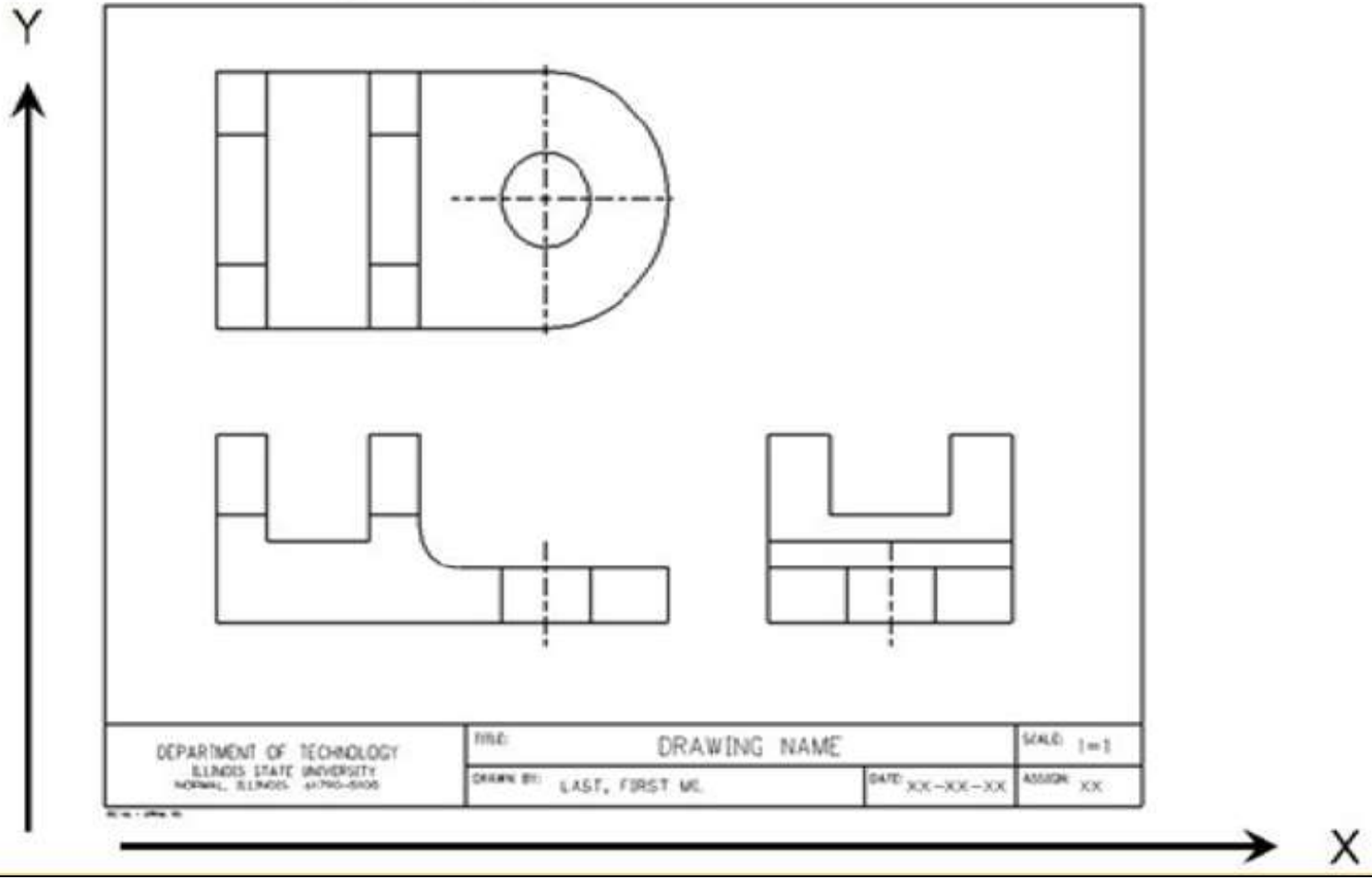
UNIT - 4

SOLID MODELING
(PART -1)

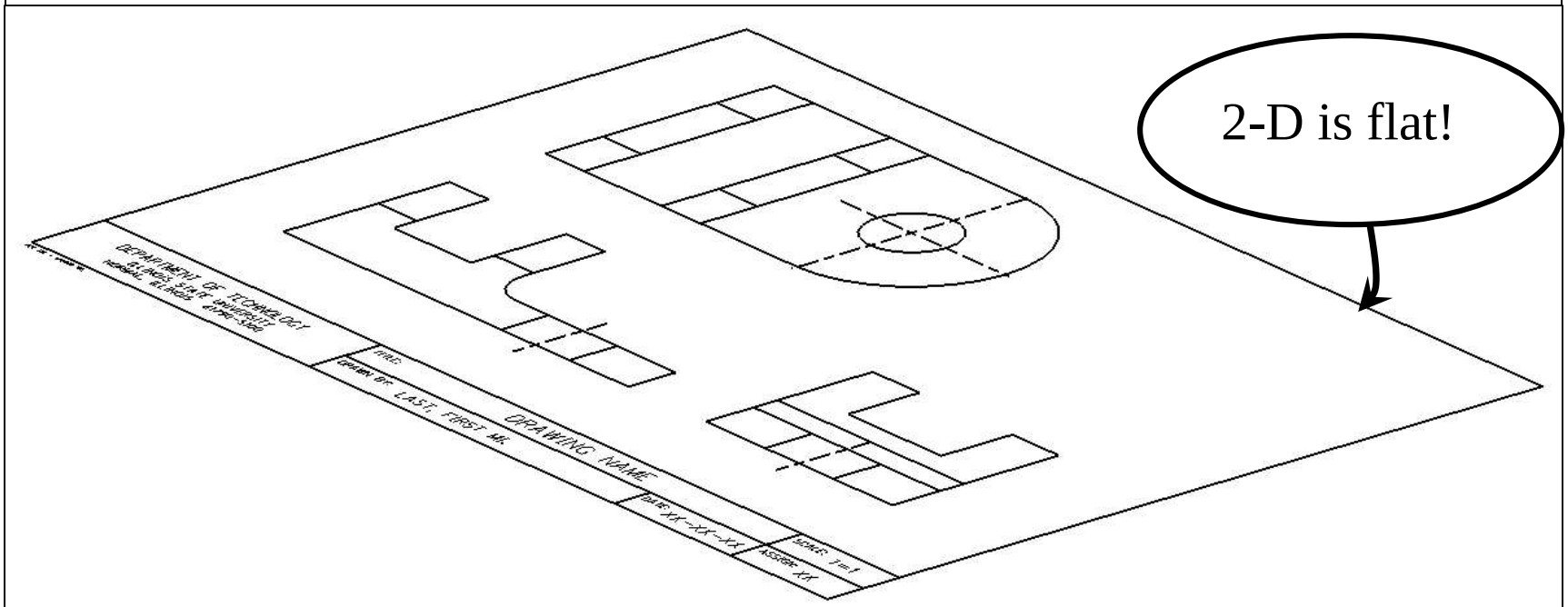
Two-Dimensional Drawing

- 3-dimensional objects have traditionally been drawn using only 2-dimensions



Two-Dimensional Drawing

– 2-D is not ideal for representing 3-D objects
  - 2-D has no **Z** axis

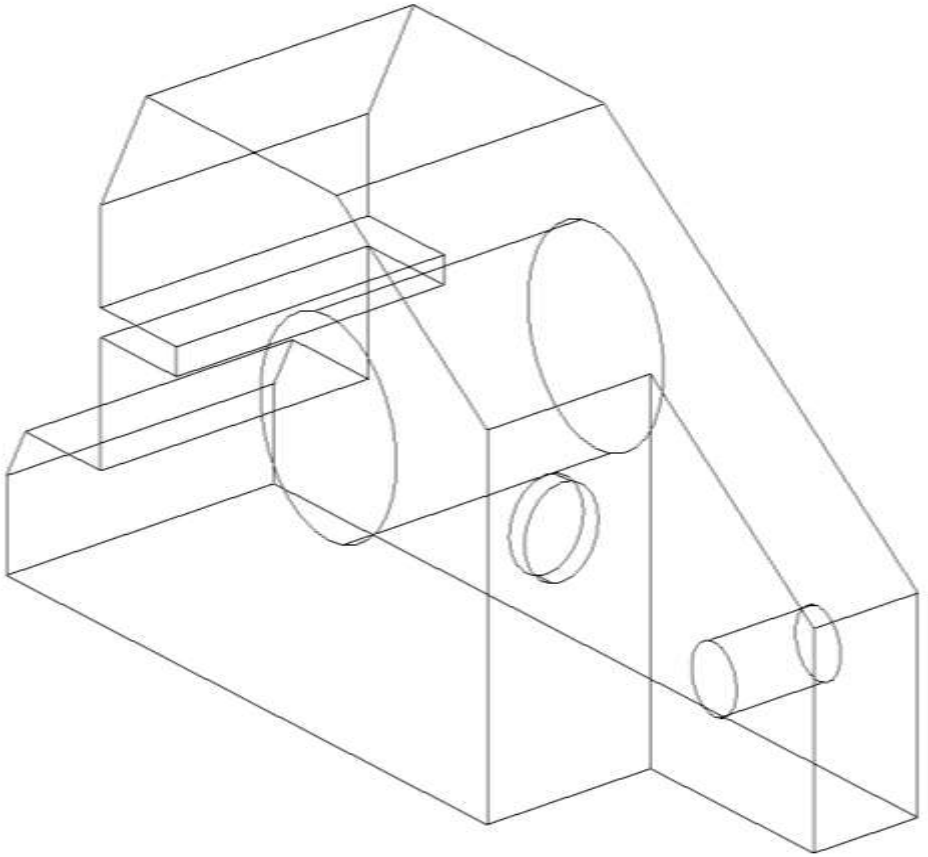


Three-Dimensional Modeling

- 3-D models include **X**, **Y**, and **Z** dimensions
- Allows better definition of three-dimensional objects
- There are three general types of 3-D models
  - Wire Frame Models
  - Surface Models
  - Solid Models

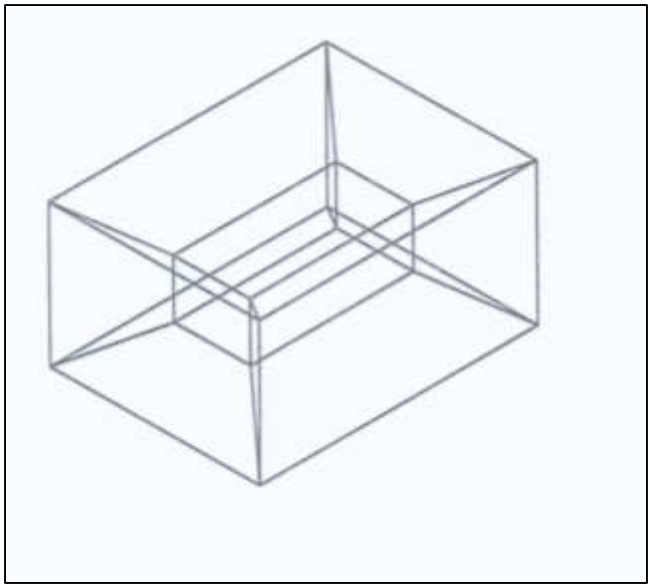
Wire Frame Models

- **Oldest form of 3D modeling**
- **Old technology - not used today**
- **Model Contains edges and vertices**
- **Cannot represent complex surfaces**
- **No details regarding interior of part**
- **Ambiguous**



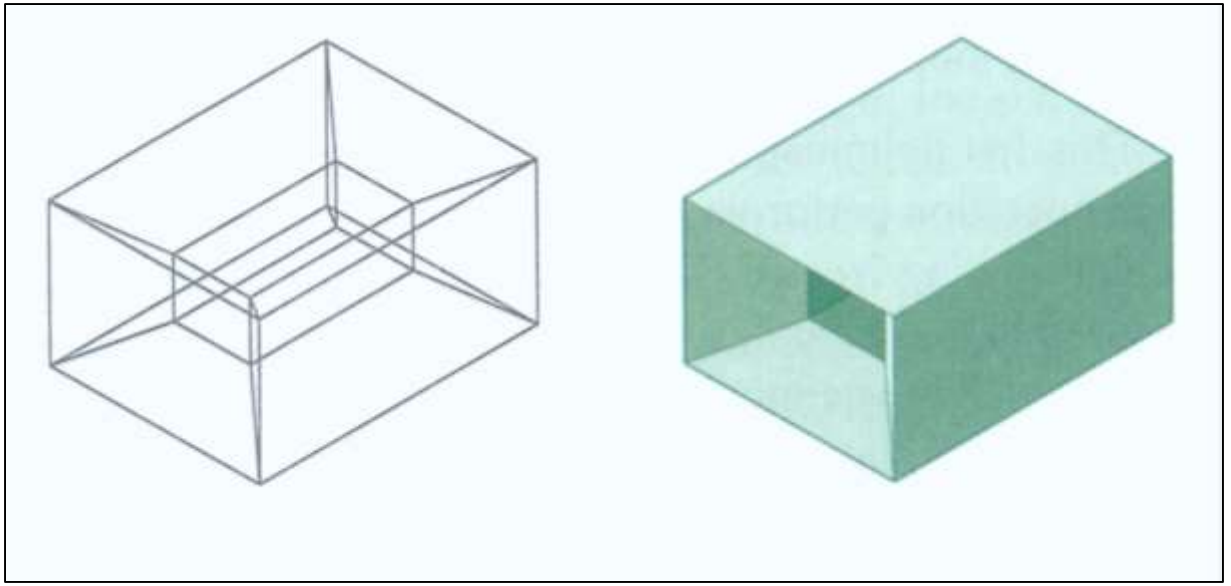
Wireframe models are Ambiguous…

What does this object really look like?



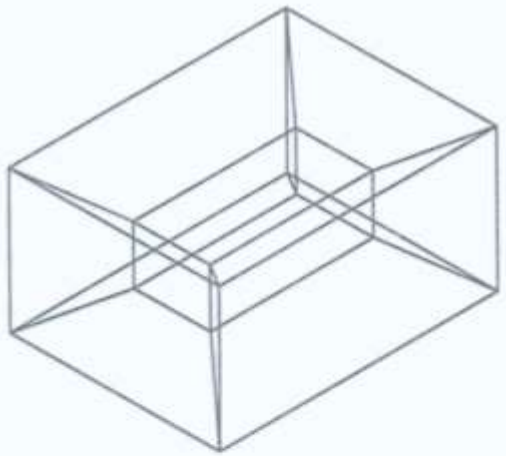
Wireframe models are Ambiguous…

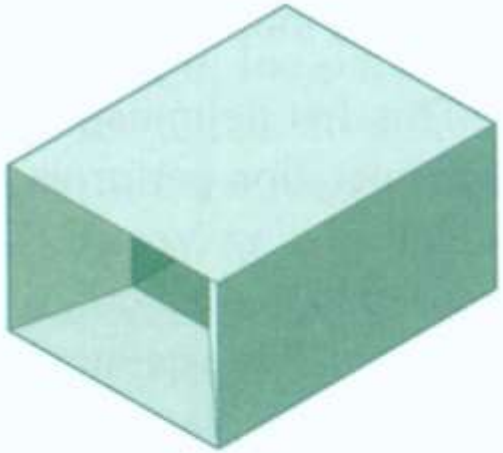
What does this object really look like?

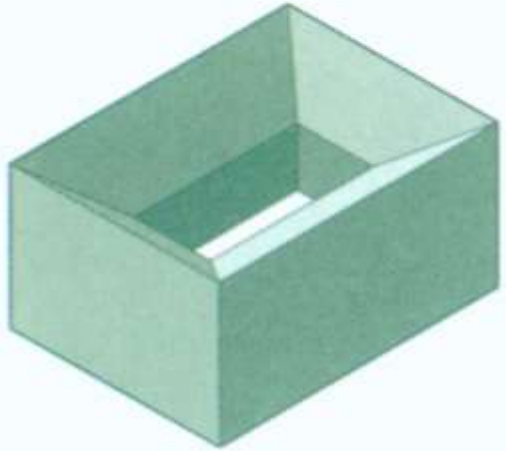


Wireframe models are Ambiguous…

What does this object really look like?

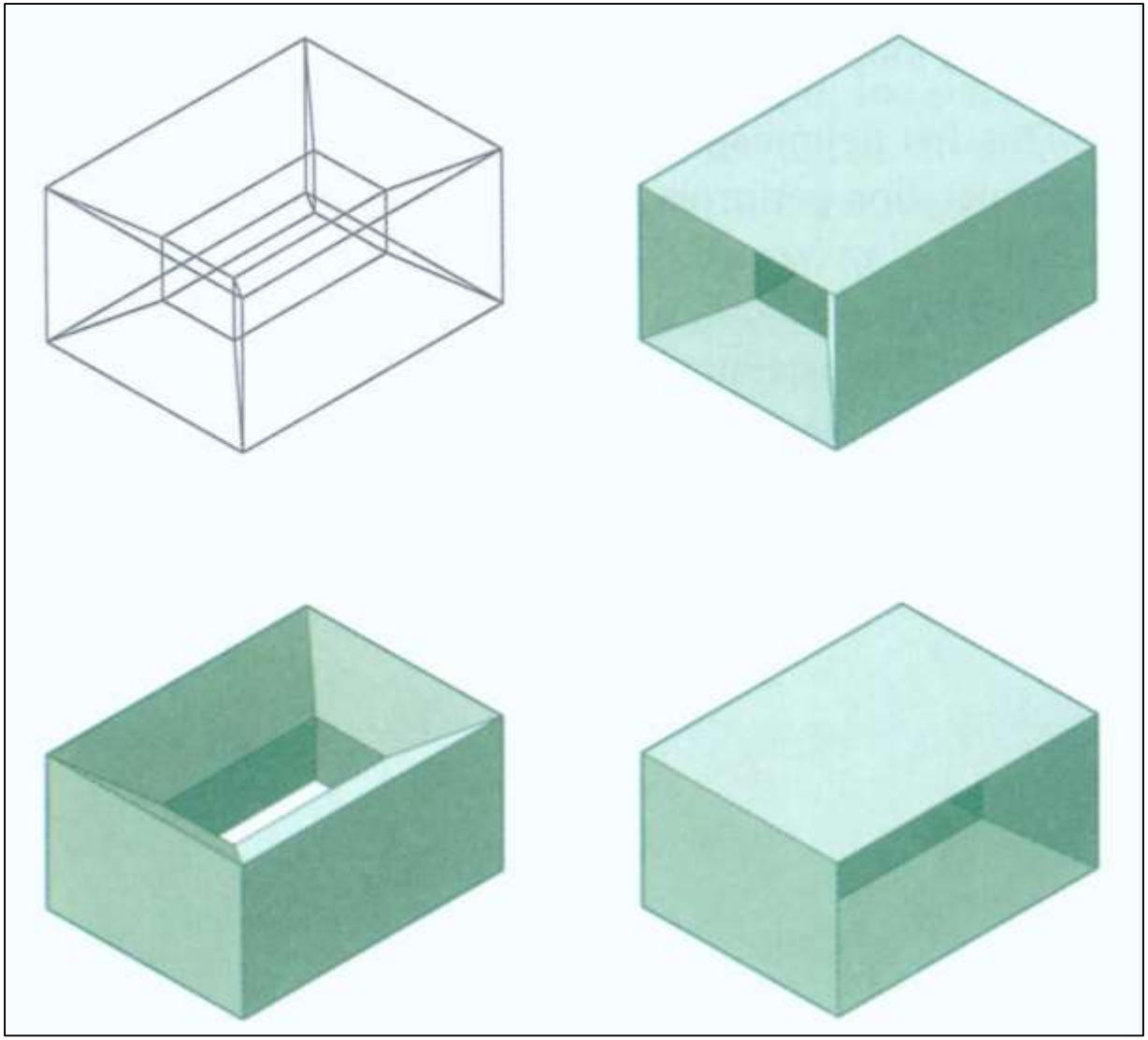






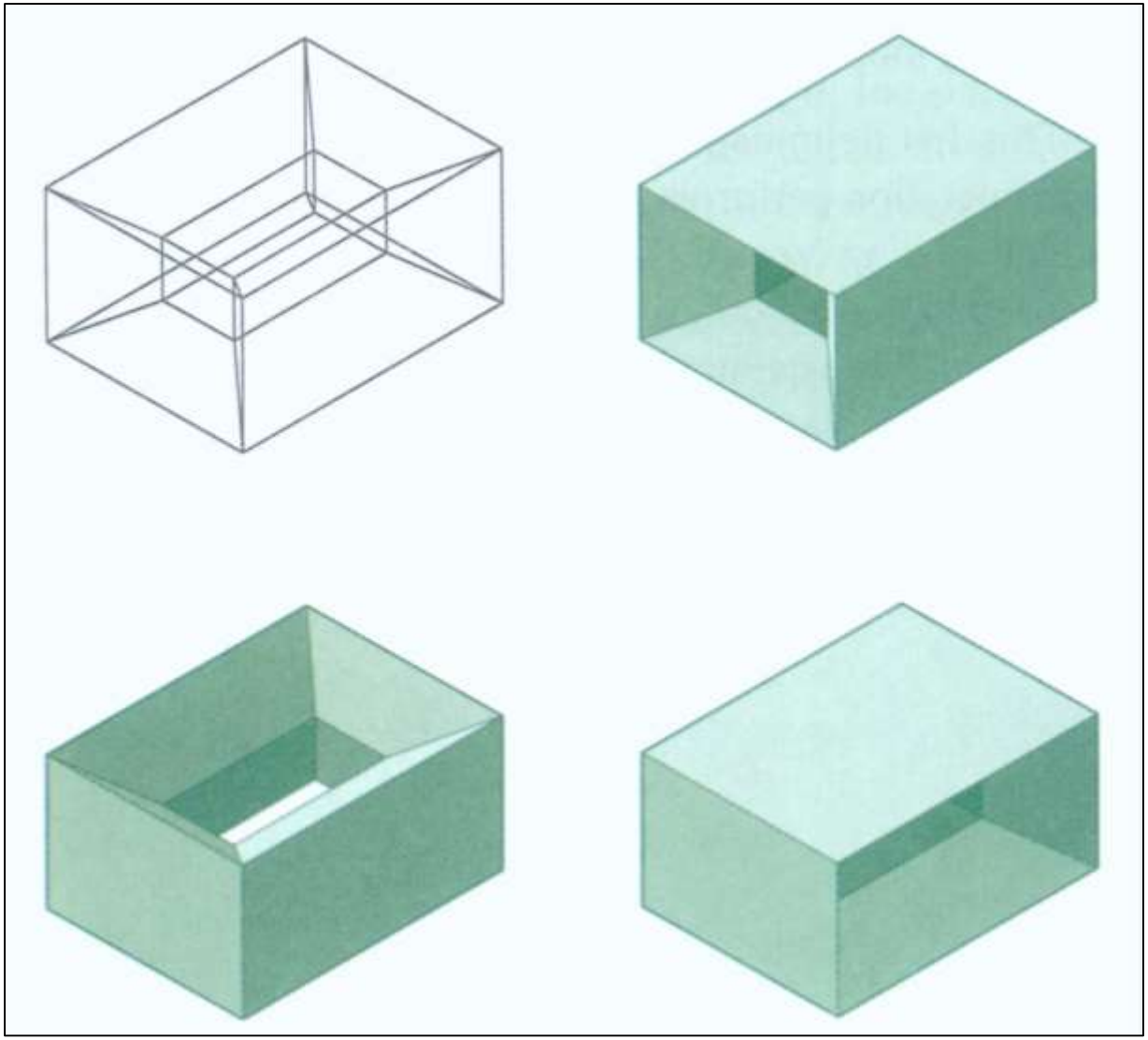
Wireframe models are Ambiguous…

What does this object really look like?



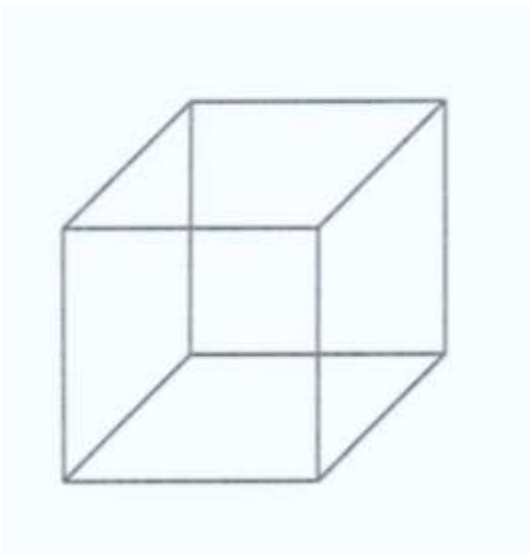
Wireframe models are Ambiguous…

What does this object really look like?



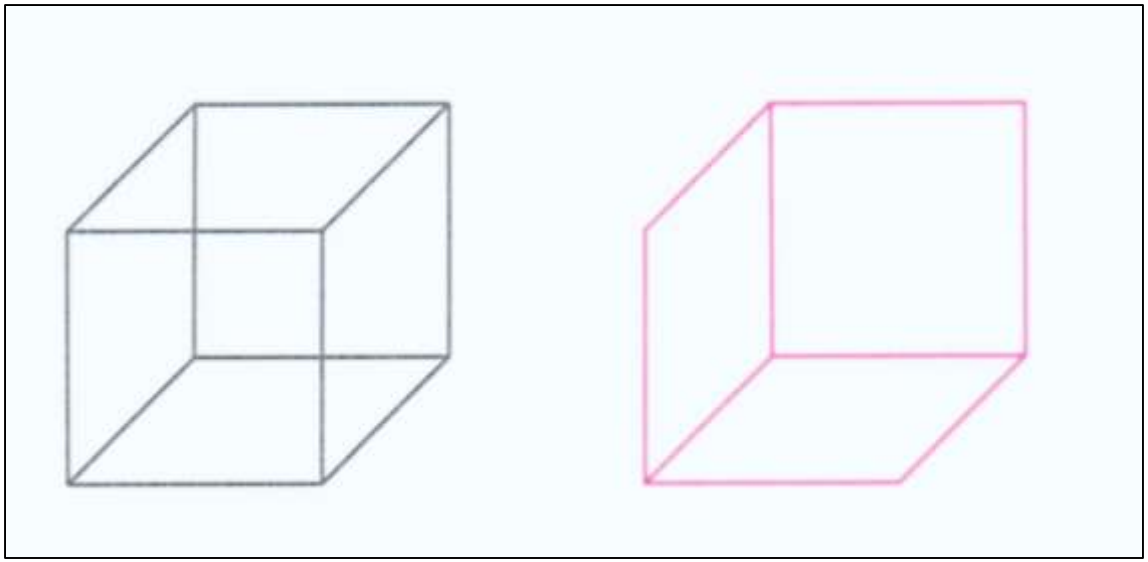
Wireframe Ambiguity (Cont.)

*Which face is **front** and which is **back**?*



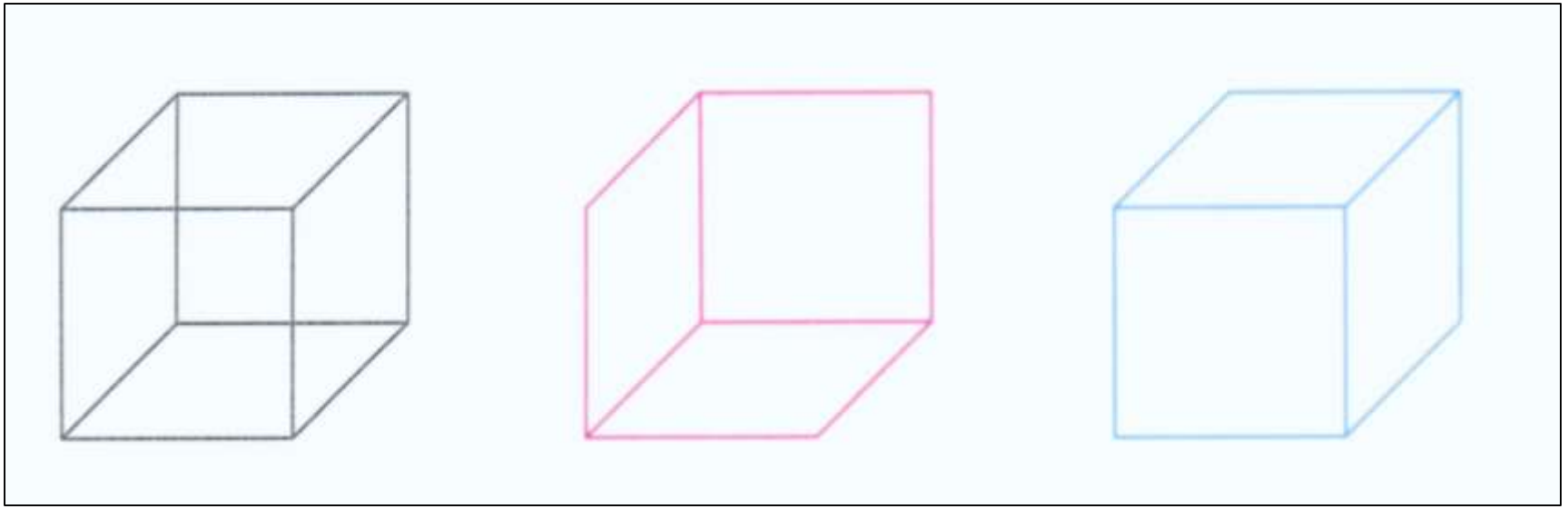
Wireframe Ambiguity (Cont.)

*Which face is **front** and which is **back**?*



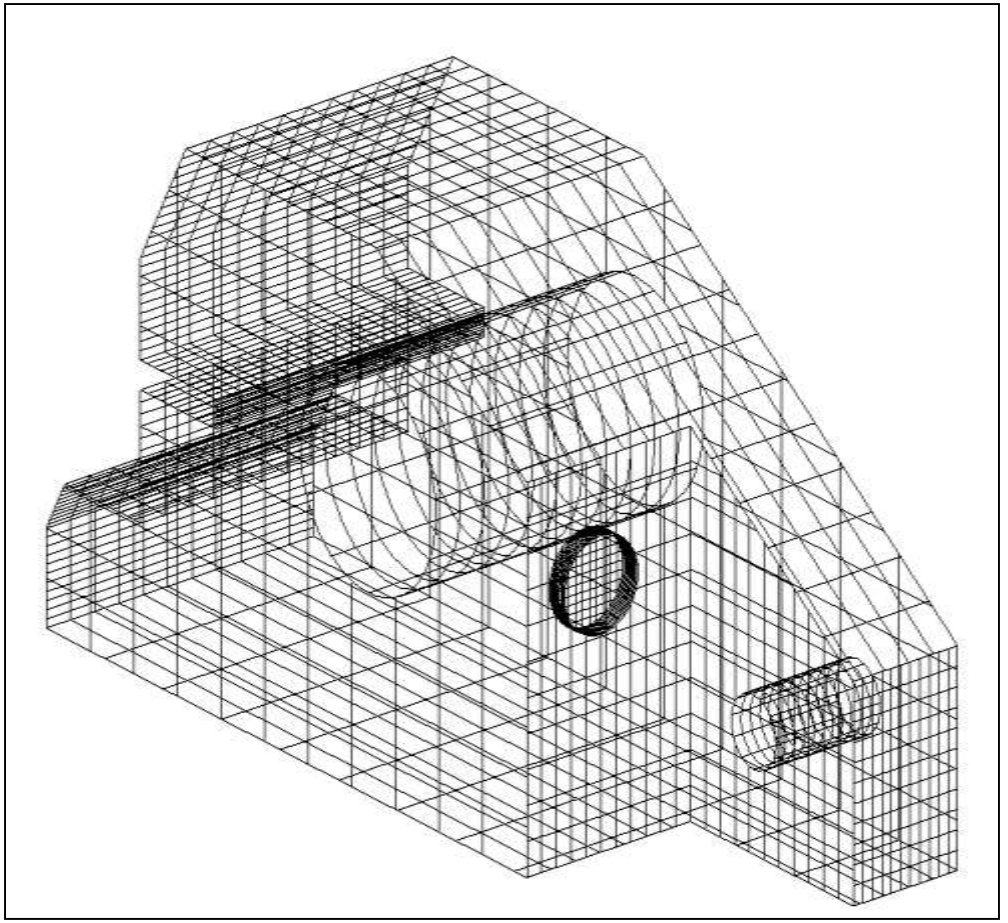
Wireframe Ambiguity (Cont.)

*Which face is **front** and which is **back**?*



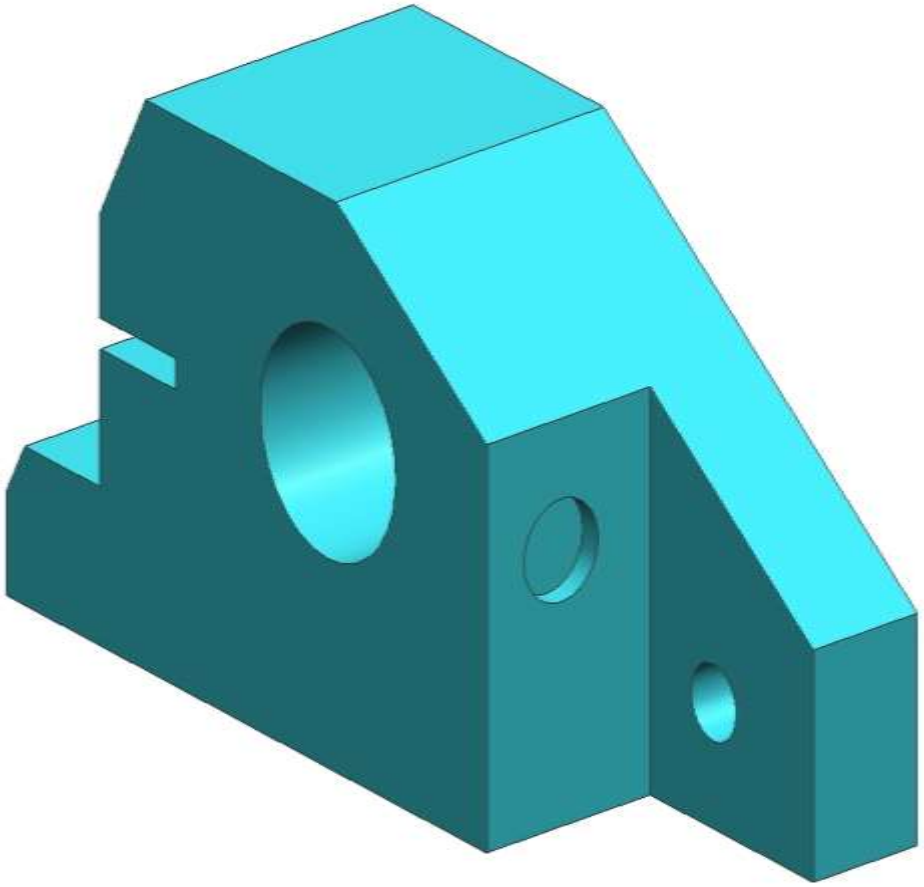
Surface Models

- **Came after Wireframe models**
- **Still used today**
- **Model Contains edges and vertices and exterior surfaces**
- **Can represent complex exterior surfaces**
- **No details regarding interior of part**
- **Too ambiguous for engineering analysis**



Solid Models

- **Current "state of the art"**
- **Model Contains edges and vertices , exterior surfaces, and interior details**
- **Part is unambiguously defined**
- **May be used for engineering analysis**



Motivation

Applications and uses for solid representations:

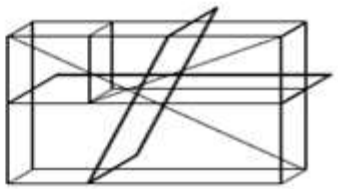
- CAD/CAM
- Finite-element analysis
- Stereolithography
- Component layout
- Interference fit
- Animation and interaction
- Engineering
- Medical Imaging
- Scientific Visualisation

Why Solid Modeling?

- Recall weakness of wireframe and surface modeling
  - Ambiguous geometric description
  - incomplete geometric description
  - lack topological information
  - Tedious modeling process
  - Awkward user interface

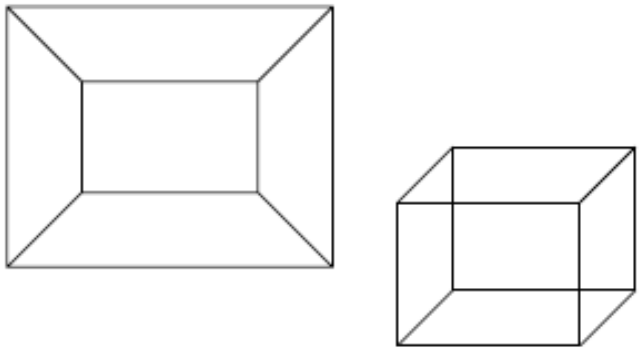
Disadvantages of Wireframe Models

- Ambiguity
- Subjective human interpretation
- Complex objects with many edges become confusing
- Lengthy and verbose to define
- Not possible to calculate Volume and Mass properties, NC tool path, cross sectioning etc

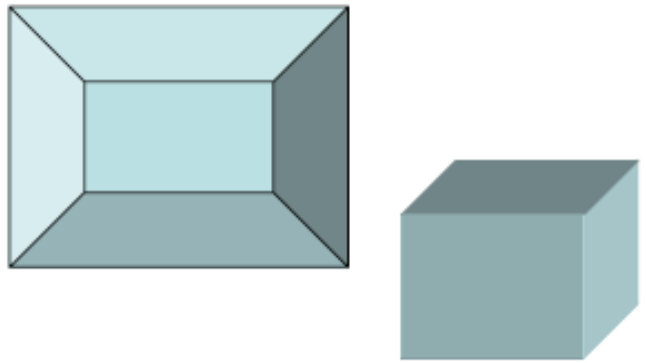


Definition of a Solid Model

A solid model of an object is a more complete representation than its surface (wireframe) model. It provides more topological information in addition to the geometrical information which helps to represent the solid unambiguously.



Wireframe Model



Solid Model

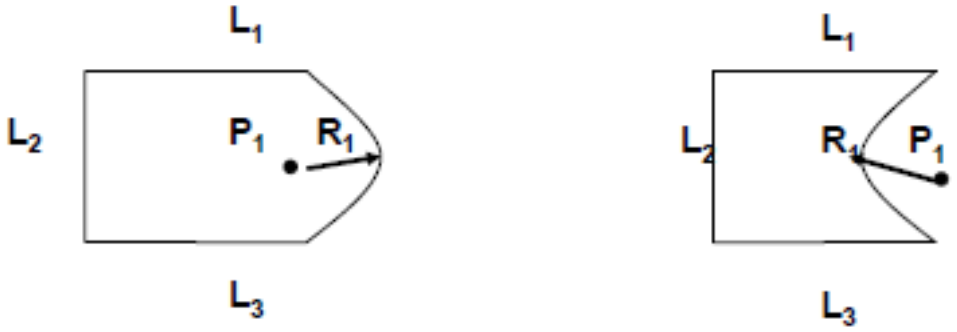
Geometry Vs Topology

Geometry:

Metrics and dimensions of the solid object. Location of the object in a chosen coordinate system

Topology:

Combinatorial information like connectivity, associativity and neighborhood information. Invisible relationship information.



Same geometry and different Topology

Solid model

- **Solid modeling** is based on *complete, valid and unambiguous* geometric representation of physical object.
  - **Complete** → points in space can be classified.(inside/ outside)
  - **Valid** →vertices, edges, faces are connected properly.
  - **Unambiguous** → there can only be one interpretation of object
- Analysis automation and integration is possible only with solid models→ has properties such as weight, moment of inertia, mass.
- **Solid model consist of geometric and topological data**
  - Geometry → shape, size, location of geometric elements
  - Topology →connectivity and associatively of geometric elements →non graphical, relational information

Advantages of Solid Models

Unlike wireframes and surface representations which contain only geometrical data, the solid model uses topological information in addition to the geometrical information to represent the object unambiguously and completely. Solid model results in accurate design, helps to further the goal of CAD/ CAM like CIM, Flexible manufacturing leading to better automation of the manufacturing process.

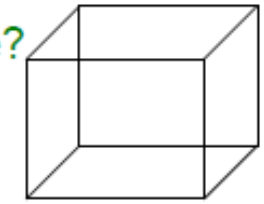
Geometry: The graphical information of dimension, length, angle, area and transformations

Topology: The invisible information about the connectivity, neighborhood, associatively etc

Is a solid model just a shaded image?

Three dimensional addressability?

Suitable for automation?



Wireframe Model

Solid Model

Manifold Vs Non-manifold

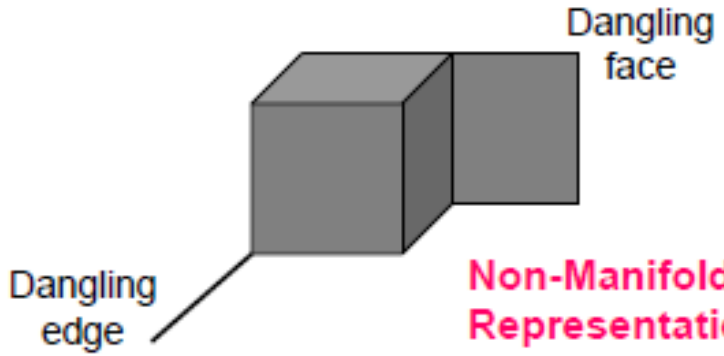
Two Manifold Representations:

Two manifold objects are well bound, closed and homomorphic to a topological ball.

Non-manifold Representations:

When restrictions of closure and completeness are removed from the solid definition, wireframe entities can coexist with volume based solids.

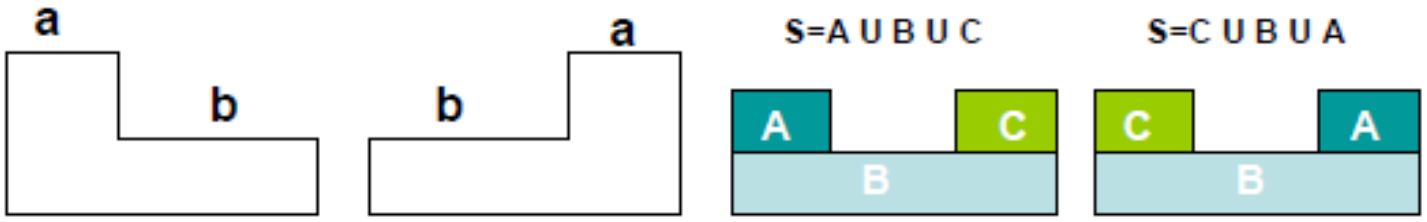
Two-Manifold Object



Non-Manifold Representation

Common Features of Modeling Schemes

- **Domain**: The type of objects that can be represented or the geometric coverage.
- **Validity**: The resulting solid model when subjected to algorithms should succeed as valid solid.
- **Completeness** and **Unambiguousness**.
- **Uniqueness**: Positional and Permutational uniqueness.

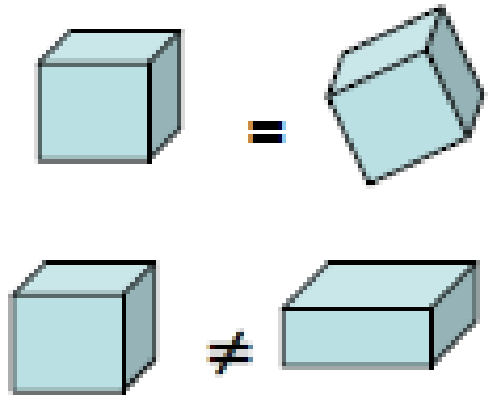


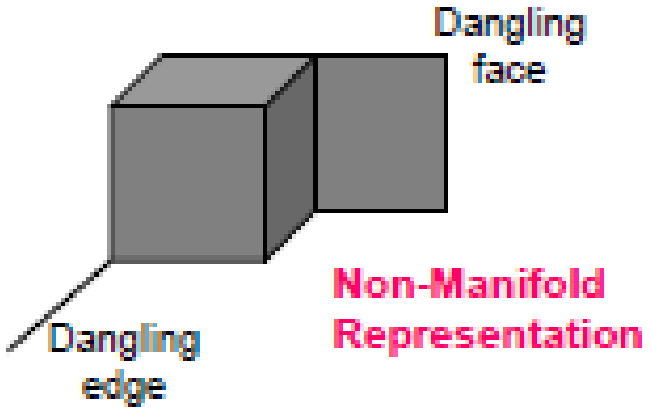
Positionally non-unique

Permutationally non-unique

Properties of Solid Models

- **Rigidity:** Shape of the solid is invariant w.r.t location/orientation
- **Homogeneous 3-dimensionality**: The solid boundaries must be in contact with the interior. No isolated and dangling edges are permitted.





Properties of Solid Models

- **Finiteness and finite describability:** Size is finite and a finite amount of information can describe the solid model.
- **Closure under rigid motion and regularised Boolean operations**: Movement and Boolean operations should produce other valid solids.
- **Boundary determinism:** The boundary must contain the solid and hence determine the solid distinctly.
- **Any valid solid must be bounded, closed, regular and semi-analytic subsets of E^3**

Properties of Solid Models

- The regularized point sets which represent the resulting solids from Boolean operations are called **r-sets (regularized sets**.
- **R-sets** can be imagined as curved polyhedra with well behaved boundaries.

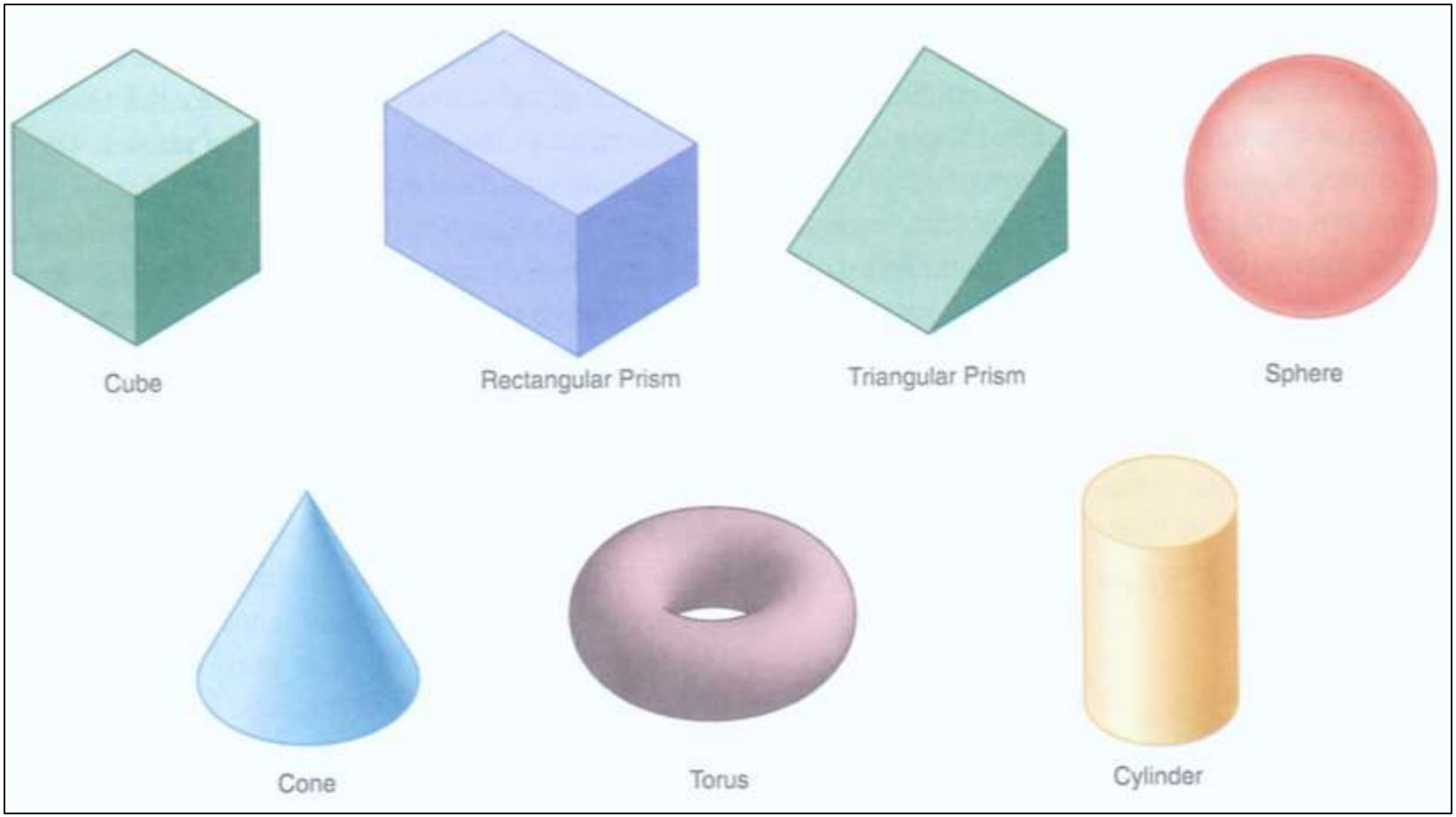
Desirable Properties of Modeling Schemes

- Conciseness, ease of creation and efficiency in application.
- **Conciseness means compact database to represent the object.**
- **Ease of operation implies user-friendliness. Most of the packages use CSG approach to create the solids**
- The representation must be usable for later operations like CAM / CNS etc

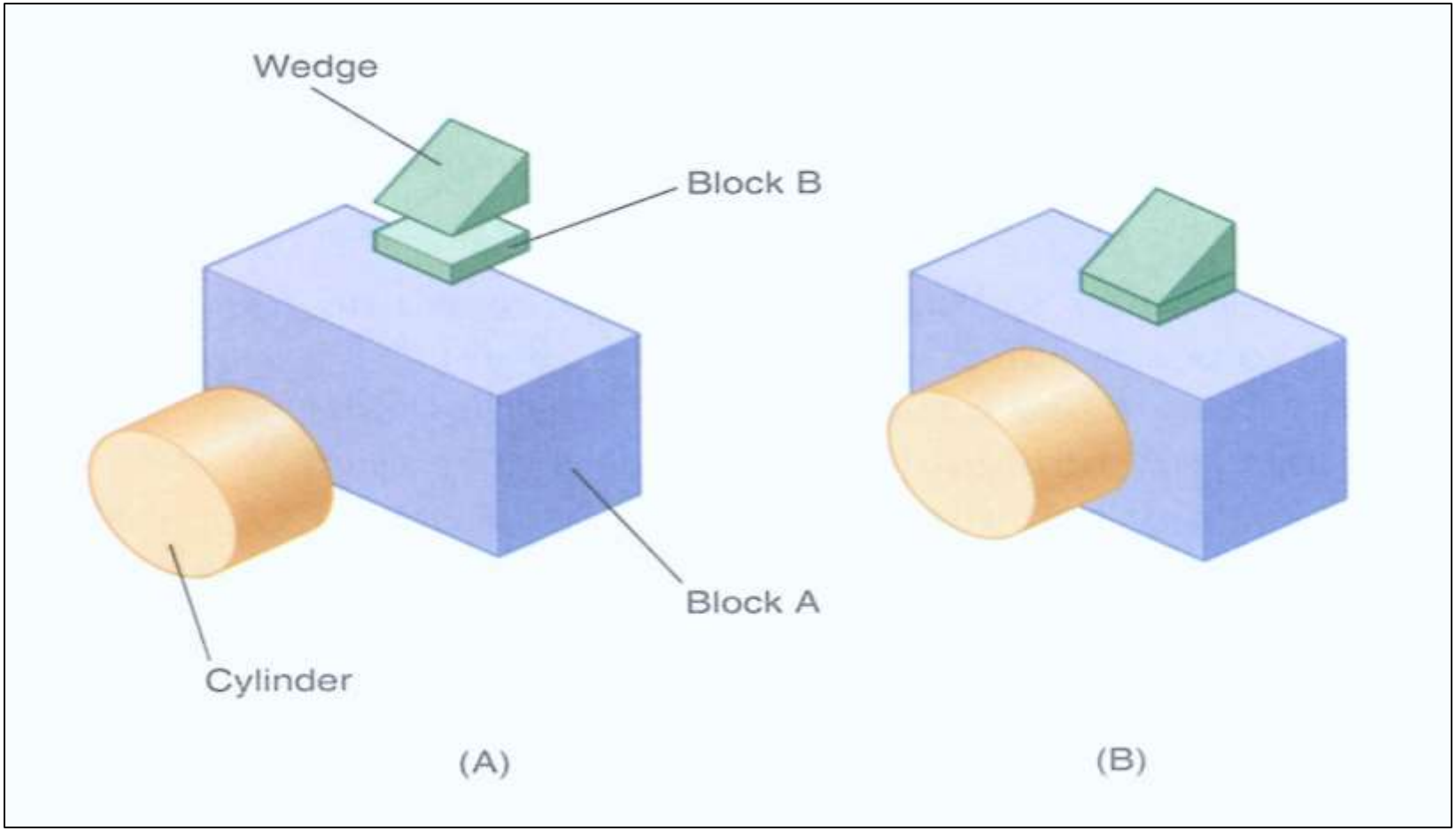
Solid model representation schemes

1. Primitive instancing
2. Regularized Boolean set operations
3. Sweep representations.
4. Boundary representations (B-reps)
5. Constructive solid geometry (CSG)
6. Spatial-partitioning representations
   - Octree representation

Primitives


Cube
Rectangular Prism
Triangular Prism
Sphere
Cone
Torus
Cylinder

Primitives in-use



Primitive Instancing

This is where objects are defined as a single primitive geometric object and scaled, translated and rotated into the world.

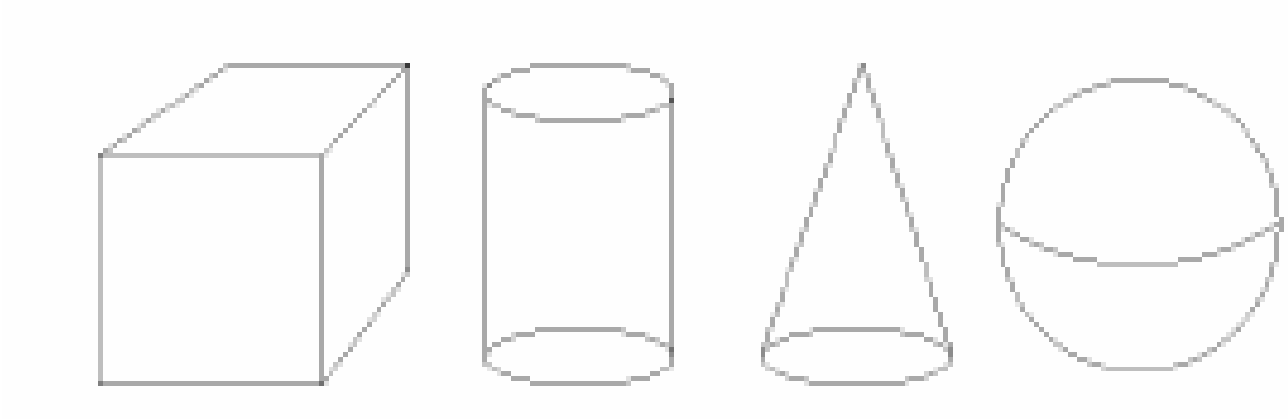
This is what you have been doing in the labs with OpenGL display lists.

Although the simplest method to implement, it fails on almost every consideration. E.g. how do you define the boolean set operations?

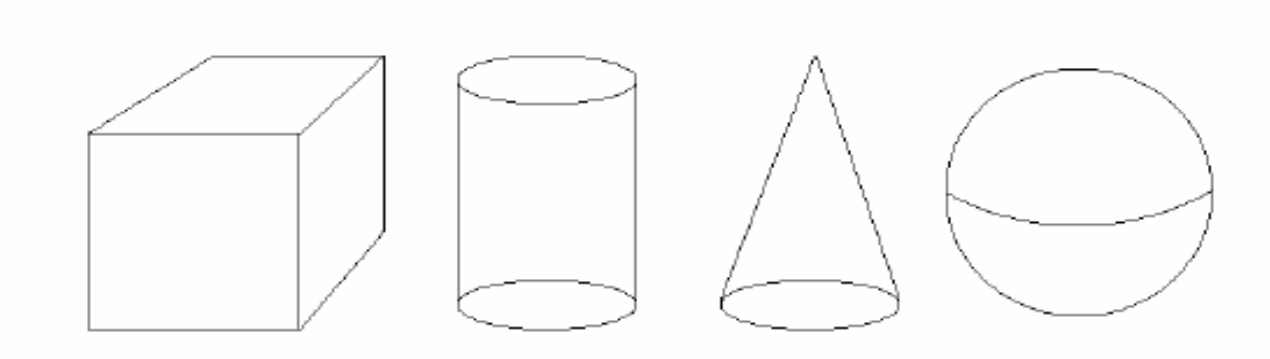
Other problems:

- No provision for combining objects
- Each model must start from scratch
- E.g. volume calculations are different for each model

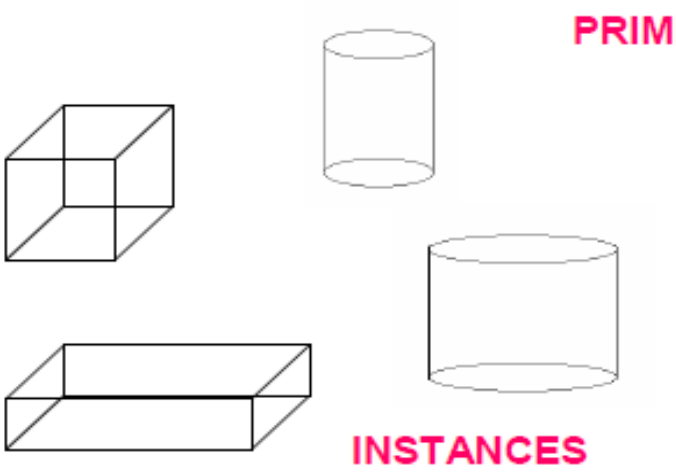
Geometric Modeling - Primitives



Primitives and Instances



PRIMITIVES

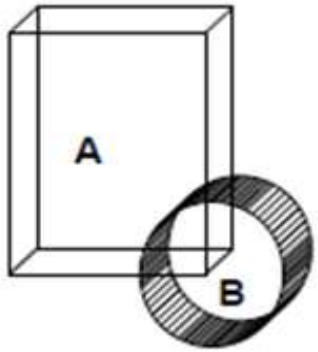


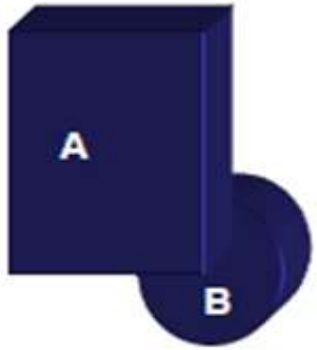
INSTANCES

Instancing: An instance is a scaled / transformed replica of its original. The scaling may be uniform or differential. It is a method of keeping primitives in their basic minimum condition like unit cube and unit sphere etc.

Operation on Primitives

- A desired solid can be obtained by combining two or more solids
- When we use Boolean (set) operations the validity of the third (resulting) solid is ensured



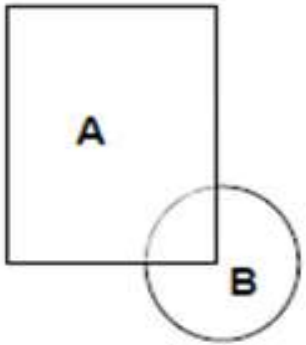


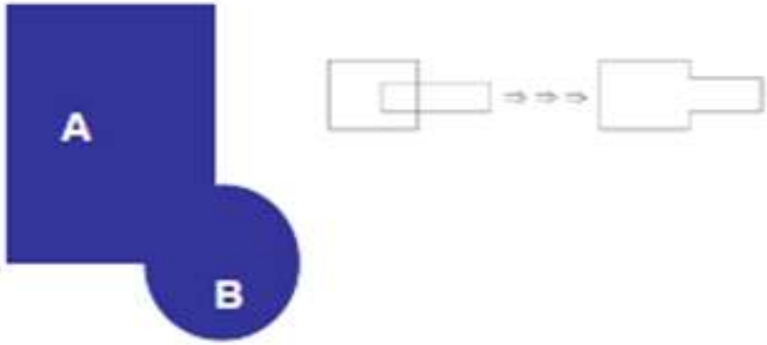
3 Dimensional

UNION: BLOCK ∪ CYLINDER $A \cup B$

Operation on Primitives

- A desired solid can be obtained by combining two or more solids
- When we use Boolean (set) operations the validity of the third (resulting) solid is ensured

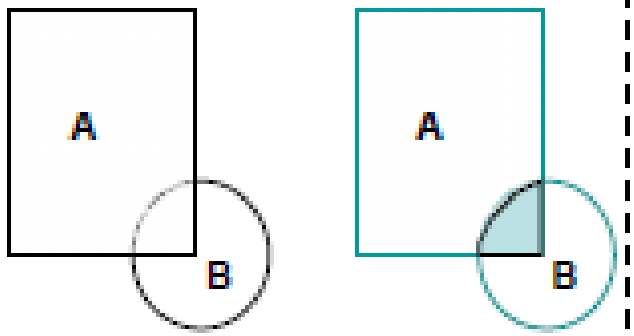




UNION: BLOCK ∪ CYLINDER

$A \cup B$

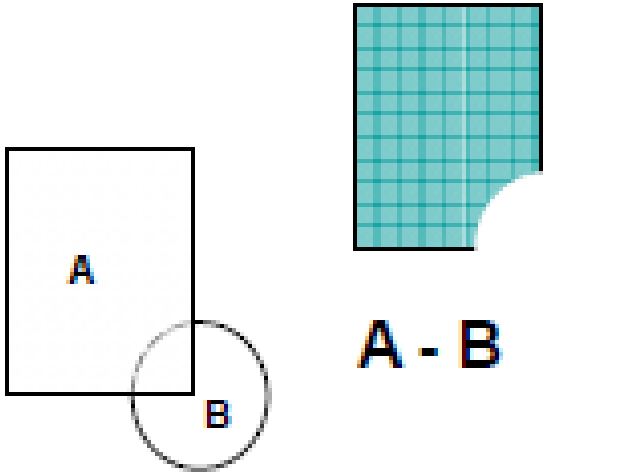
Operation on Primitives



INTERSECTION:

BLOCK ∩ CYLINDER

A ∩ B



A - B

B - A

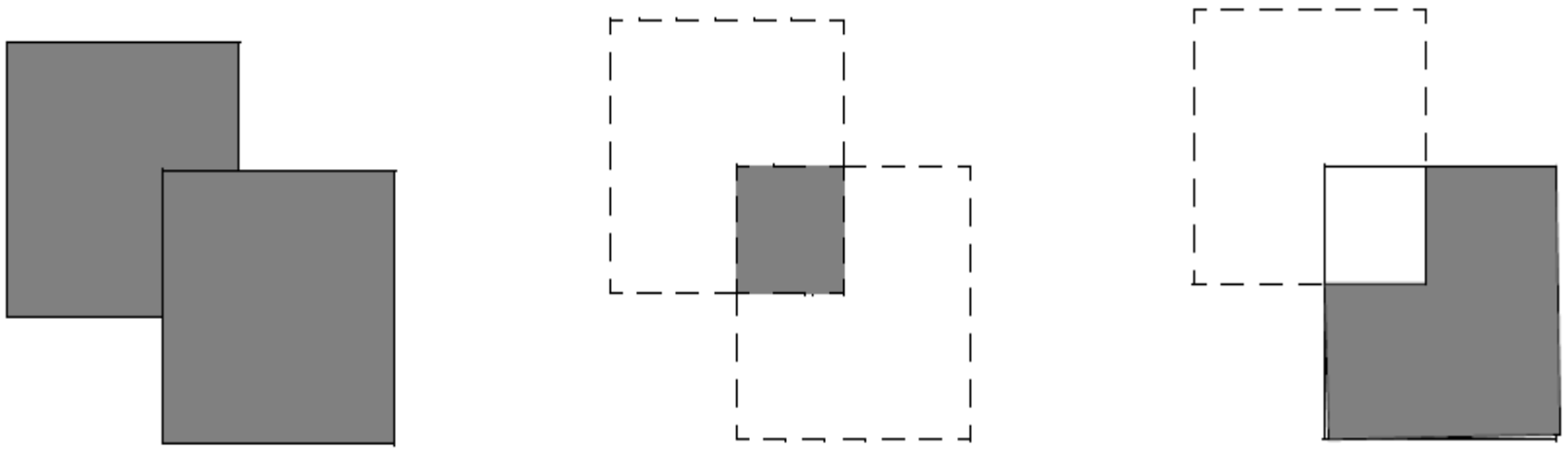
DIFFERENCE:

BLOCK - CYLINDER

Regularized Boolean Set Operations

Normal boolean set operations are union, intersection and subtraction. Applied to volumes these operations not closed, because they can yield points, lines or planes.

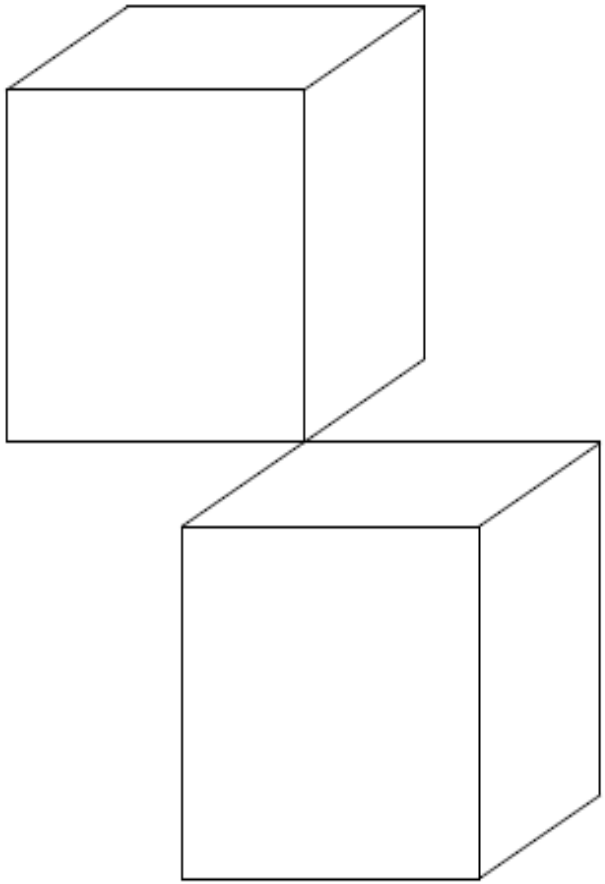
Regularized boolean operations always yield volumes, so the intersection of 2 cubes sharing an edge is NULL. It would be a line in a normal boolean operation.



union

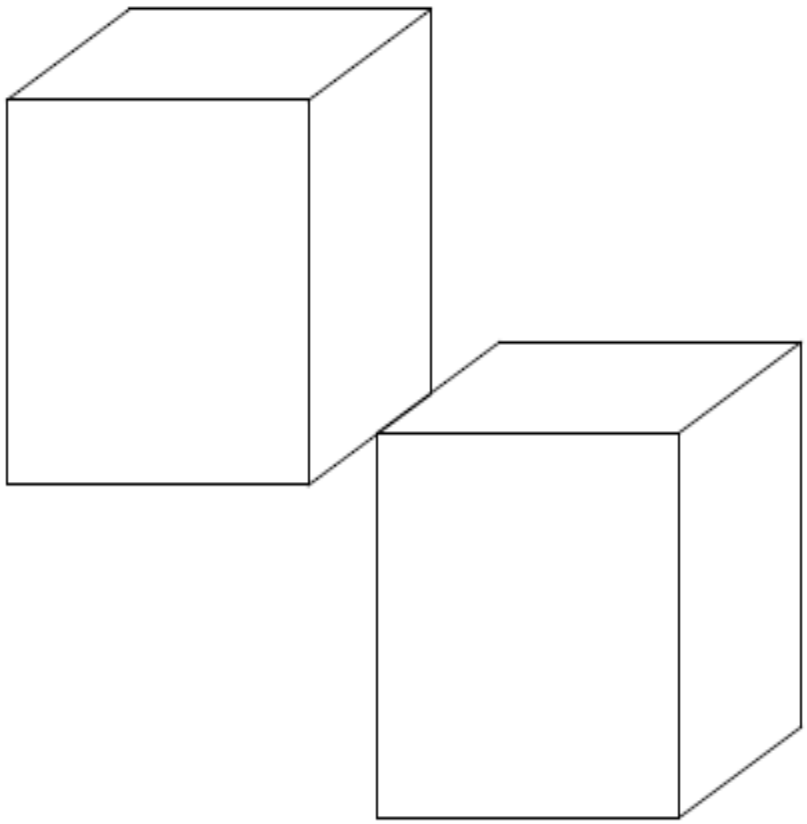
intersection

subtraction



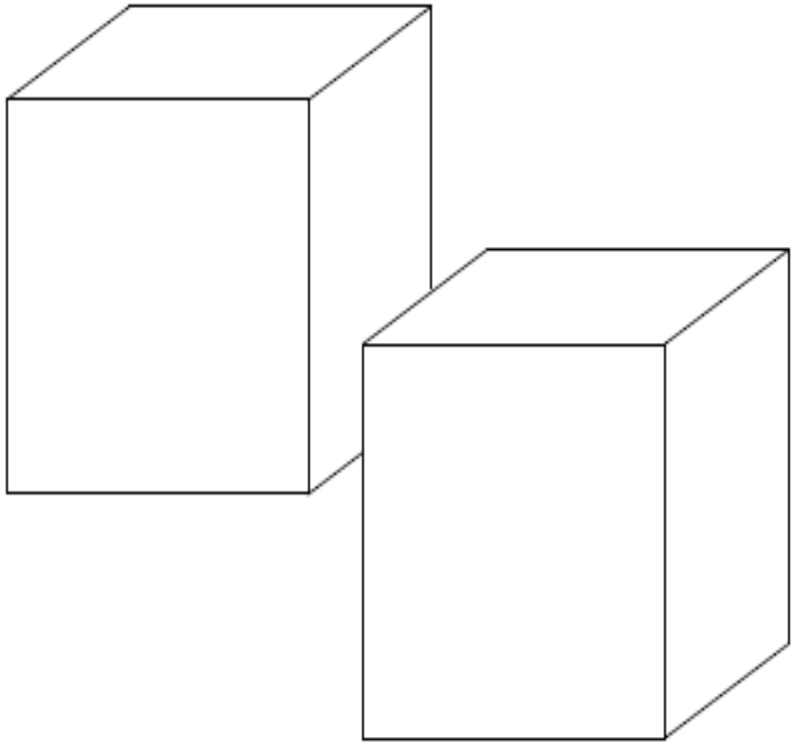
Intersection = •

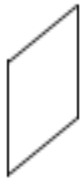
i.e. a point in 3D space, not a solid model



Intersection = /

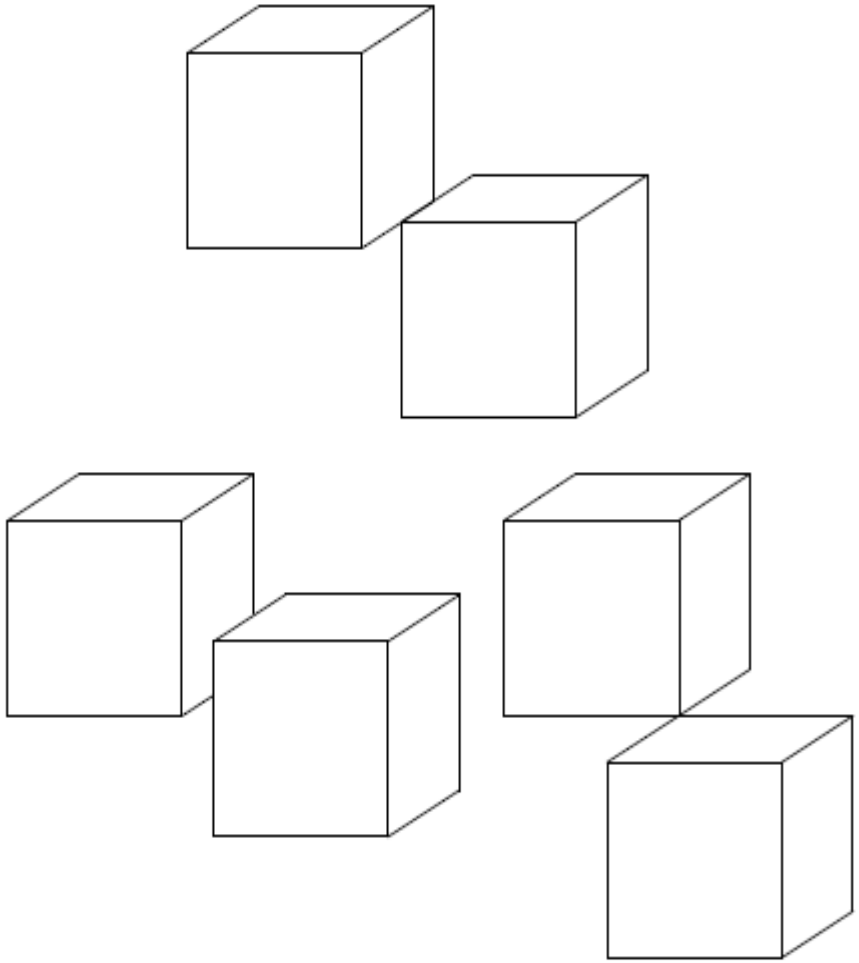
i.e. a 1D line through 3D space, not a solid model



Intersection = 

i.e. a 2D plane, not a solid model

Regularized Boolean Set Operations



Using *regularized* boolean operators:

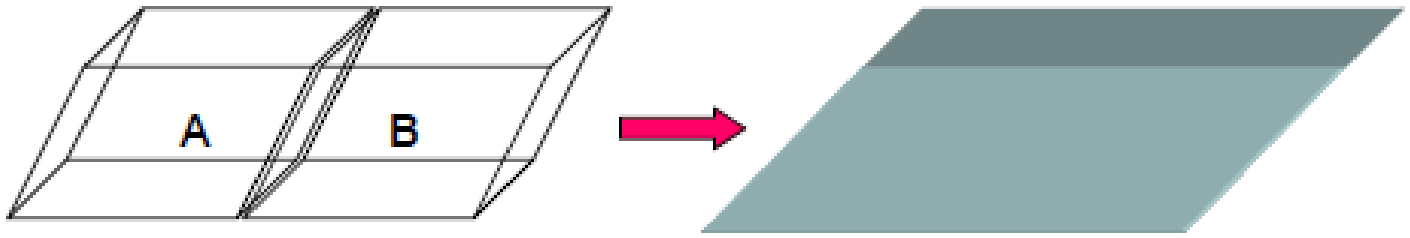
All 3 intersections = NULL

Effectively, we throw away any results from an operation that is of lower dimensionality than the original solids.

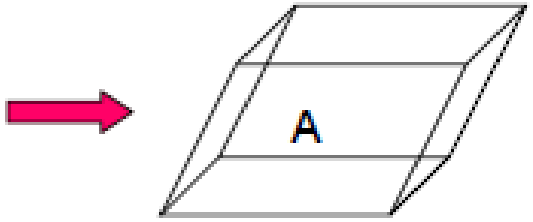
Regularized set Operations

- The interior of the solid is geometrically closed by its boundaries

$$A \cup^* B$$



$$A -^* B$$



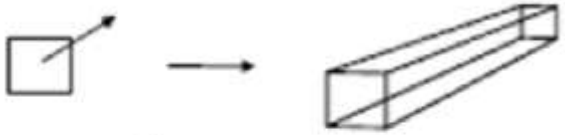
$$A \cap^* B = \varphi$$

Sweep Volume

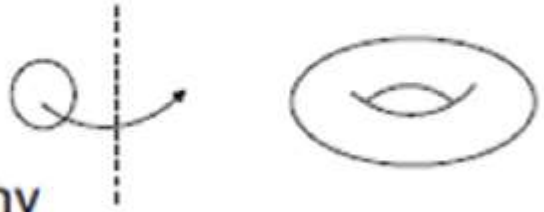
- Sweep Volume: sweeping a 2D area along a trajectory creates a new 3D object
- Translational Sweep: 2D area swept along a linear trajectory normal to the 2D plane

- Tapered Sweep: scale area while sweeping

- Slanted Sweep: trajectory is not normal to the 2D plane

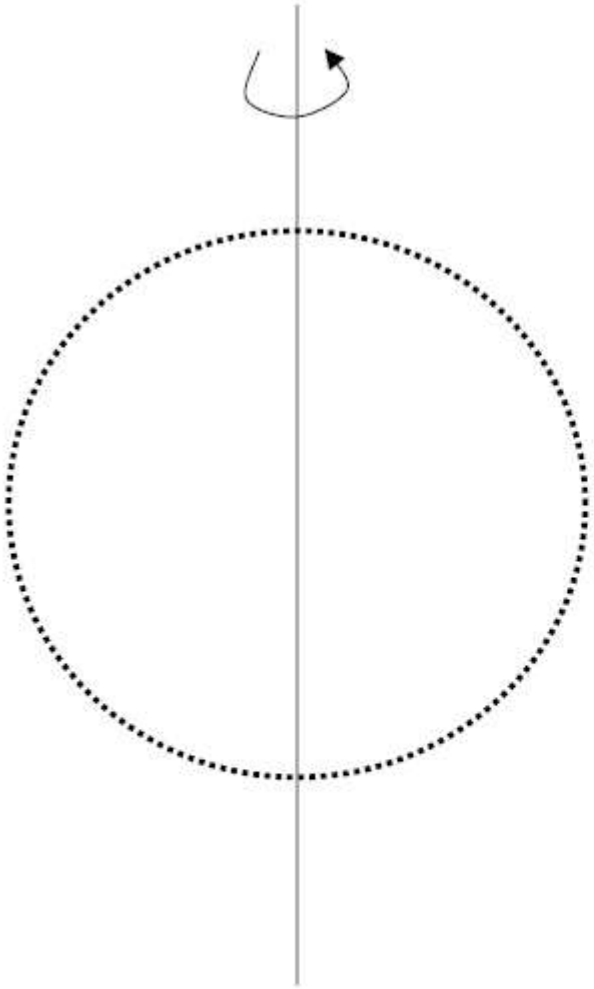


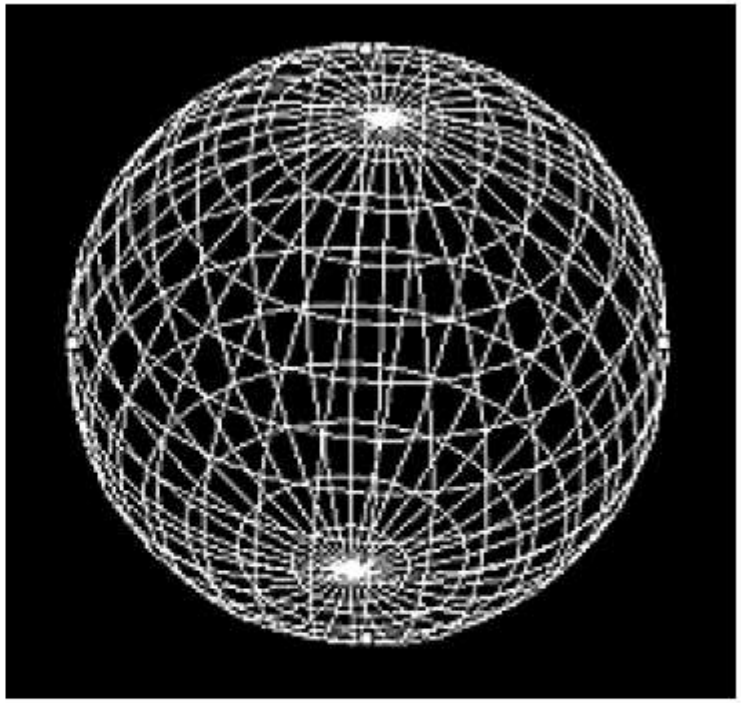
- Rotational Sweep: 2D area is rotated about an axis



- General Sweep: object swept along any trajectory and transformed along the sweep

Rotational Sweep





Boundary representation (B-Rep)

- Solid model is defined by their enclosing surfaces or boundaries. This technique consists of the geometric information about the faces, edges and vertices of an object with the topological data on how these are connected.
- Why B-Rep includes such topological information?
  - A solid is represented as a closed space in 3D space (surface connect without gaps)
  - The boundary of a solid separates points inside from points outside solid.
- Surface model
  - A collection of surface entities which simply enclose a volume lacks the connective data to define a solid (i.e topology).
- B- Rep model
  - Technique guarantees that surfaces definitively divide model space into solid and void, even after model modification commands.
  - B-Rep graph store face, edge and vertices as nodes, with pointers, or branches between the nodes to indicate connectivity.

Boundary representation- validity

- System must validate topology of created solid.
- B-Rep has to fulfill certain conditions to disallow self-intersecting and open objects
- This condition include
  - Each edge should adjoin exactly two faces and have a vertex at each end.
  - Vertices are geometrically described by point coordinates
  - At least three edges must meet at each vertex.
  - Faces are described by surface equations
  - The set of faces forms a complete skin of the solid with no missing parts.
  - Each face is bordered by an ordered set of edges forming a closed loop.
  - Faces must only intersect at common edges or vertices.
  - The boundaries of faces do not intersect themselves

B-reps (Boundary Representations)

Object is defined in terms of its surface boundaries, vertices, edges and faces. Curved surfaces are always approximated with polygons – piecewise linear/planar.

Very commonly used, in practice.

Use planar, polygonal boundaries, but may also use convex polygons or triangles.

Polyhedron is a solid that is bounded by a set of polygons whose edges are each a member of an even number of polygons. Additional constraints will be discussed later on.

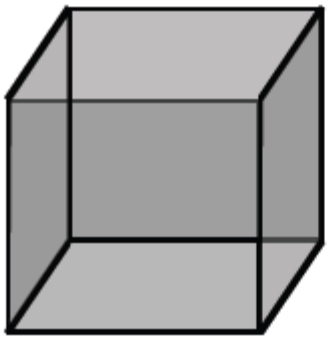
A simple polyhedron can always be deformed into a sphere. Polyhedron has no holes (not a torus).

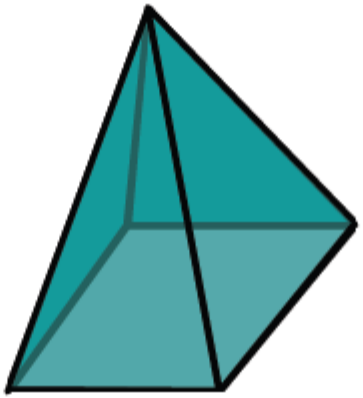
Euler's formula:

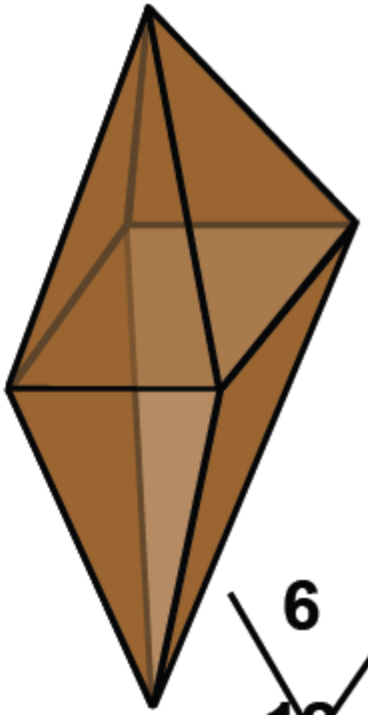
Let V be the number of vertices, E the number of edges, and F the number of faces of a simple polyhedron. Then

$$V - E + F = 2$$

Verify Euler's formula with these examples:



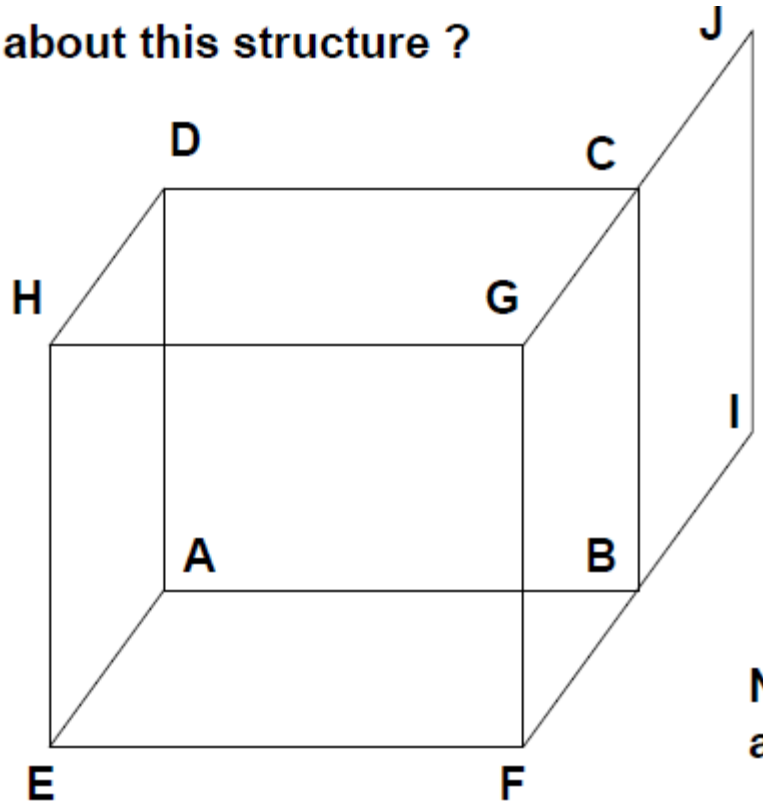




V =	8	5	6
E =	12	8	12
F =	6	5	9
			6
		Actual:	12
			8

Also applicable for curved edges and non-planar faces

What about this structure ?



V = 10;

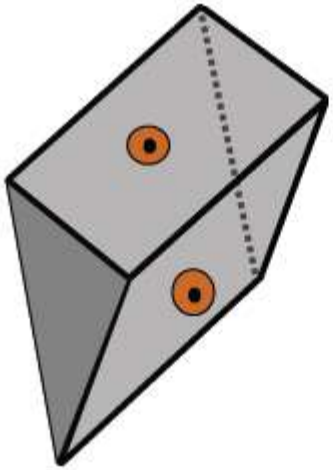
E = 15;

F = 7.

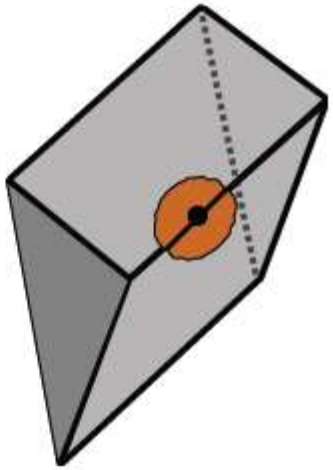
Formula still holds good,
but this is not a bound volume.

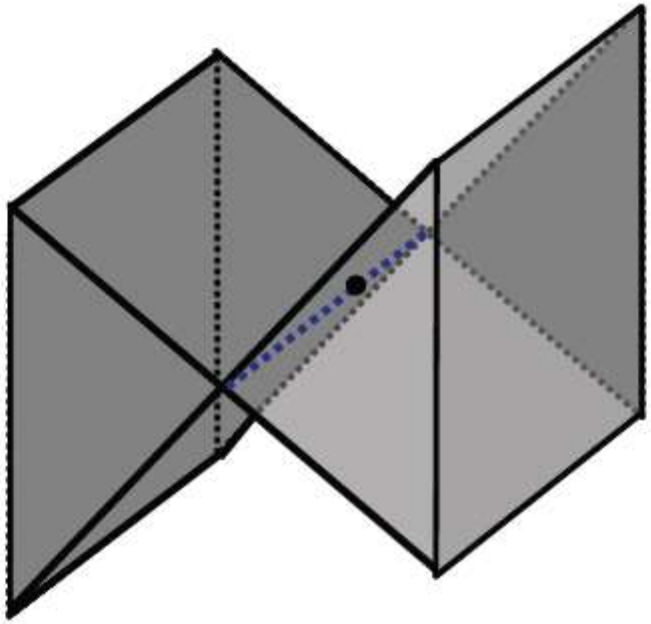
Need additional constraints to guarantee a solid object:

- Each edge must contain two vertices
- Must be shared exactly by two faces
- At least three edges must meet to form a vertex
- Faces must not interpenetrate



Solids with boundaries with 2-manifolds





Solids with boundaries but not 2-manifolds

Generalized Euler's formula for objects with 2-manifolds, and faces which may have holes:

$$V - E + F - H = 2(C - G)$$

where, H - No. of **HOLES** in the face;

G - No. of holes that pass through the object;

C - No. of separate **COMPONENTS** (or parts) of the object.

If C = 1, G is called as ***GENUS.*** If C > 1, G is the sum of the **GENERA** of its components.

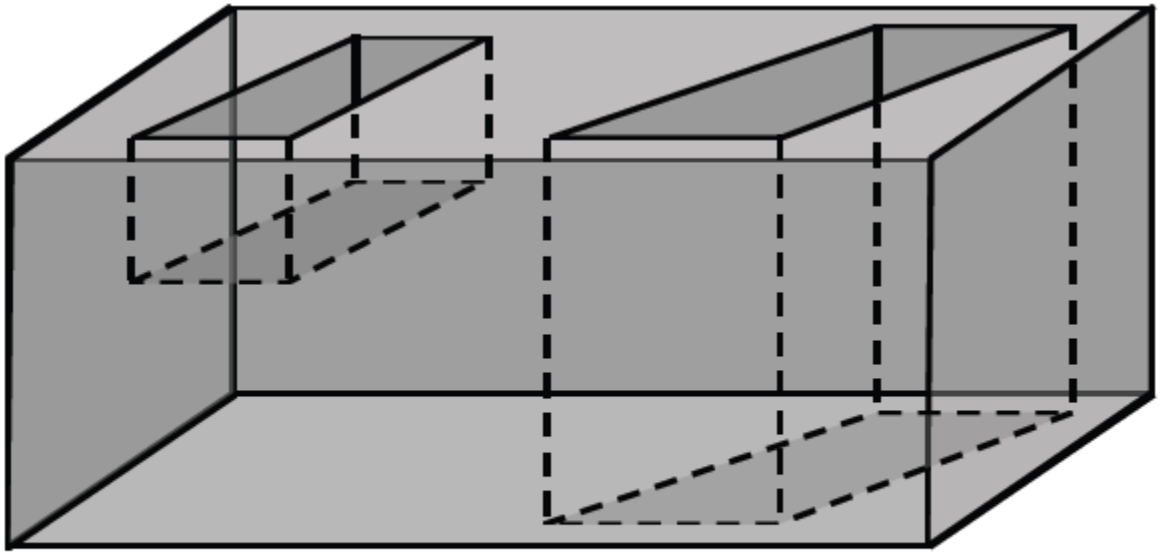
V = 24

E = 36

F = 15

H = 3

C = G = 1

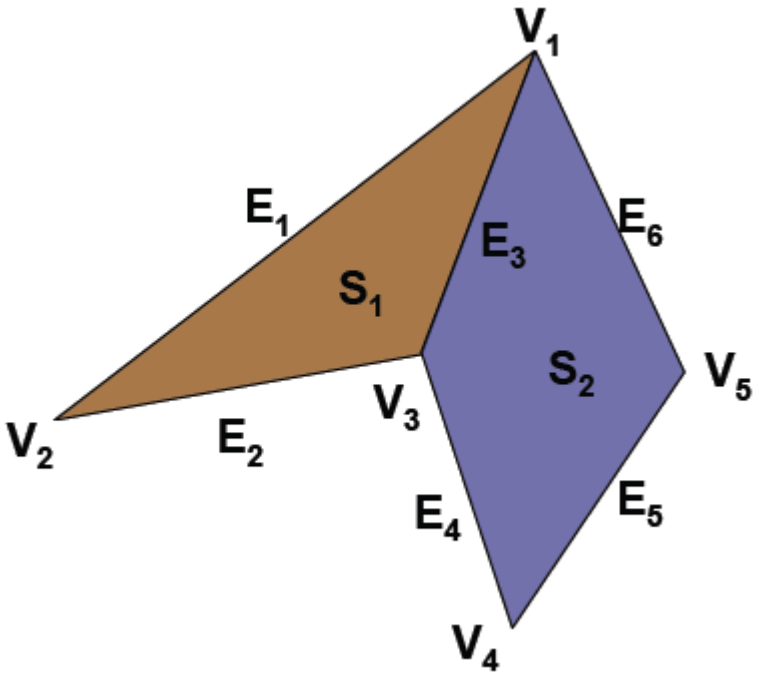


Vertex Table

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 Table

E_1: V_1, V_2
E_2: V_2, V_3
E_3: V_3, V_1
E_4: V_3, V_4
E_5: V_4, V_5
E_6: V_5, V_1



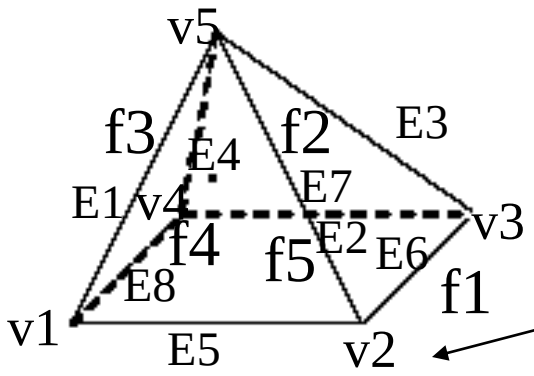
Polygon Surface Table

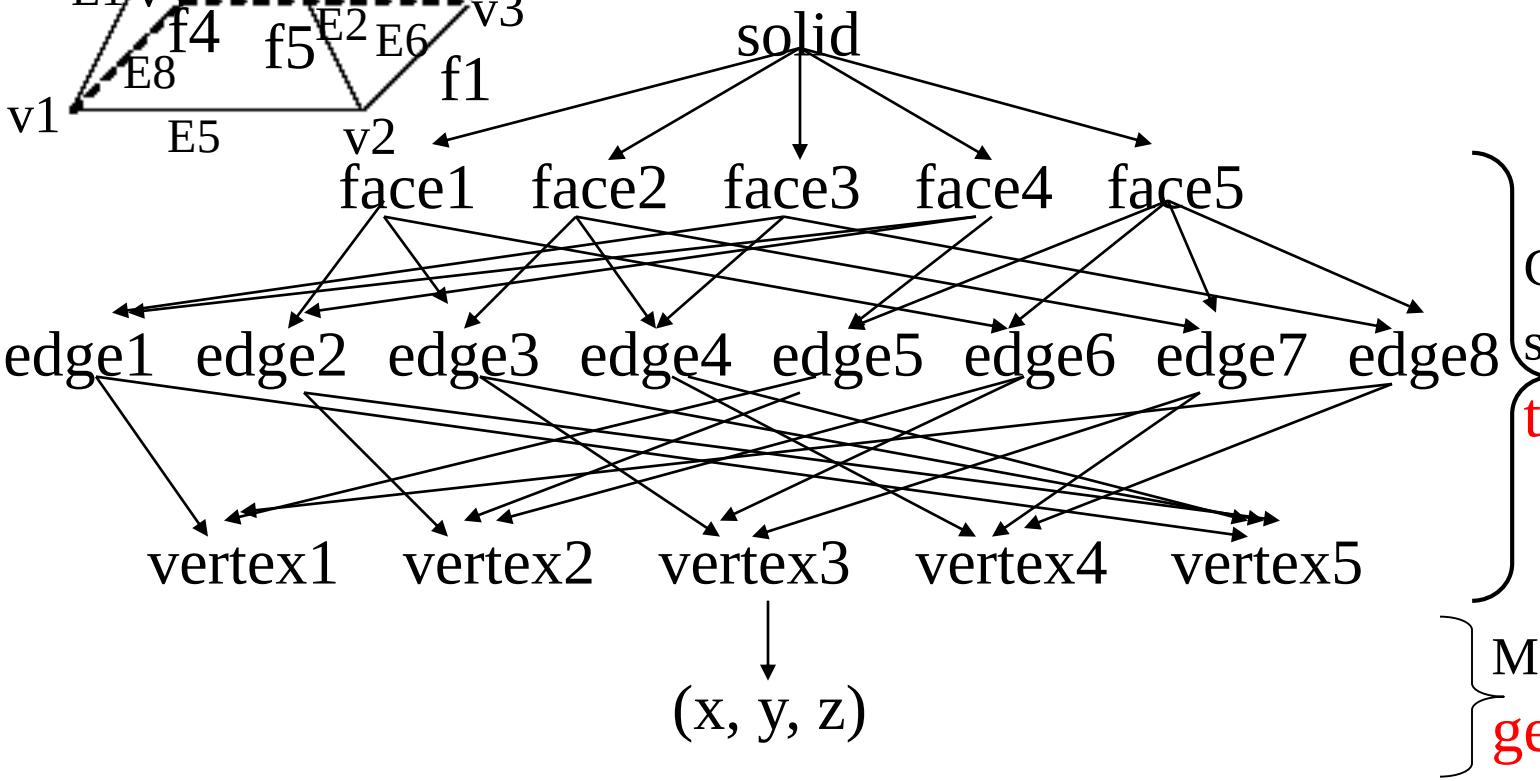
S_1: E_1, E_2, E_3
S_2: E_3, E_4, E_5, E_6

Complete (expanded) Edge Table

E_1: V_1, V_2, S_1
E_2: V_2, V_3, S_1
E_3: V_3, V_1, S_1, S_2
E_4: V_3, V_4, S_2
E_5: V_4, V_5, S_2
E_6: V_5, V_1, S_2

B-Rep data structure





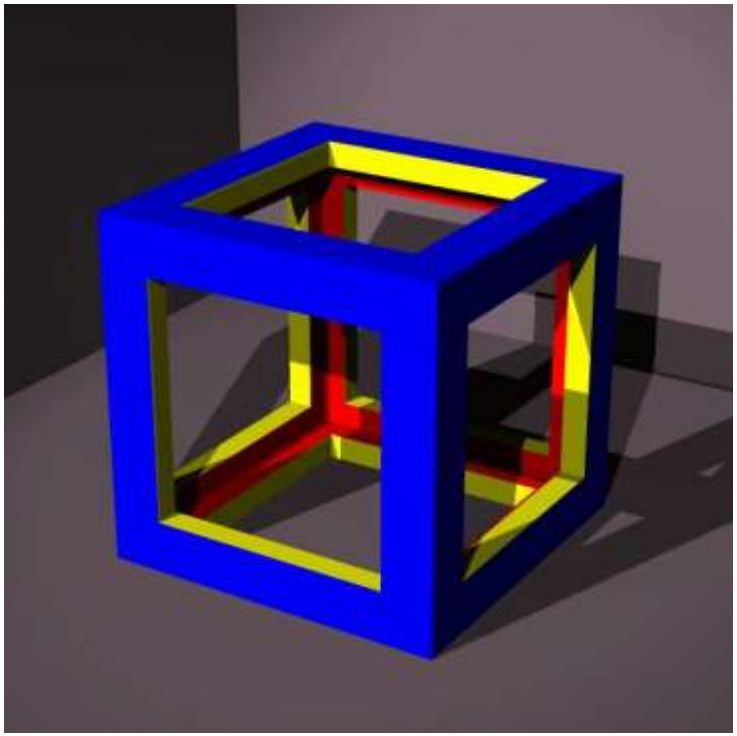
Combinatorial structure / topology

Metric information/ geometry

Boundary representation- ambiguity and uniqueness

- Valid B-Reps are unambiguous
- Not fully unique, but much more so than CSG
- Potential difference exists in division of
  - Surfaces into faces.
  - Curves into edges
- Capability to construct unusual shapes that would not be possible with the available CSG→ aircraft fuselages, swing shapes
- Less computational time to reconstruct the image
- Requires more storage
- More prone to validity failure than CSG
- Model display limited to planar faces and linear edges
  - complex curve and surfaces only approximated

Constructive Solid Geometry



Constructive solid geometry (CSG)

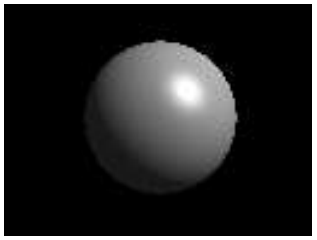
- Objects are represented as a combination of simpler solid objects (*primitives*).
- The primitives are such as cube, cylinder, cone, torus, sphere etc.
- Copies or “instances” of these primitive shapes are created and positioned.
- A complete solid model is constructed by combining these “instances” using set specific, logic operations (Boolean)
- Boolean operation
  - each primitive solid is assumed to be a set of points, a boolean operation is performed on point sets and the result is a solid model.
  - Boolean operation → union, intersection and difference
  - The relative location and orientation of the two primitives have to be defined before the boolean operation can be performed.
  - Boolean operation can be applied to two solids other than the primitives.

Constructive Solid Geometry Methods

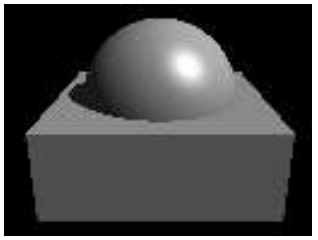
- Constructive Solid Geometry (CSG) performs solid modelling by generating a new object from two 3-dimensional objects using a set operation
- Valid set operations include
  - Union
  - Intersection
  - Difference

CSG Operations

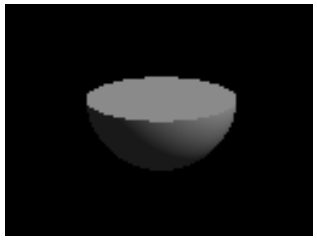
- Primitives:



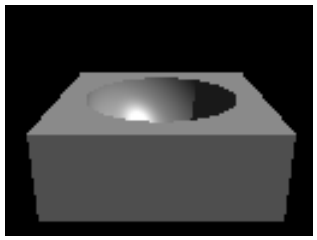
- Union:



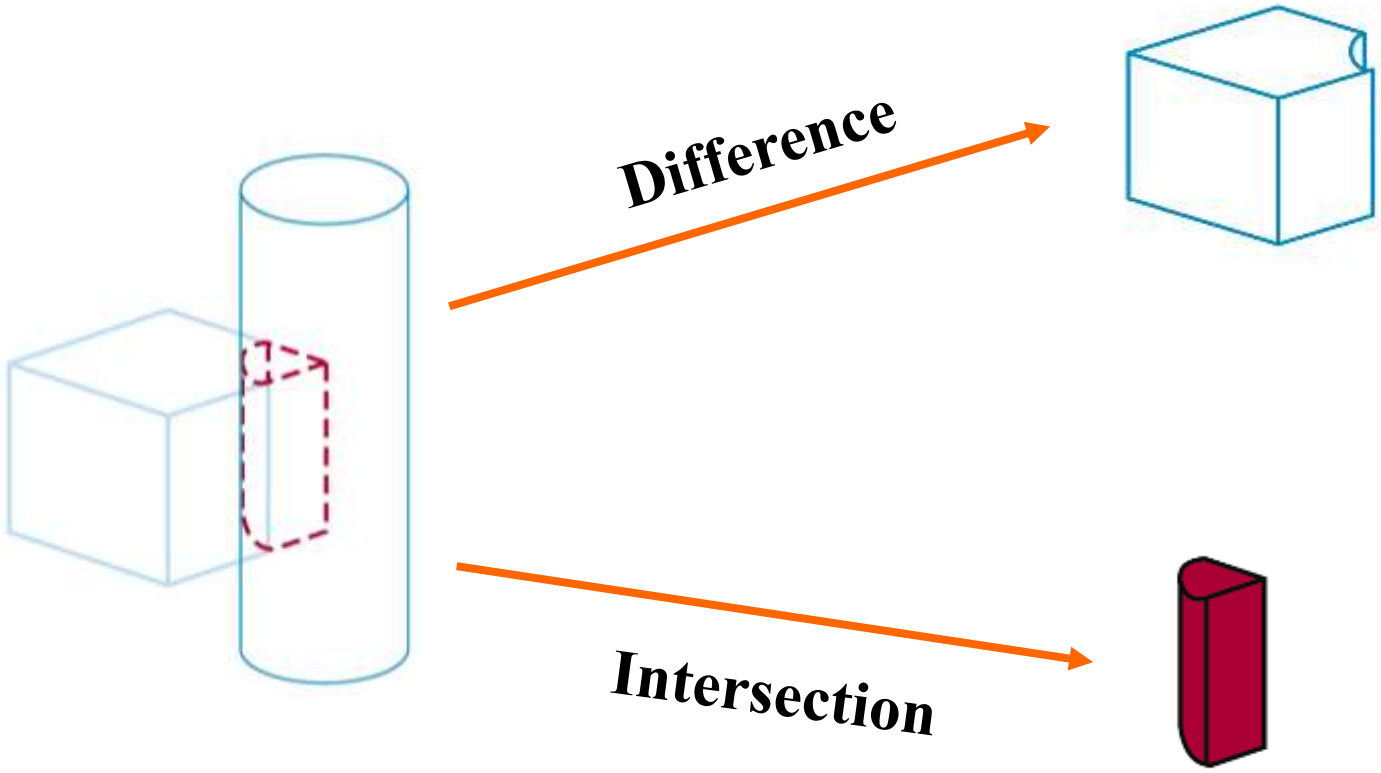
- Intersection:



- Difference:



Constructive Solid Geometry Methods (cont…)

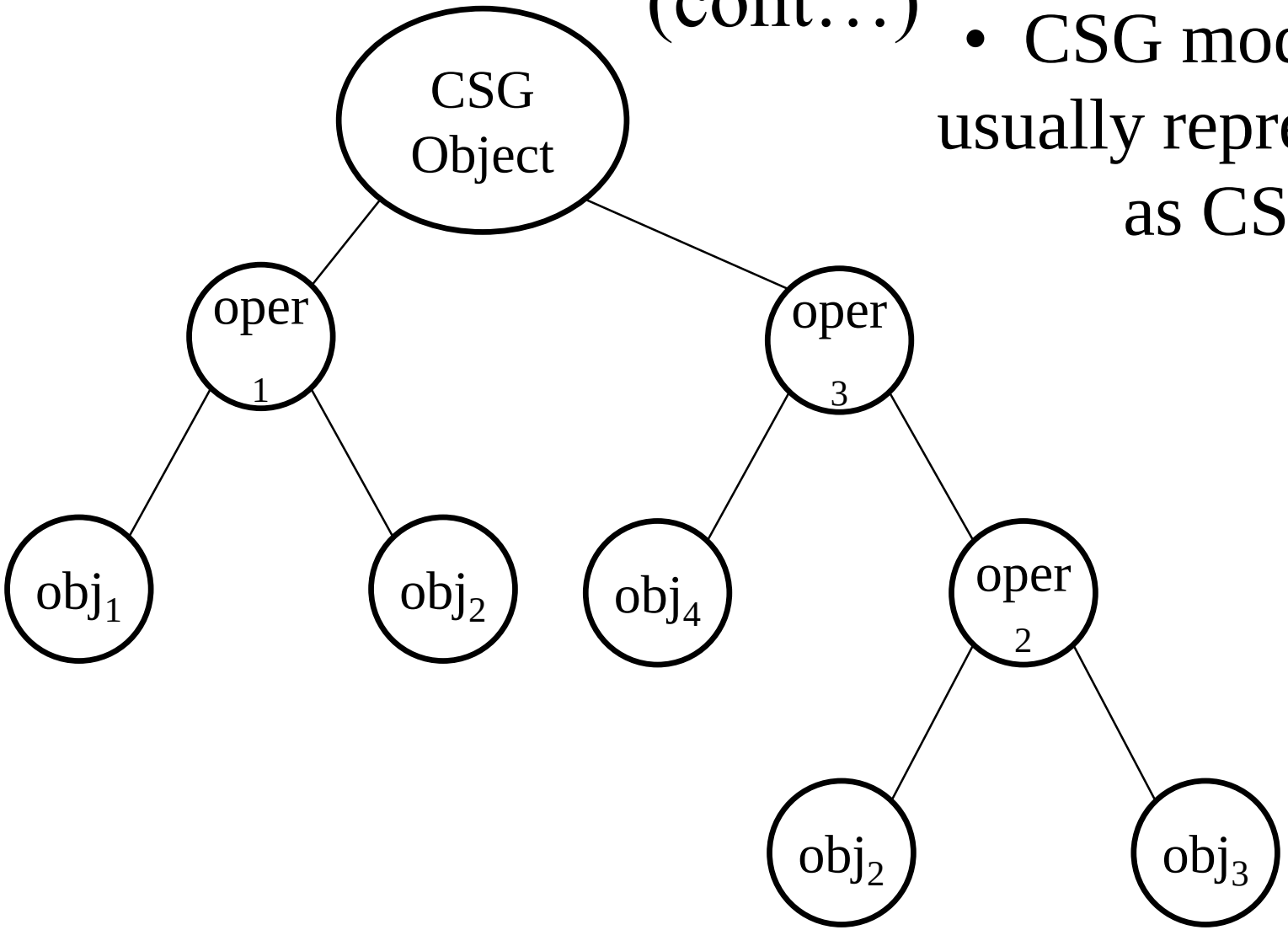


Constructive Solid Geometry Methods (cont…)

- CSG usually starts with a small set of primitives such as blocks, pyramids, spheres and cones
- Two objects re initially created and combined using some set operation to create a new object
- This object can then be combined with another primitive to make another new object
- This process continues until modelling complete

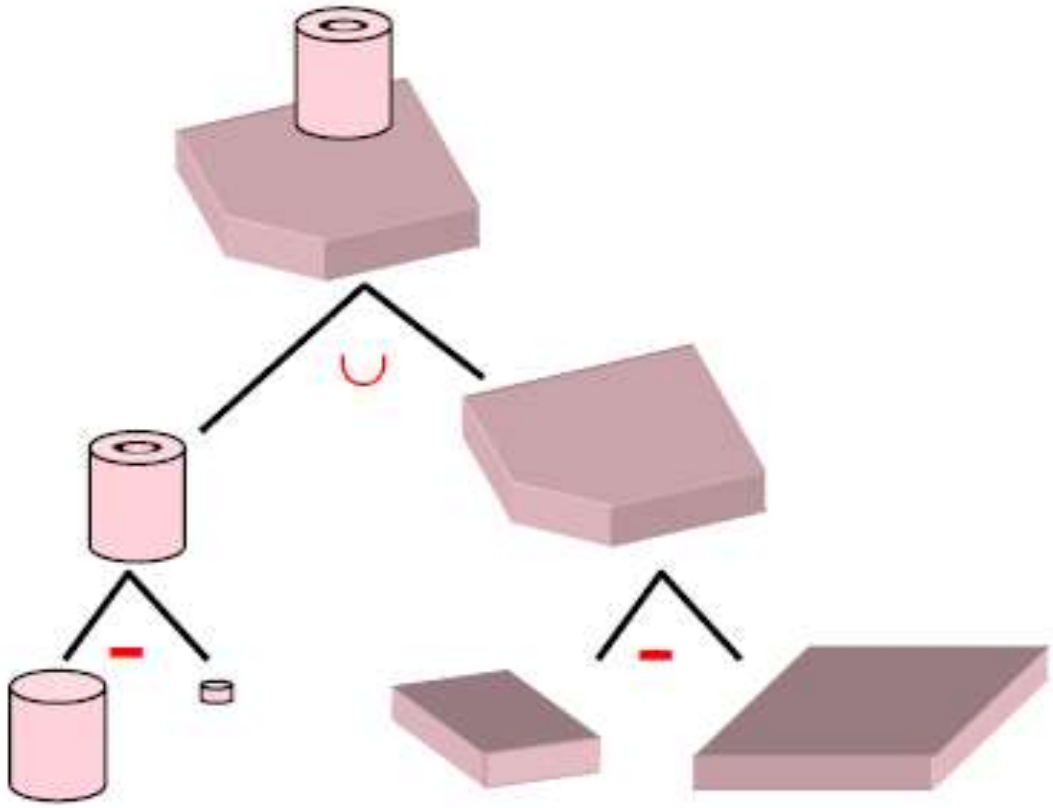
Constructive Solid Geometry Methods (cont…)

- CSG models are usually represented as CSG trees

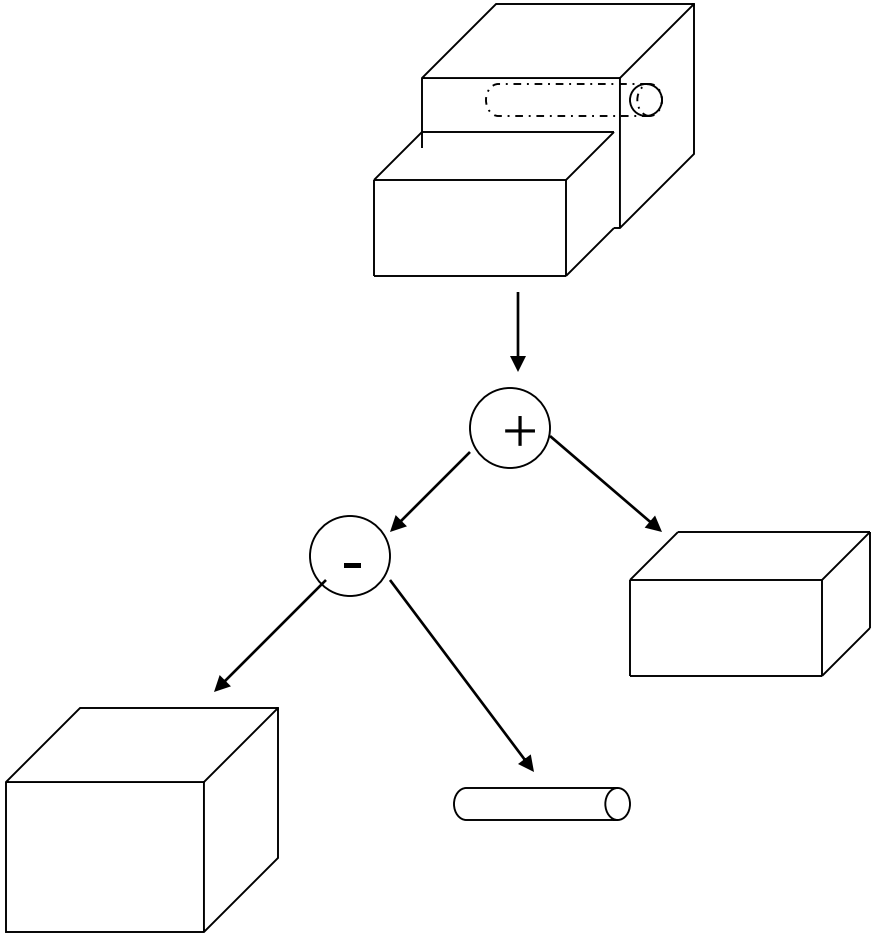


Constructive Solid Geometry

An object defined by a binary CSG tree



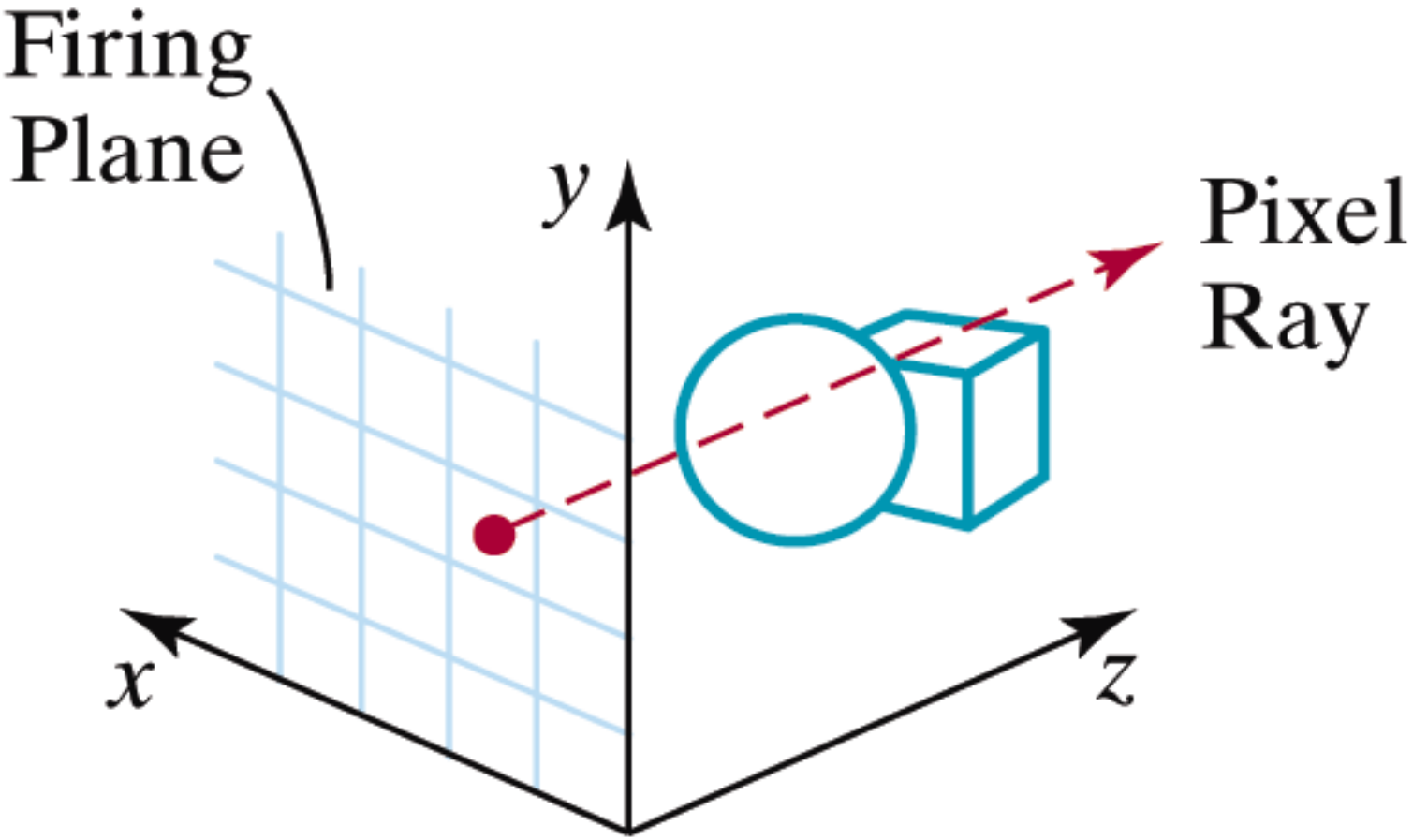
Constructive solid geometry (CSG)- CSG tree



Ray-Casting

- **Ray-casting** is typically used to implement CSG operators when objects are described with boundary representations
- Ray casting is applied by determining the objects that are intersected by a set of parallel lines emanating from the *xy* plane along the *z* axis
- The *xy* plane is referred to as the **firing plane**

Ray-Casting (cont…)

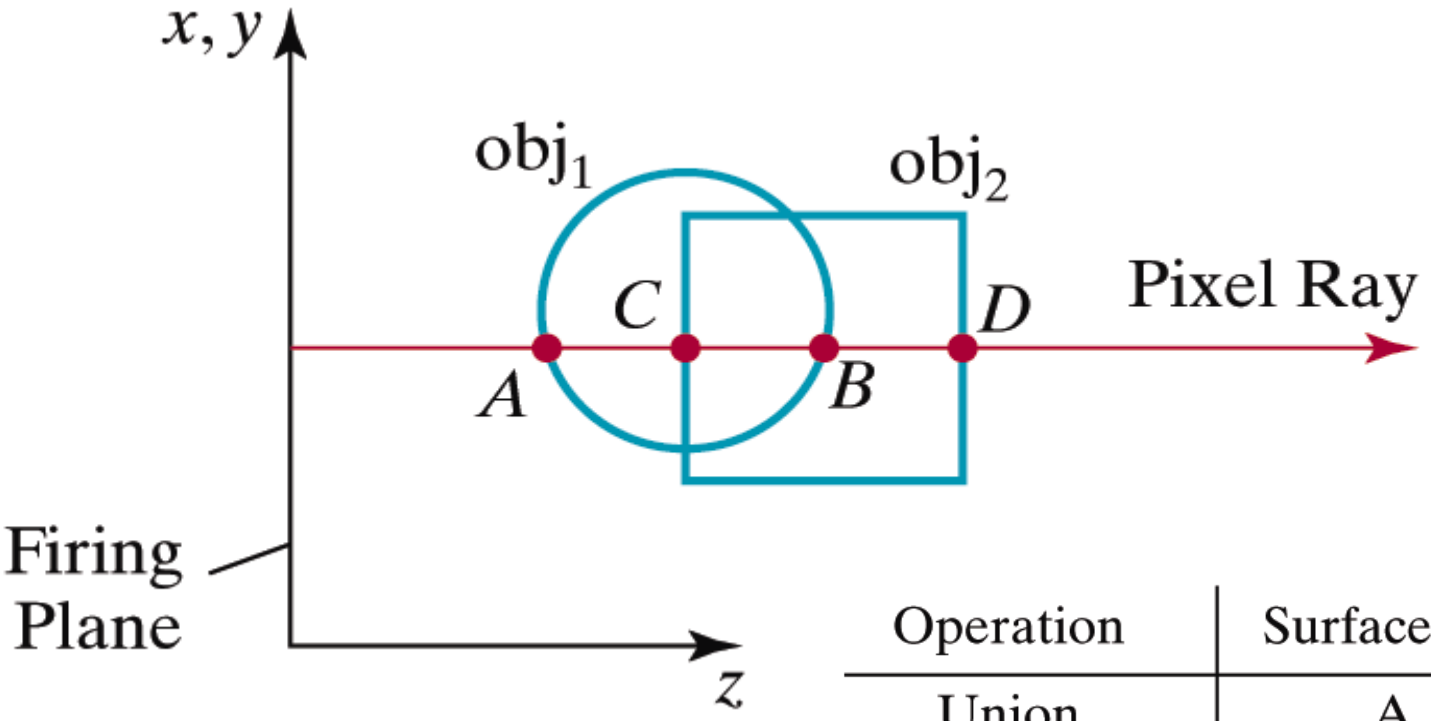


Images taken from Hearn & Baker, "Computer Graphics with OpenGL" (2004)

Ray-Casting (cont…)

- Surface intersections along each ray are calculated and these are sorted according to distance from the firing plane
- The surface limits for the composite object are then determined by the specified set operation

Ray Casting Example



Operation	Surface Limits
Union	A, D
Intersection	C, B
Difference ($obj_2 - obj_1$)	B, D

CSG Data Structure

- typedef struct _csgtree
  - Operator op;
  - Quadric primitive;
  - struct _csgtree *right;
  - struct _csgtree *left;
- } CSGTreePtr;
- Each leaf node represents a primitive (usually a quadric primitive).
- ‘op’ can also be ‘leaf’ rather than a combination operator.

Intersecting a Ray

- Pseudo code for ray-tree intersection

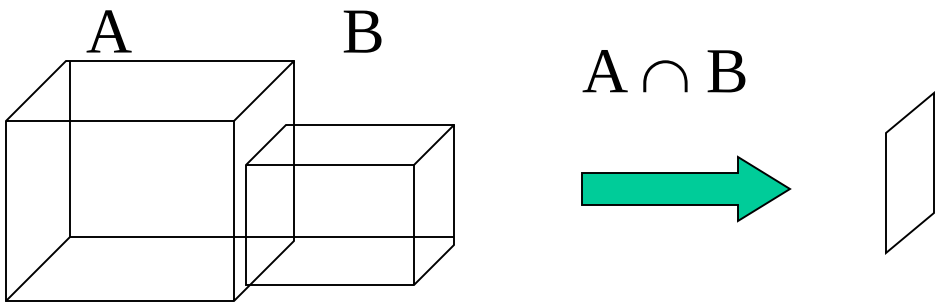
```
Classification RayCast(Ray ray, CSGTRee *solid)
{
        if(solid->op){
                left = RayCast(ray,solid->left);
                right = RayCast(ray,solid->right);
                Combine(solid->op,left,right);
        }
        else{
                //transform ray to local primitive coordinates
                switch(solid->primitive){
                        CASE cube:          .....
                        CASE sphere:        {intersection
                        CASE cone:          calculations}
                        CASE cylinder:      .....
                        //etc               .....
                }
        }
}
```

Intersecting a Ray

- The result will be a sequence of intersection points along a ray, represented by the parametric values
  - $t_0, t_1, t_2, \ldots, t_n$
  - where there are n intersections
- Each individual intersection with a solid will result (typically) in two values.

Constructive solid geometry (CSG)-Boolean operation

- When using Boolean operation, be careful to avoid situation that do not result in a invalid solid



Constructive Solid Geometry (CSG)- **Boolean operation**

- Boolean operation
  - Are intuitive to user
  - Are easy to use and understand
  - Provide for the rapid manipulation of large amounts of data.
- Because of this, many non-CSG systems also use Boolean operations
- Data structure does not define model shape explicitly but rather implies the geometric shape through a procedural description
  - E.g: object is not defined as a set of edges & faces but by the instruction : *union primitive1 with primitive 2*
- This procedural data is stored in a data structure referred to as a CSG tree
- The data structure is simple and stores compact data → easy to manage
- CSG tree → stores the history of applying boolean operations on the primitives.
  - Stores in a binary tree format
  - The outer leaf nodes of tree represent the primitives
  - The interior nodes represent the boolean operations performed.

Constructive Solid Geometry (CSG) – Advantage and Disadvantage

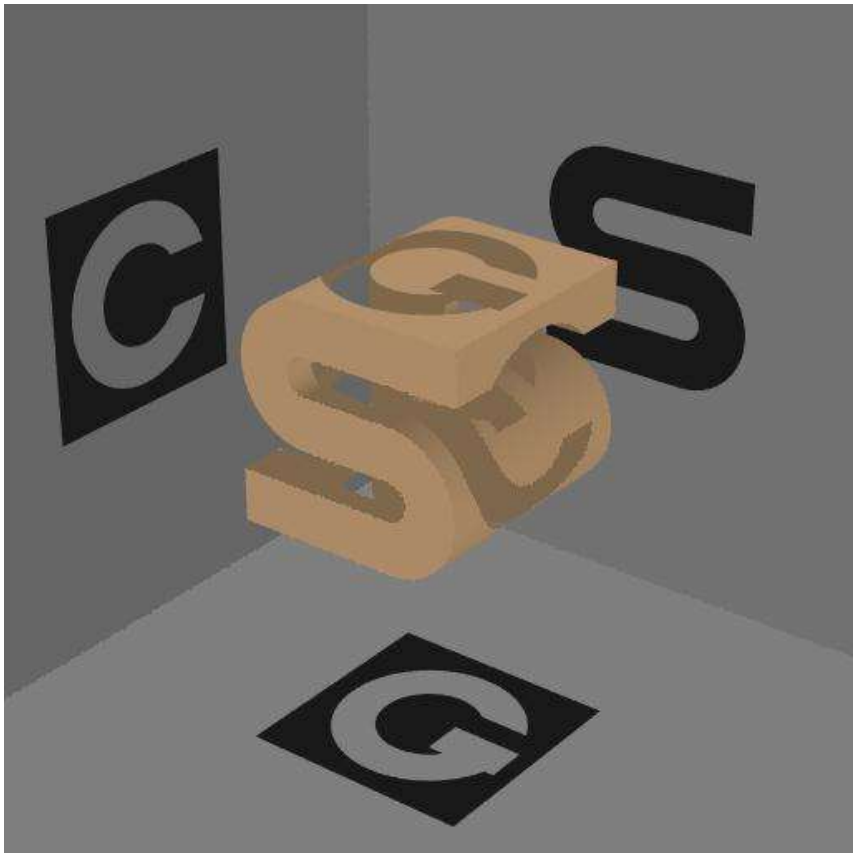
Advantage

- CSG is powerful with high level command.
- Easy to construct a solid model – minimum step.
- CSG modeling techniques lead to a concise database → less storage.
  - Complete history of model is retained and can be altered at any point.
- Can be converted to the corresponding boundary representation.

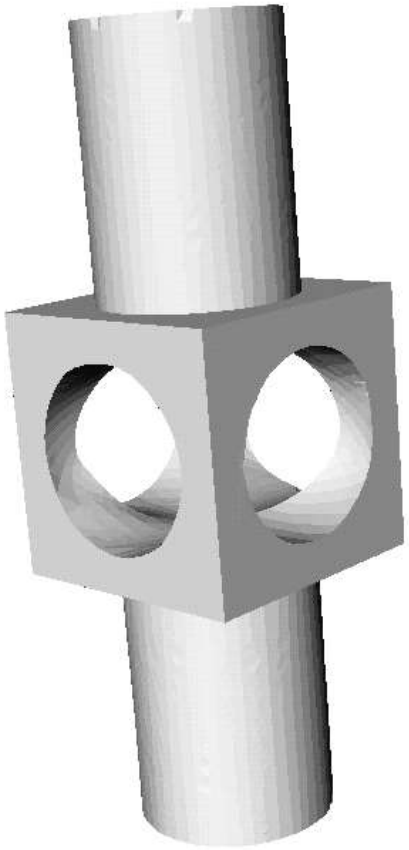
Disadvantage

- Only boolean operations are allowed in the modeling process → with boolean operation alone, the range of shapes to be modeled is severely restricted → **not possible to construct unusual shape.**
- Requires a great deal of computation to derive the information on the boundary, faces and edges which is important for the interactive display/ manipulation of solid.

Examples



www-2.cs.cmu.edu/afs/cs/misc/rayshade/all_mach/omega/doc/Examples/jpg/csg.jpg

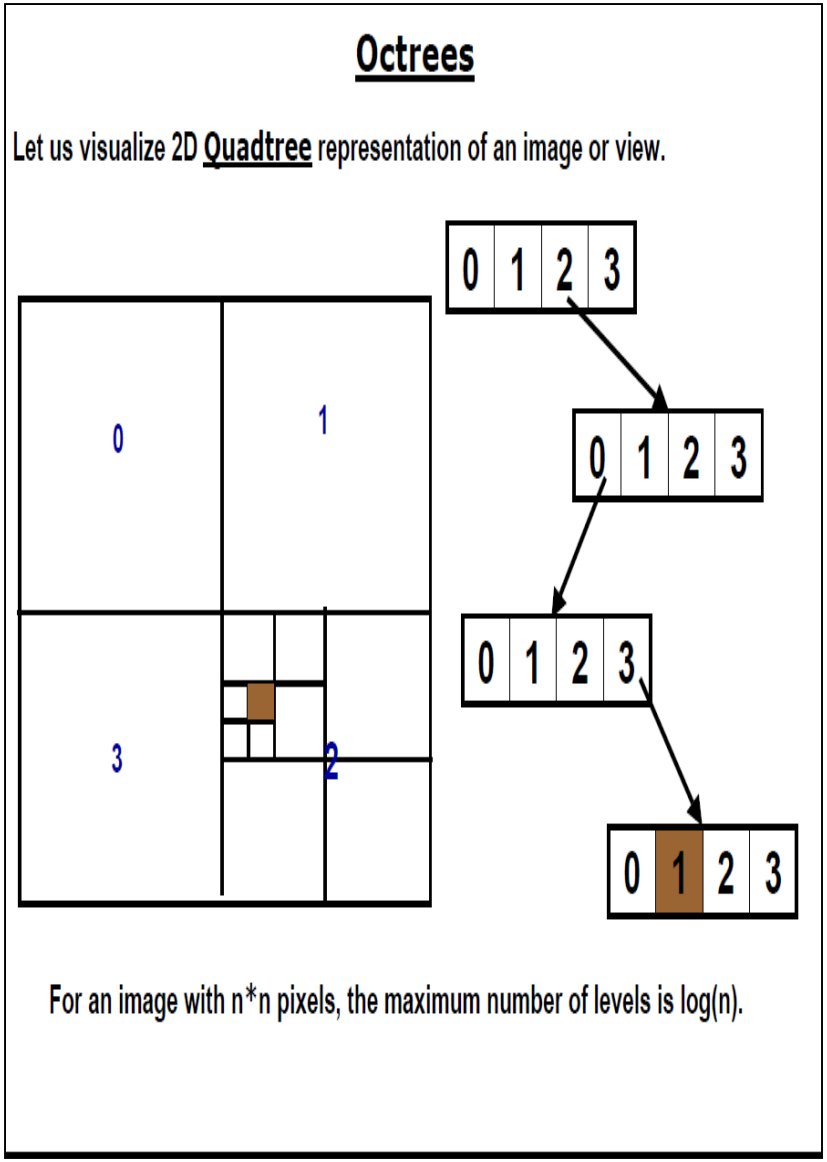


ccvweb.csres.utexas.edu/ccv/projects/VisualEyes/visualization/domainpara/algebraic2/parameterization/gallery/Reconstruction/Csg/

Octrees

- Hierarchical tree structures, called **Octrees,** are used to represent solid objects in some graphics systems.
- The **fundamental idea** behind both the **quadtree** and **octrees** is the divide and conquer power of binary

  sub division.
- **For a heterogeneous region of space**, the successive sub-division into quadrants continues until all quadrants are **homogenous color.**
- It also provides a convenient representation for storing information about object interiors.
- **Quadtree is used to speed up 3-D picking in graphics package**.

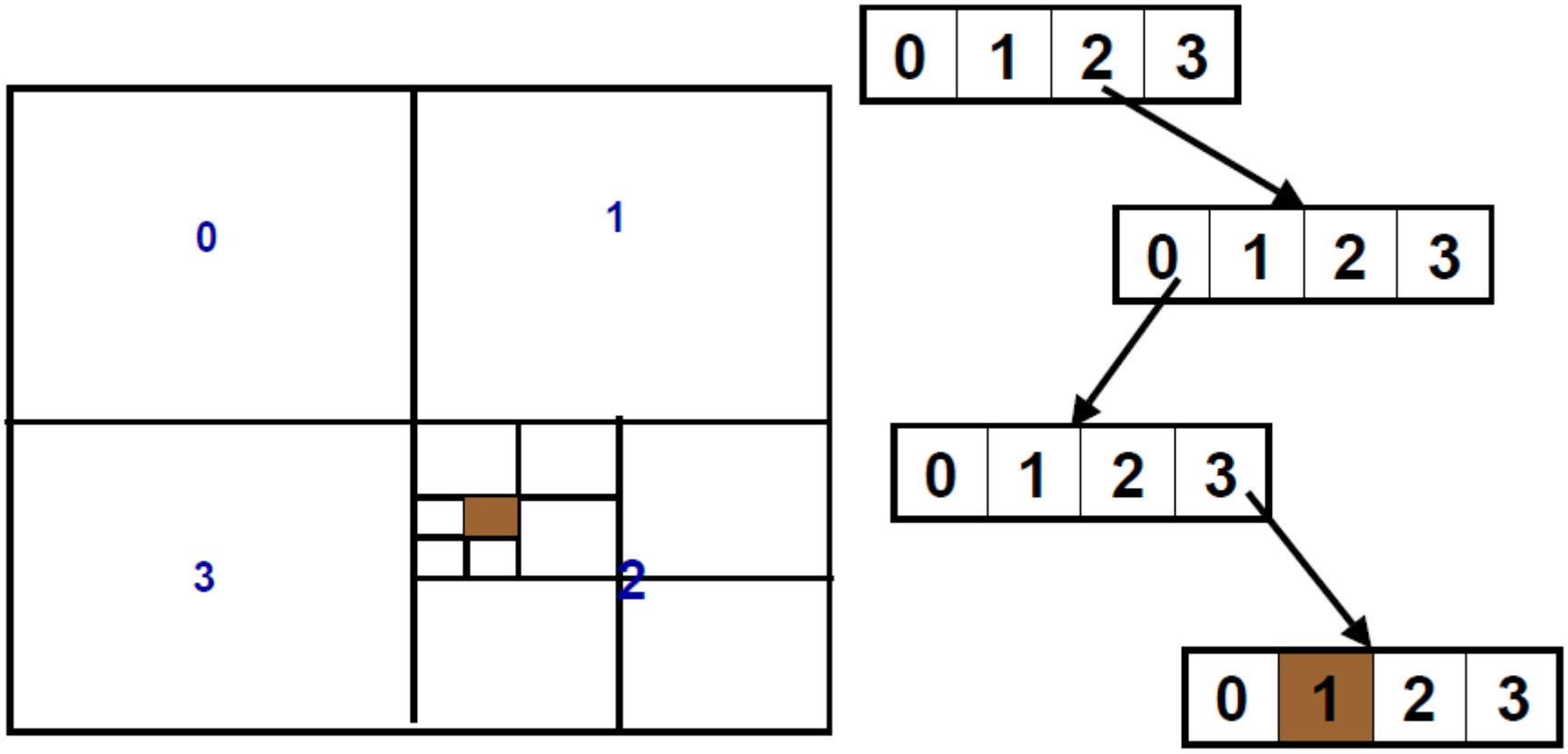


Octrees Advantages and applications

- Medical Imaging and other applications that require displays of object cross sections commonly use octrees representation
- **This representation for solids takes advantage of spatial coherence to reduce storage requirements for solids 3-D Objects.**
- 3-D octrees rotations are accomplished by applying the transformations to the occupied octants.
- **Visible Surface Detection (or) Hidden Surface elimination is accomplished by projecting octrees nodes on to the viewing surface in a front to back order.**
- Individual elements of a 3-D space are called **volume elements (**or) **voxels.**

Octrees

Let us visualize 2D **Quadtree** representation of an image or view.



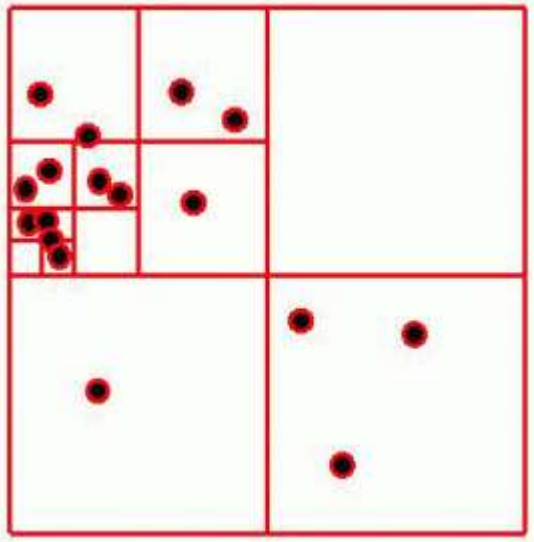
For an image with n*n pixels, the maximum number of levels is log(n).

2D-Quadtree

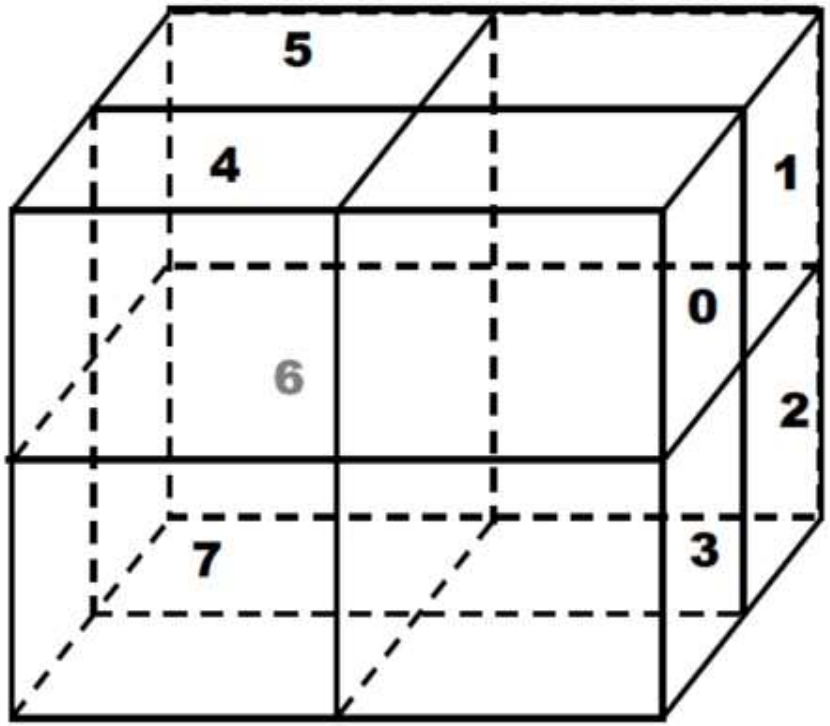
- A node either has 4 children or none
- Construct a quadtree with rule as no more than 3 dots can exist in a square.

2D-Quadtree

- A node either has 4 children or none
- The below quadtree has a rule that no more than 3 dots can exist in a square.

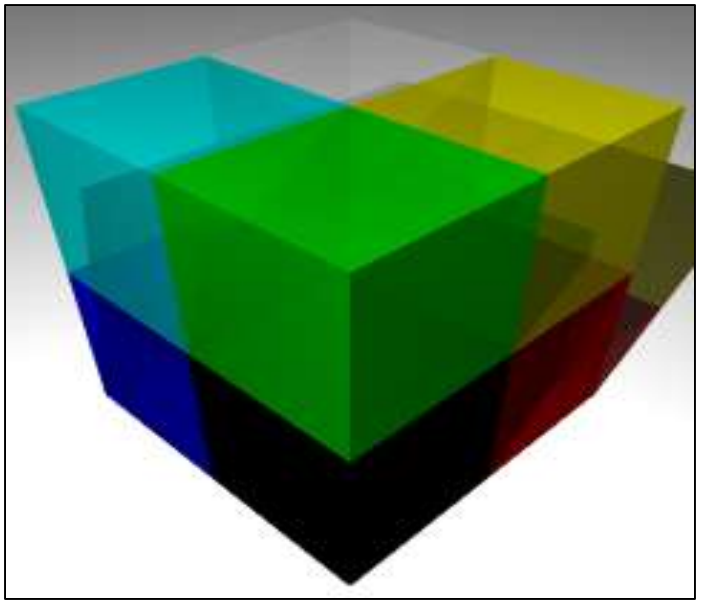


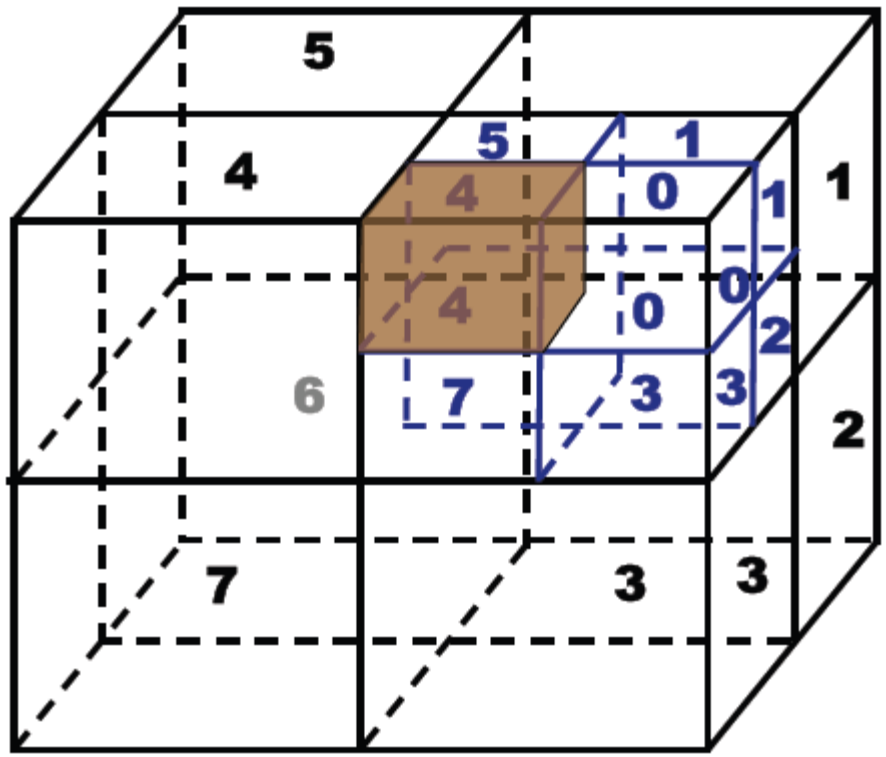
Octrees

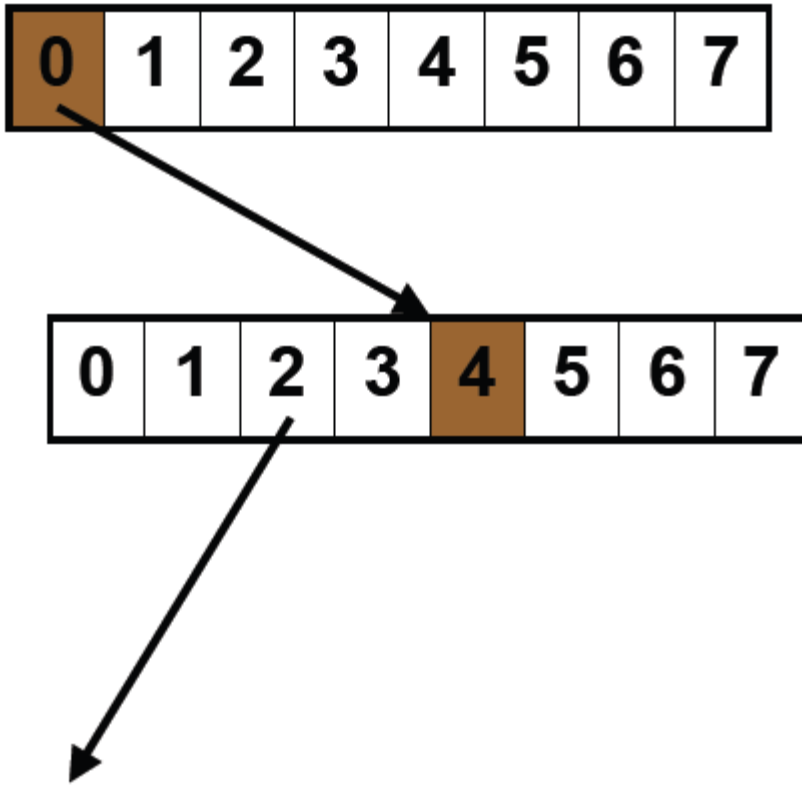


0	1	2	3	4	5	6	7

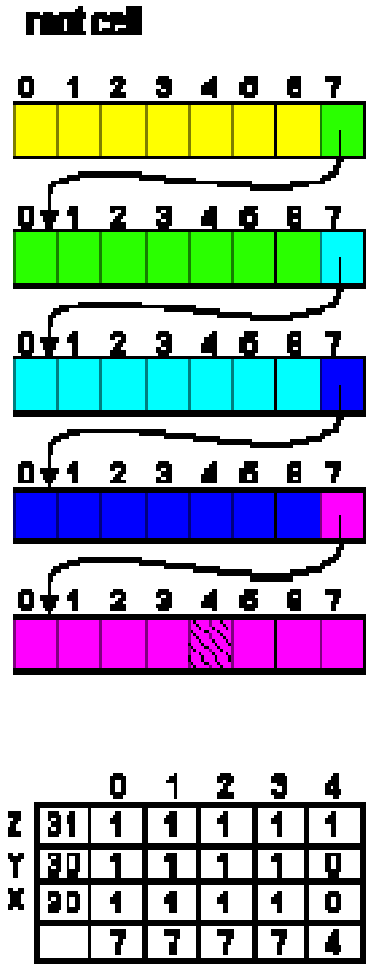
Individual elements are called Volume elements or VOXELS

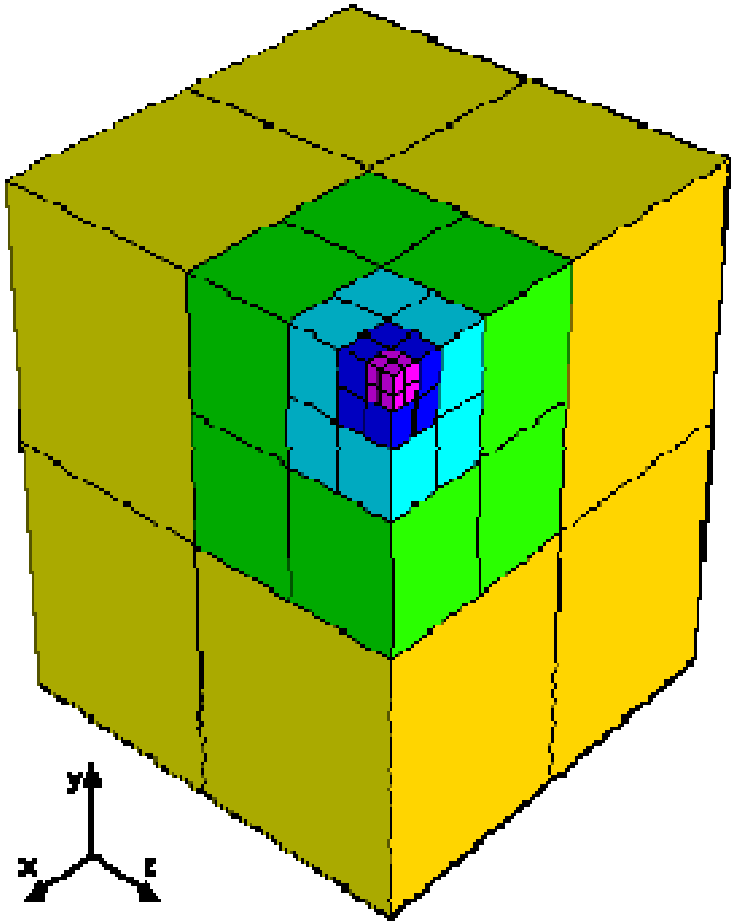



5
4
1
6
7
3
2
5
4
1
0
1
4
0
0
2
7
3
3

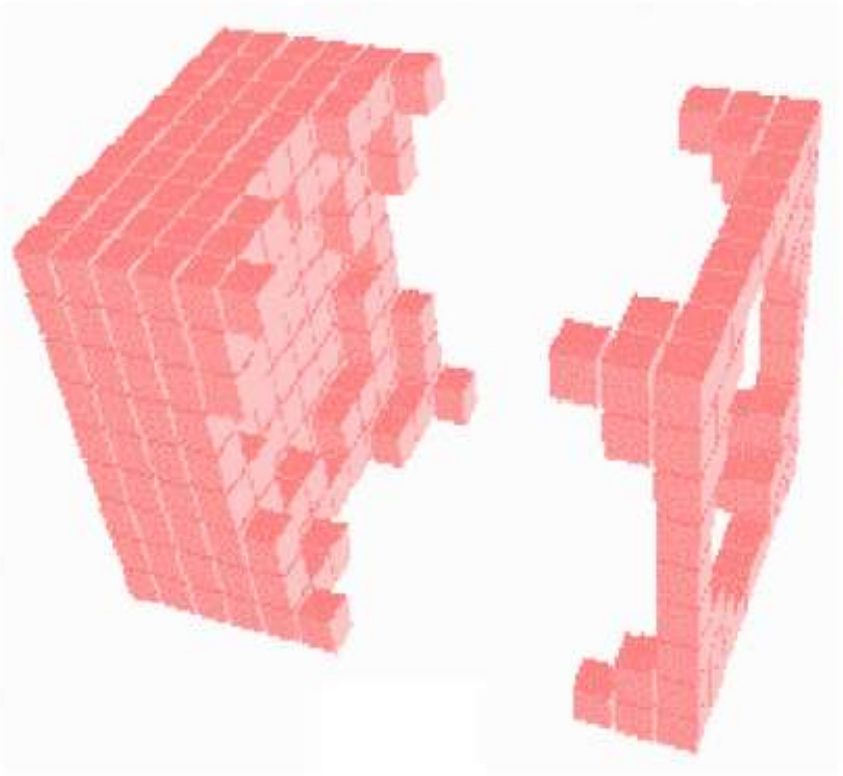

0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7

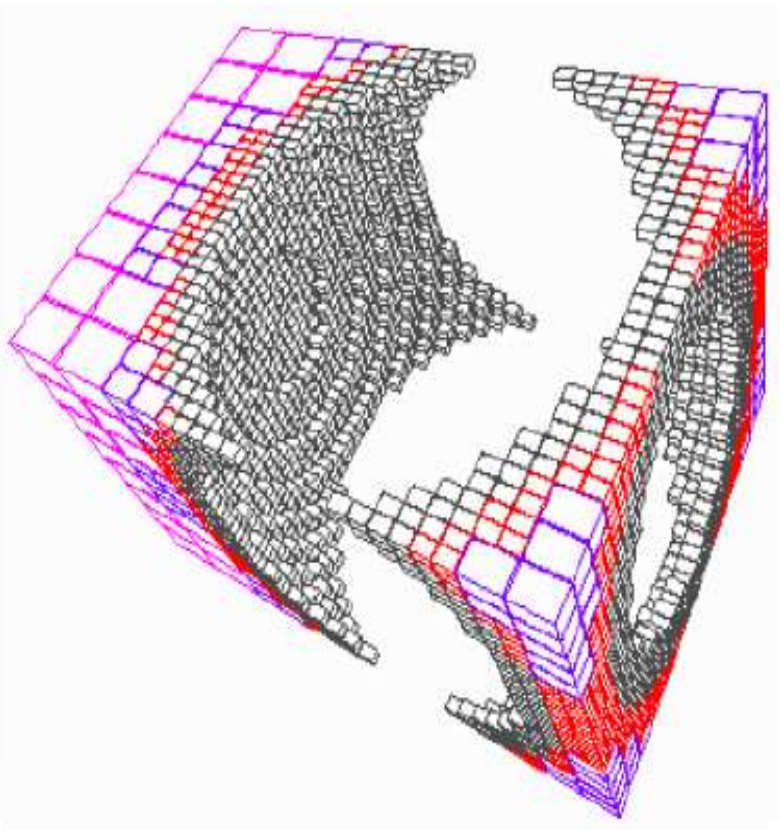
Octrees (Quadtrees): Axis aligned, regularly spaced planes cut space into cubes (squares)





Octrees





Parameter	Sweeps	B-reps	CSG	Octree
Accuracy	High	Sometimes Requires approx.	High	High
Domain	Limited	Can Represent wide class of objects	Limited to some extend	Limited
Uniqueness	Unique	Unique	Not Unique	Unique
Validity	Easy to Validate	Most difficult to validate	Easy to Validate	Easy to Validate
Closure	Not closed under Boolean operations	May suffer from closure problems under Boolean operations	Closed	Closed
Compactness	More Compact and efficient	Compact and evaluation is not necessary	More Compact needs evaluation of Boolean operations and transformation	Compact