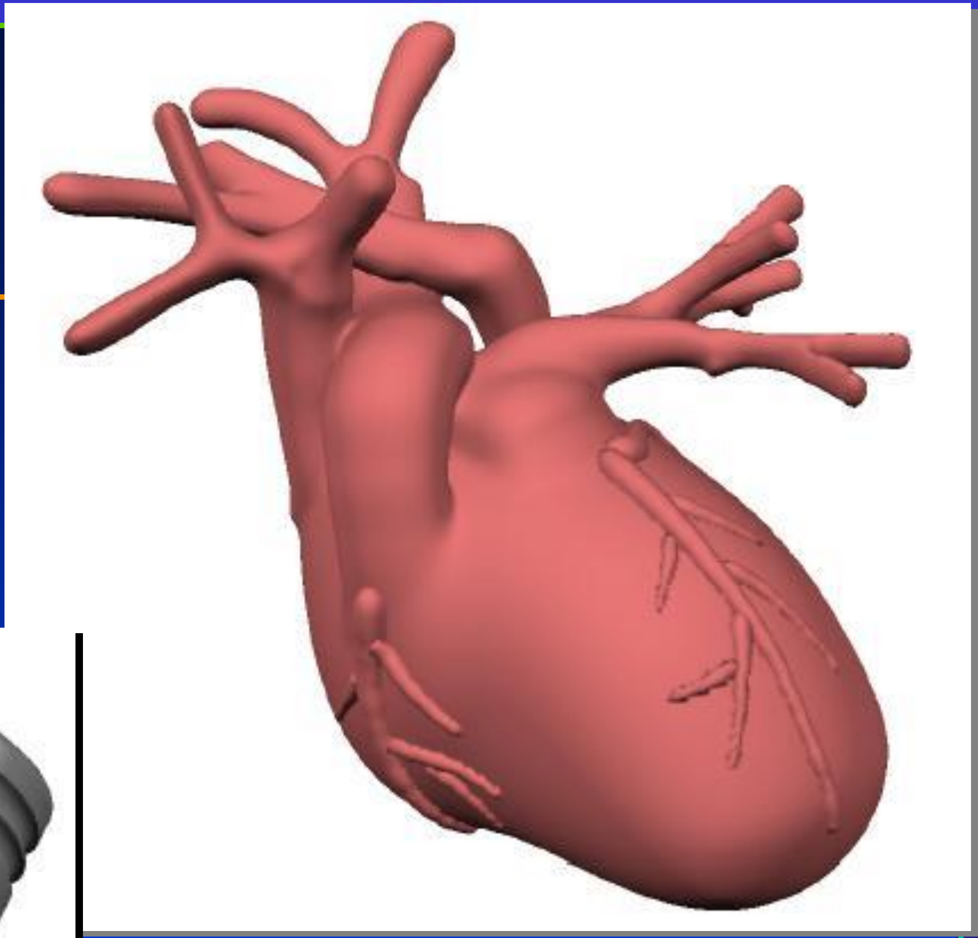


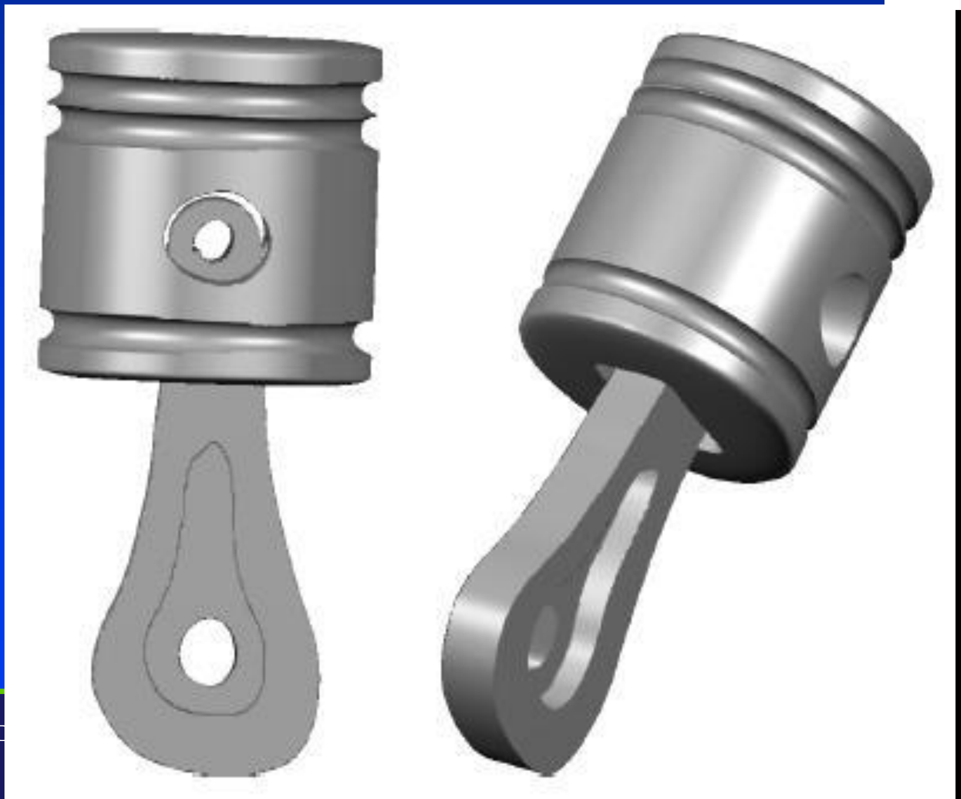

Difference (CSG)
Blending
Difference (CSG)



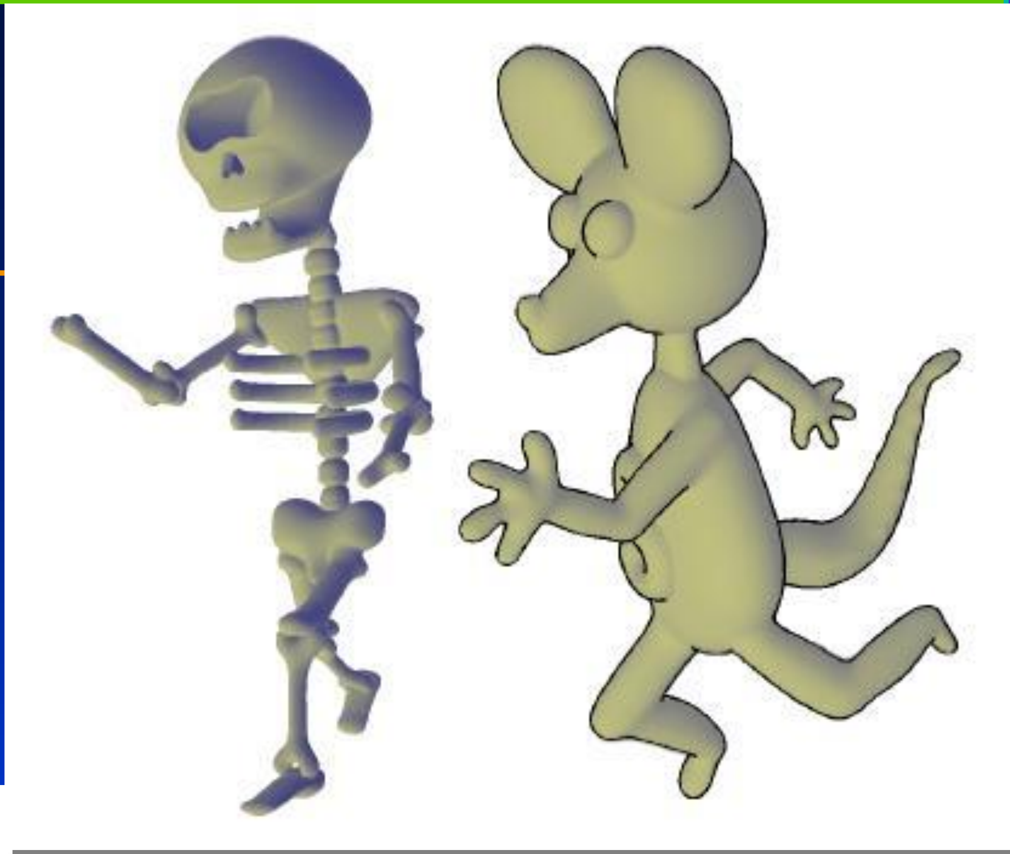

Difference (CSG)

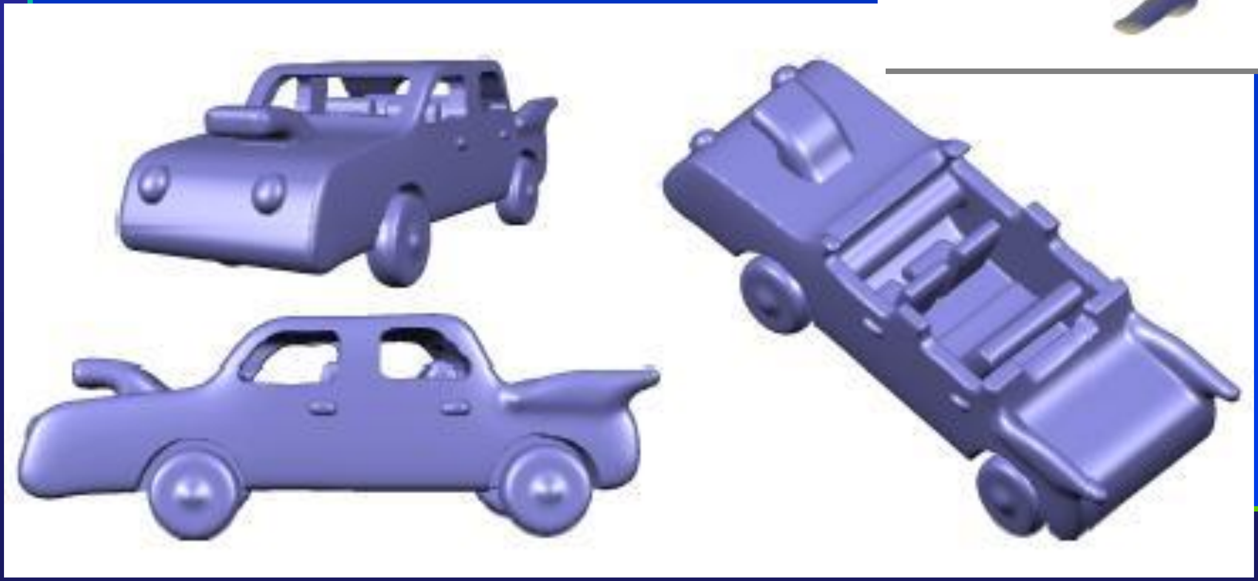
Results





Results





BlobTree

- BlobTree has allowed us to create complex 3D models in a sketch-based modeling system

△ The User must understand BlobTree structure