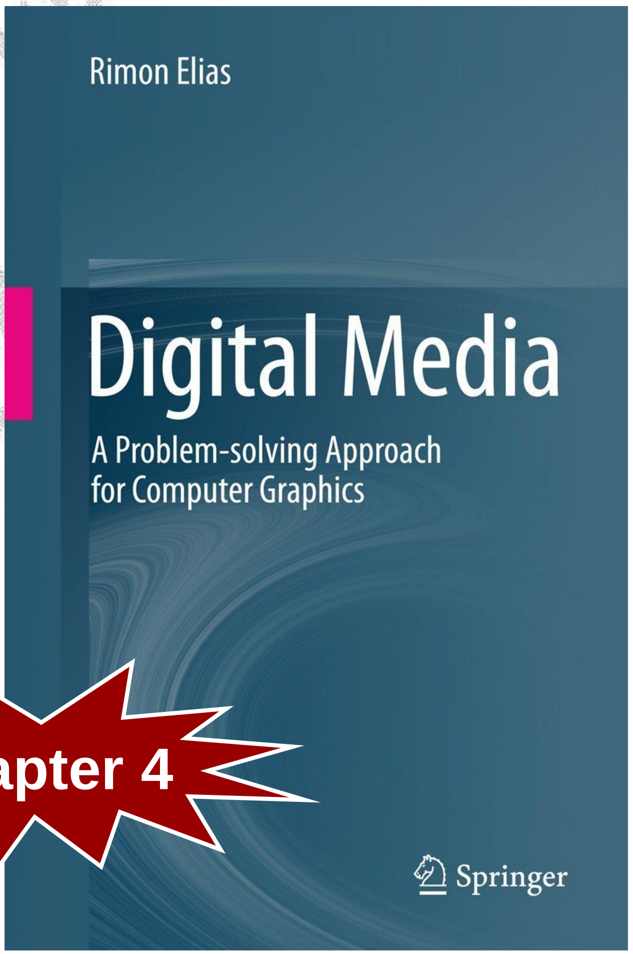
DMET 502/701 Computer Graphics

3D Solid Modeling

Assoc. Prof. Dr. Rimon Elias

Contents

- 3D modeling
  - Wireframes
  - Boundary representations (B-rep)
  - Constructive solid geometry (CSG)
  - Spatial Representations
    - Spatial Enumeration
    - Trees
  - Sweep Representations
    - Translational Sweeps
    - Rotational Sweeps
  - Polygonal Modeling


Rimon Elias

Digital Media

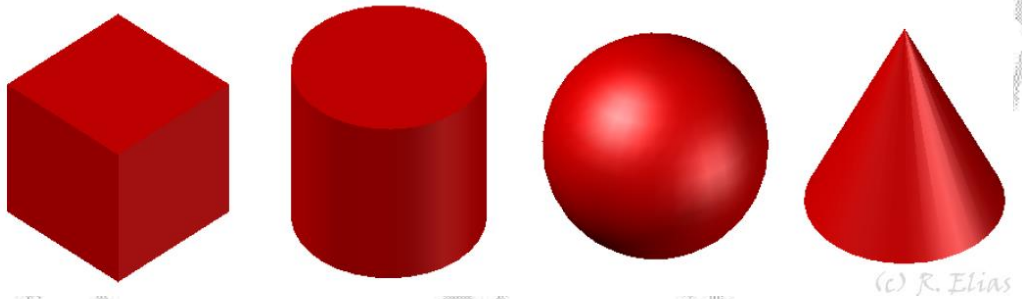
A Problem-solving Approach for Computer Graphics

Springer

Chapter 4

What is a Solid?

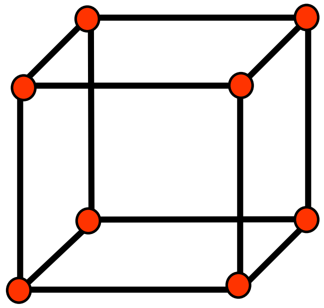
- A solid is a 3D model that is used to represent objects in 3D space.
- Examples of simple solids are cubes, cones, spheres, cylinders, etc.



- A 3D model may be represented in different ways. Among them:
  - Wireframes
  - Boundary representations (B-rep)
  - Constructive solid geometry (CSG)
  - …

Wireframe Models

- In a **wireframe** model, only **vertices** and **edges** are represented. No face information is available with this model.
- A wireframe model can be represented using two tables:
  - **Vertex table** where each entry is a vertex number and its coordinates.
  - **Edge table** where each entry is an edge number, start & end vertices.



Vertex #	x	y	z

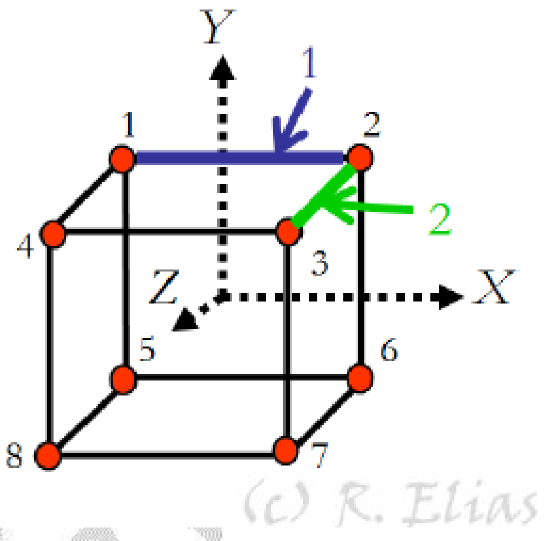
Vertex Table

Edge #	Start vertex	End vertex

Edge Table

Wireframe Models: An Example

- Suppose that the center of this cube is at $[0,0,0]^T$.



Vertex #	x	y	z
1	-1	1	-1
2	1	1	-1
3	1	1	1
4	-1	1	1
5	-1	-1	-1
6	1	-1	-1
7	1	-1	1
8	-1	-1	1

Vertex Table

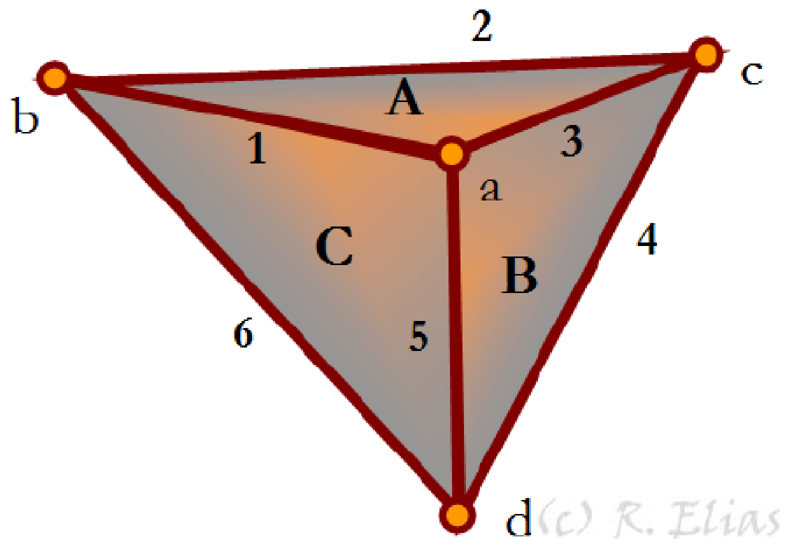
Edge #	Start vertex	End vertex
1	1	2
2	2	3
..	..	..
..	..	..

Edge Table

(Complete this table yourself)

Wireframe Models: An Example

- **Example:** Consider the tetrahedron shown where the vertices are indicated by lowercase letters (a, b, c and d), the faces by uppercase letters (A, B, C and D) and the edges by digits (1, 2, 3, 4, 5 and 6). The coordinates of the vertices are $[x_a, y_a, z_a]^T$, $[x_b, y_b, z_b]^T$, …

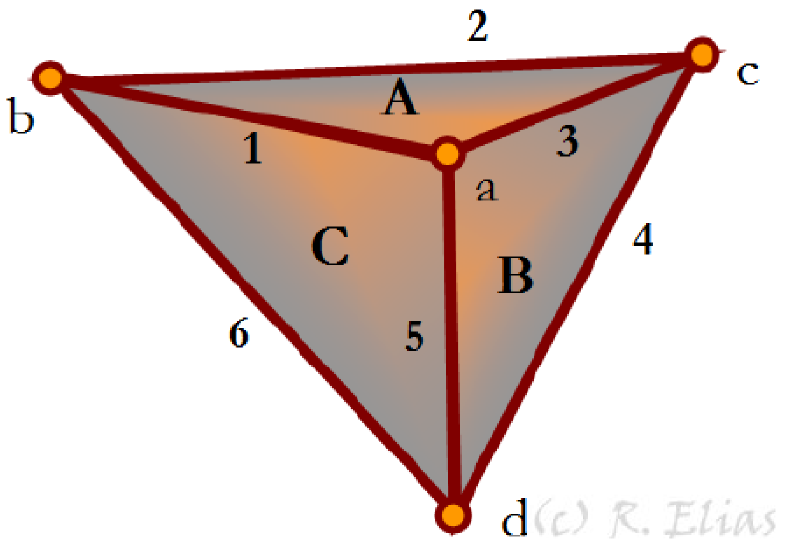


- Write down the entries of the edge table for this tetrahedron if it is to be represented as a **wireframe** model.

Wireframe Models: An Example

- **Solution:**

Edge #	Start vertex	End vertex
1	a	b
2	b	c
3	c	a
4	c	d
5	d	a
6	d	b

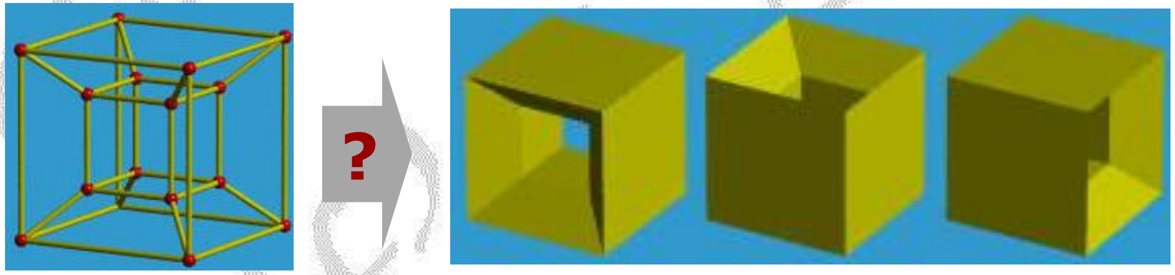


Wireframe Models: Pros and Cons

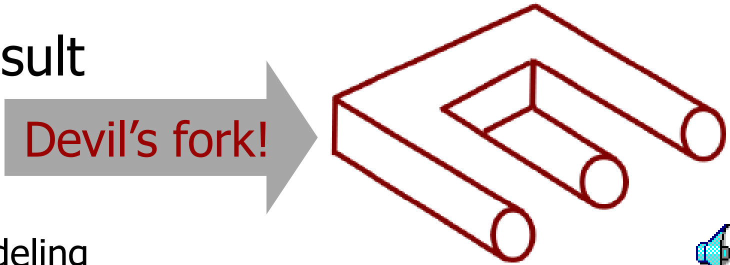
+
- Wireframe models are easy to construct, modify, clip and transform.

+
- Best choice for previewing models since less information processes faster.

-
- Representing a solid as a wireframe model could be ambiguous sometimes and could result in different interpretations.



-
- Sometimes, using wireframes might result in unrealistic models.



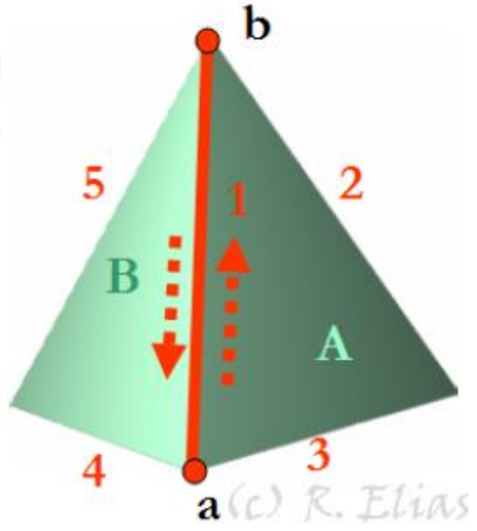
Boundary Representations

- **Boundary representations** or **B-reps** can be used to avoid the source of ambiguity arising in wireframe models.
- This is achieved by adding surface information to vertex and edge information that appears in wireframe modeling.
- Consequently, a solid is bounded by surface patches or faces forming exterior and interior for the solid.

B-rep: Baumgart's Winged-Edge Data Structure

- **Baumgart's *winged-edge* data structure** is a B-rep that uses edges to keep track of everything.

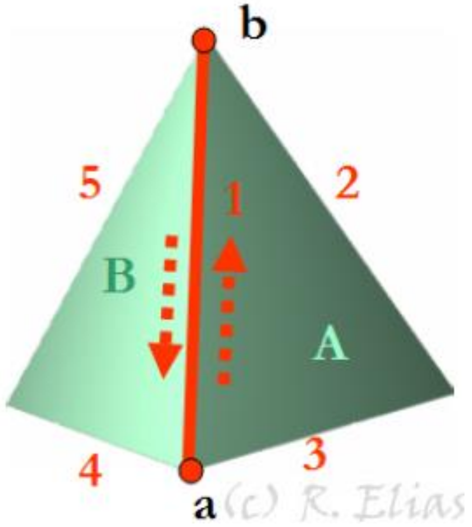
- Consider the pyramid shown:
  - Edge **1** connects vertices **a** and **b**.
  - Edge **1** is the intersection between the faces **A** and **B**.
- Edges are clockwise-ordered while they are viewed from outside.
  - The edges of face **A** are ordered as **1**, **2**, and **3**.
  - The edges of face **B** are ordered as **1**, **4** and **5**.



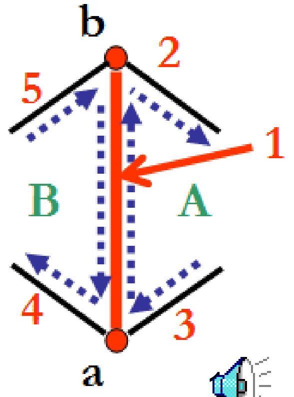
B-rep: Baumgart's Winged-Edge Data Structure

Traversal

- Faces, edges and vertices are traversed and the direction of traversal is important.
- Notice that
  - if you traverse edge **1** from vertex **a** to vertex **b**, face **A** will be on the right while face **B** will be on the left.
  - if you traverse edge **1** from vertex **b** to vertex **a**, face **A** will be on the left while face **B** will be on the right.

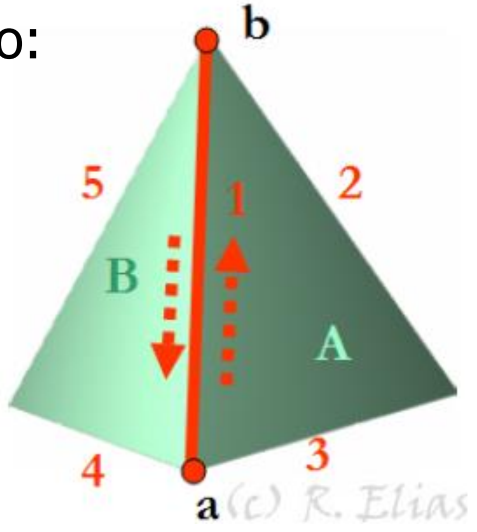


- Each edge is traversed once while traversing the face containing it. This means edge **1** is traversed twice; once while traversing **A** and another time while traversing **B**.
- Note that this is done in different directions ➔



B-rep: Baumgart's Winged-Edge Data Structure

- For each edge (e.g., **1**), we need the following info:
  - Vertices of this edge (**a** and **b**).
  - Its left and right faces (**B** and **A**).
  - The predecessor and successor of this edge when traversing its left face (**5** and **4**).
  - The predecessor and successor of this edge when traversing its right face (**3** and **2**).



Baumgart's Winged-Edge Data Structure

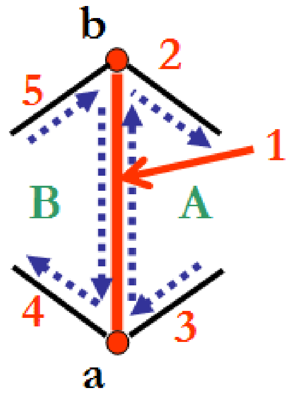
Edge	Vertices		Faces		Left traverse		Right traverse	
Name	Start	End	Left	Right	Pred	Succ	Pred	Succ
1	**a**	**b**	**B**	**A**	**5**	**4**	**3**	**2**

Edge Table

B-rep: Baumgart's Winged-Edge Data Structure

- Because the four edges **2**, **3**, **4** and **5** look like wings of edge **1**; thus, edge **1** is said to be **winged**.

Winged!



More tables

- Two more tables:
  - **The vertex table** contains one entry for each vertex that represents an edge passing through it.
  - **The face table** contains one entry for each face that represents one of its edges.
- Note that because there are different choices in each case, tables for the same solid may vary.

Vertex table

Vertex	edge
a	1
b	5
..	..

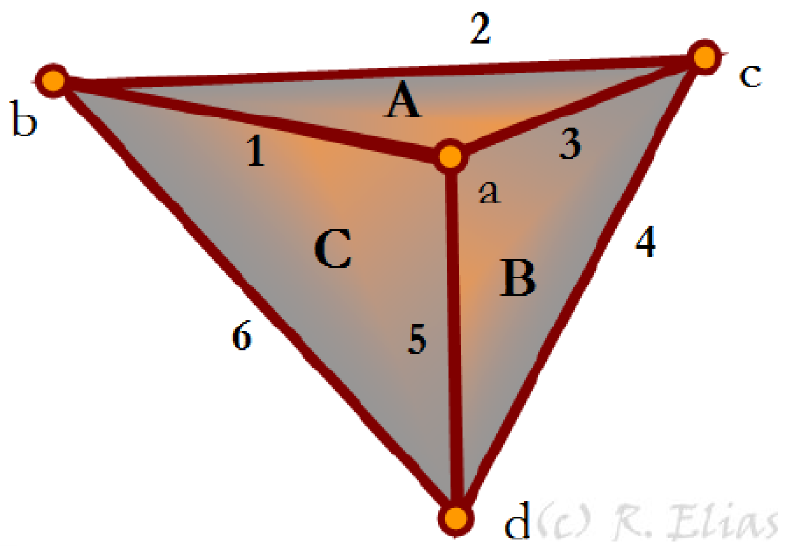
Face table

Face	edge
A	2
B	5
..	..

Note that by using the three tables, many queries about adjacency may be answered in constant time.

Winged-Edge Data Structure: An Example

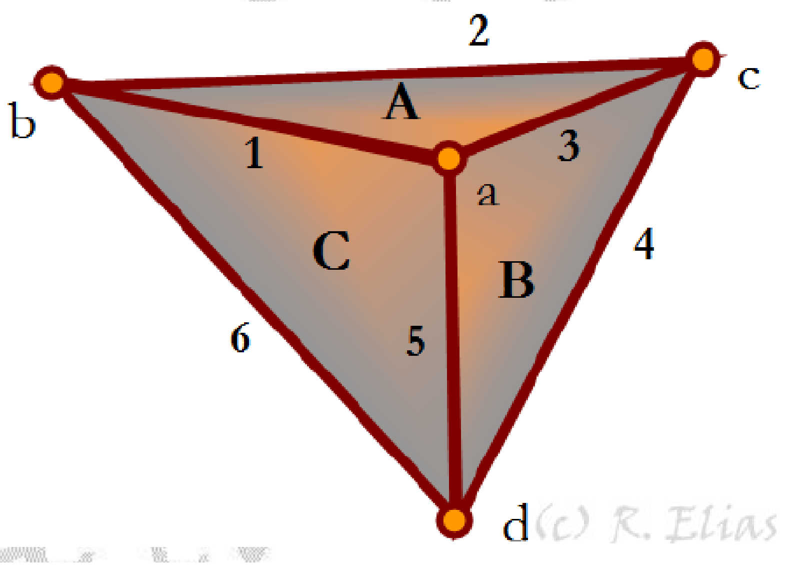
- **Example:** Consider the tetrahedron shown where the vertices are indicated by lowercase letters (a, b, c and d), the faces by uppercase letters (A, B, C and D) and the edges by digits (1, 2, 3, 4, 5 and 6).



- Write down the entries of the edge table of **Baumgart's Winged-Edge Data Structure** for this model.

Winged-Edge Data Structure: An Example

- **Solution:**



Edge	Vertices		Faces		Left traverse		Right traverse	
#	Start	End	Left	Right	Pred.	Succ.	Pred.	Succ.
1	a	b	C	A	6	5	3	2
2	b	c	D	A	4	6	1	3
3	c	a	B	A	5	4	2	1
4	c	d	D	B	6	2	3	5
5	a	d	B	C	4	3	1	6
6	d	b	D	C	2	4	5	1

Constructive Solid Geometry

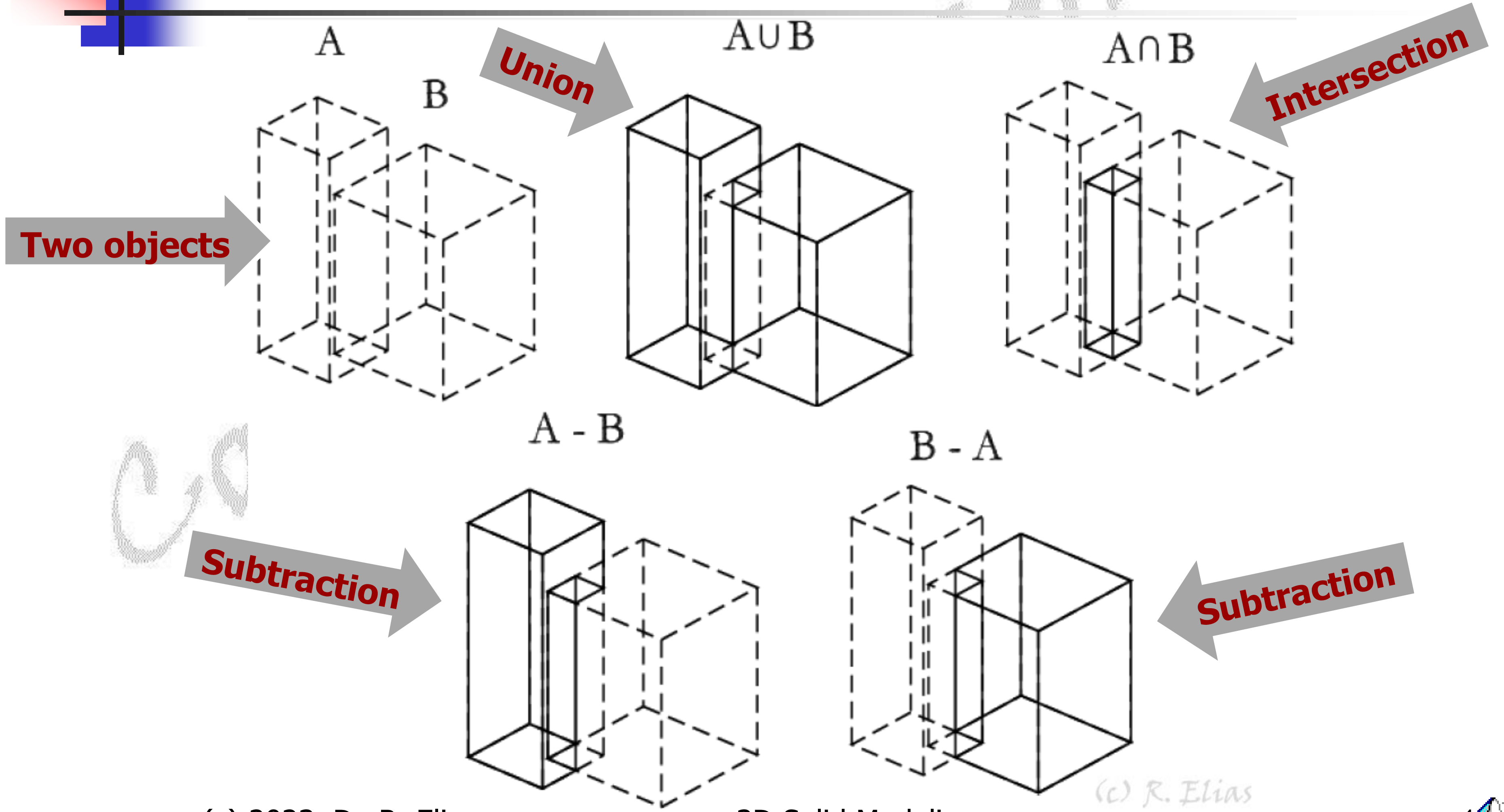
- **Constructive solid geometry (CSG)** is a technique used in solid modeling.

- The simplest solid objects used for the representation are called **primitives**. Typically they are the objects of simple shape: cubes, cylinders, spheres, cones, etc.

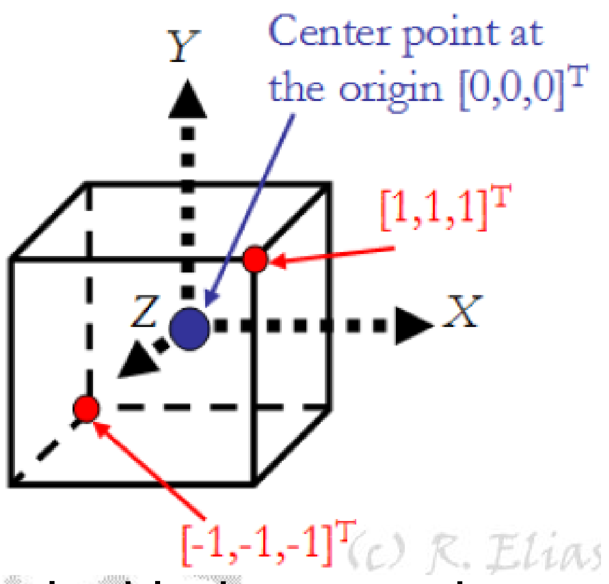
- A primitive may be created by a procedure that accepts its parameters.
- A primitive may be transformed (e.g., rotated, translated, etc.)

- Complex objects are created using Boolean operators to combine primitives.

Constructive Solid Geometry: Boolean Operations



CSG Expressions



- Consider the "primitive" cube shown. It extends from $[-1,-1,-1]^T$ to $[1,1,1]^T$ (i.e., height=width=length=2).
- This cube is to be transformed into a rectangular block centered at $[1,2,3]^T$ and whose side lengths are 2, 3, 3 units along the *x*-, *y*- and *z*-directions.
- To achieve this:
  - Scale the primitive 1.5 times along the *y*- and *z*-directions. (The scale along the *x*-direction should remain 1.)
  - Translate using a translation vector $[1,2,3]^T$.
  - This can be written as:

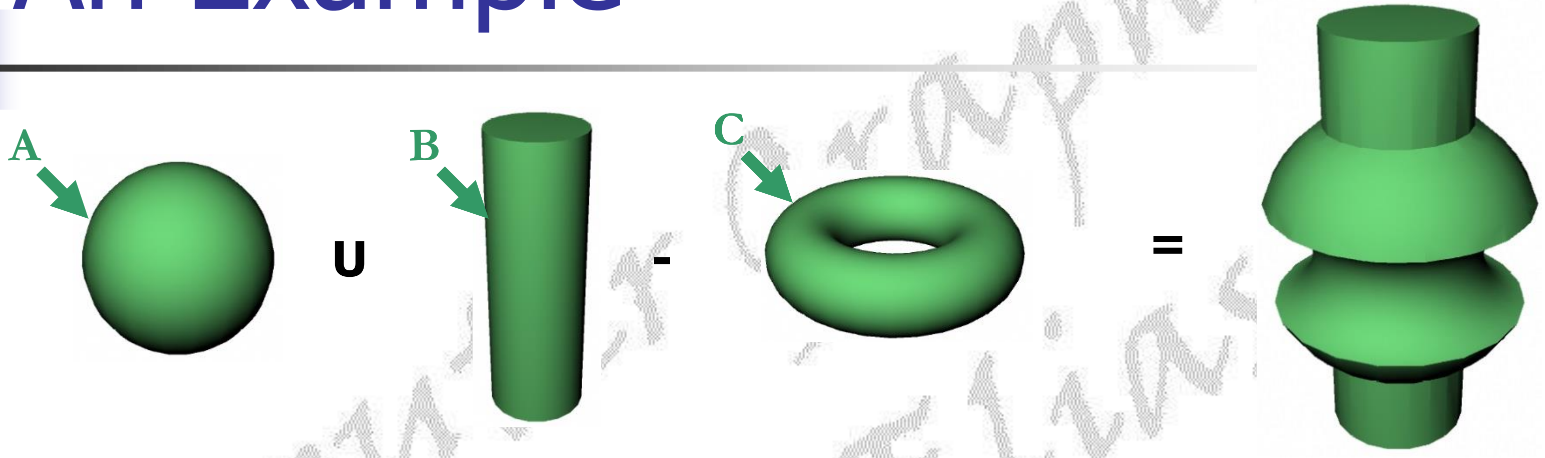
CSG Expression **translate(scale(Block, < 1, 1.5, 1.5 >), < 1, 2, 3 >)**

CSG Expressions

$$\underbrace{\text{Translate}\ (\underbrace{\overbrace{\text{scale}\ (\overbrace{\text{Block}}^{\text{Argument 1}},\ \overbrace{<1,\ 1.5,\ 1.5>}^{\text{Argument 2}})}^{\text{Scaling}}}_{\text{Argument 1}},\ \underbrace{<1,\ 2,\ 3>}_{\text{Argument 2}})}_{\text{Translation}}$$

- This means that:
  - The primitive is called "Block."
  - "Block" is scaled using scaling factors of 1, 1.5 and 1.5 along the *x*-, *y*- and *z*-directions respectively.
  - After scaling, this solid gets translated using a translation vector $[1, 2, 3]^T$.

CSG Expressions: An Example



- Suppose that you have three primitives; a sphere (A), a cylinder (B) and a torus (C).
- It is required to have the result shown on the right.
- This is achieved by unionizing A and B and subtracting C from them.
- This can be written as a CGS expression.

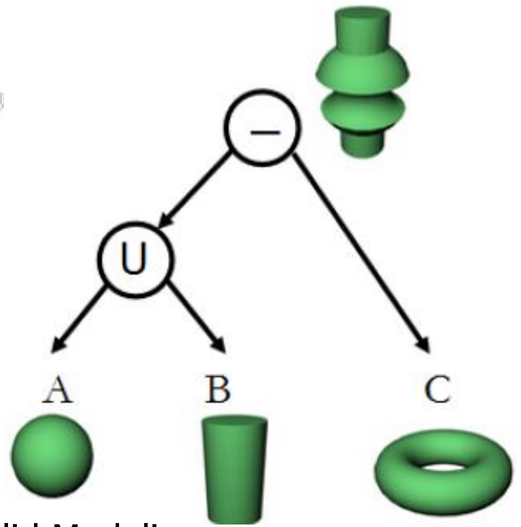
diff(union(A, B), C) $\equiv$ **A U B \ C**

CSG: Expression & Tree

- Every solid constructed using the CSG technique has a corresponding CSG expression which in turn has an associated **CSG tree**.
- The CSG tree is a representation of the final design.
- An object is stored as a tree data structure with operators at the internal or branch nodes and simple primitives at the leaves.

CSG Expression

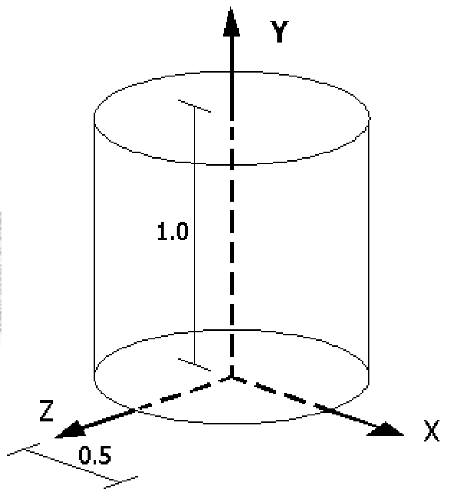
diff(union(A, B), C)

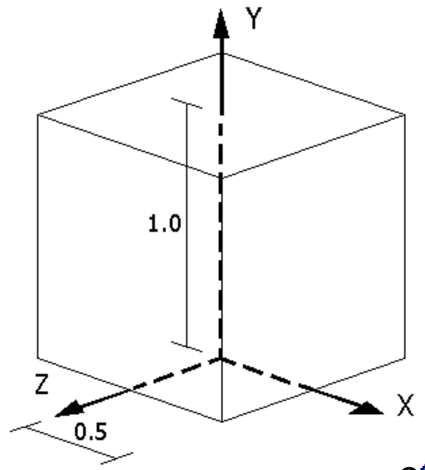


CSG Tree

CSG: An Example

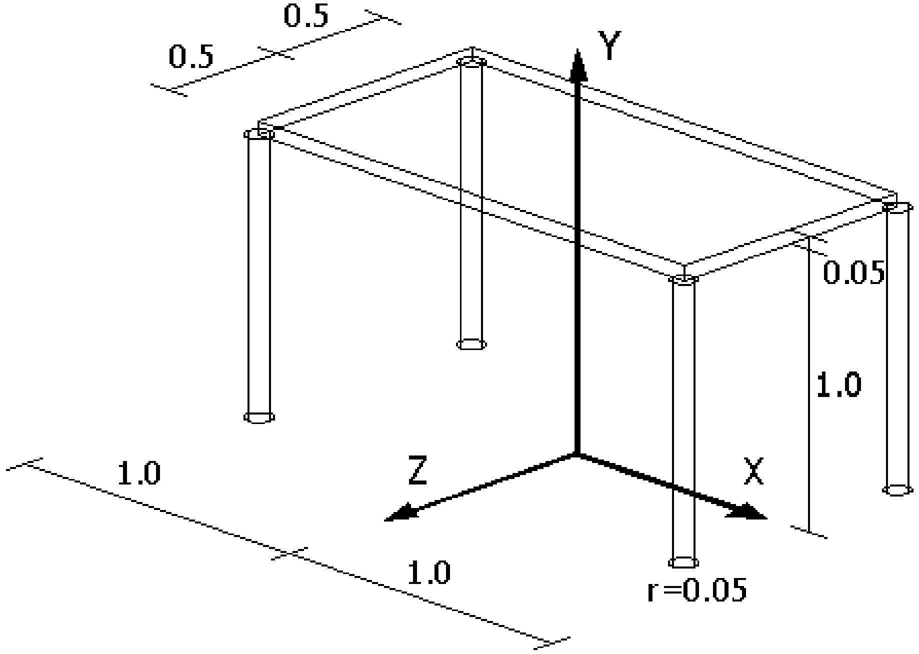
- **Example:** Suppose that you are asked to build a 3D modeling software.
- This software can create two types of solid primitives; cylinders and cubes.
- Every primitive instance is initially created at the origin of the 3D coordinate space (as shown) with lengths equal to the unit.
- You will use these primitives to build a 3D model for a table.





CSG:
An Example (cont.)

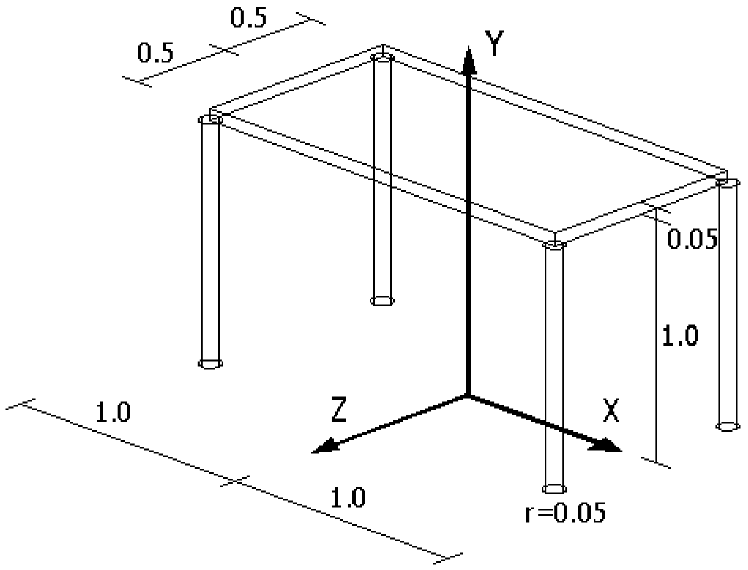
- Create the CSG expression required to represent this solid.
- Build the CSG tree representing the previous expression.



CSG: An Example (cont.)

- **Solution:** Create the top of the table as a box and transform it.
- Create the 4 legs as cylinders and transform them.
- Unionize all the five parts (top+4 legs).

- **CSG expression**
  - The CSG expression can be written as:

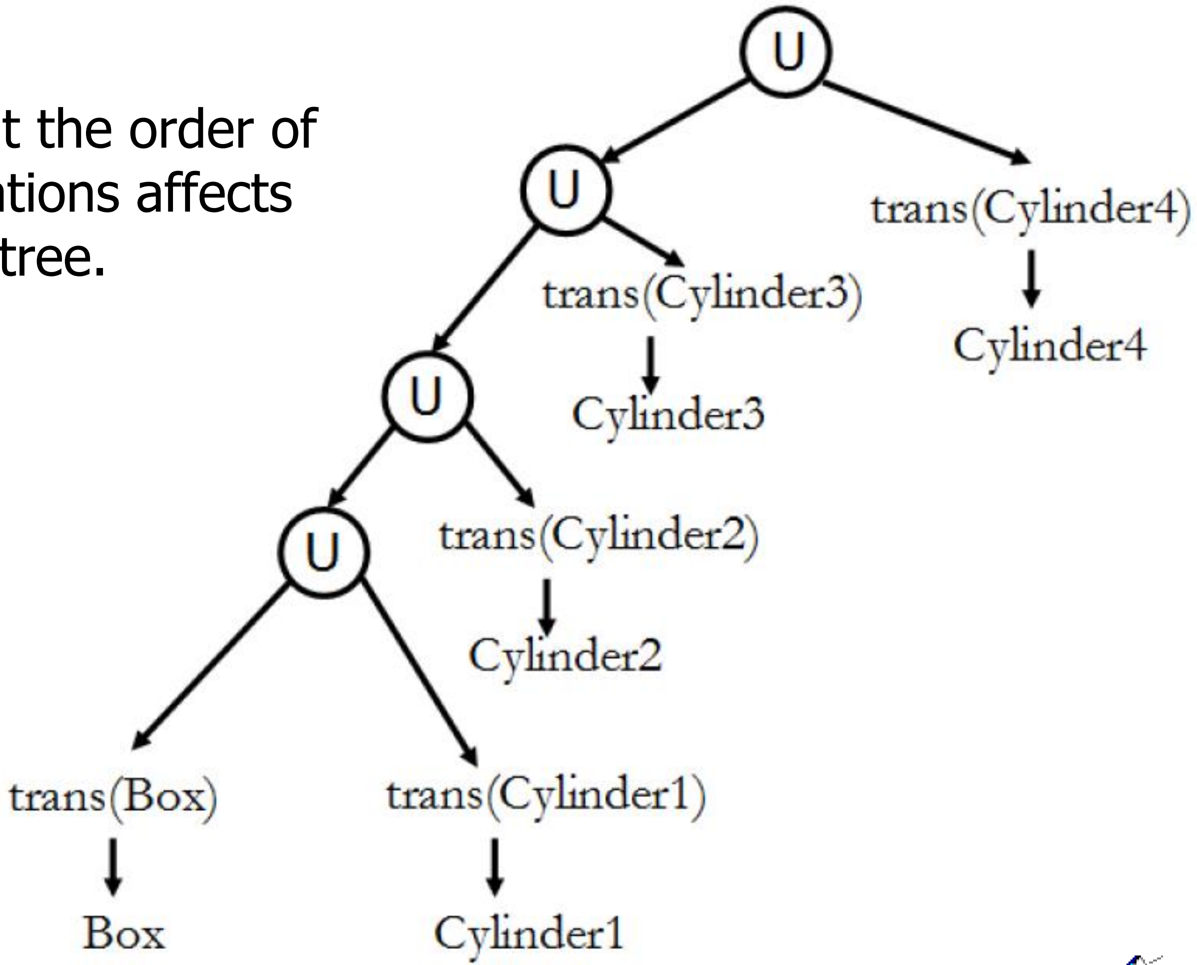


((((trans(box) U trans(cylinder1)) U trans(cylinder2)) U trans(cylinder3)) U trans(cylinder4))

CSG: An Example (cont.)

- **CSG Tree**
  - Pay attention that the order of performing operations affects the nodes in the tree.

In some implementations, intersection and union nodes may have any number of children. If this is the case in our example, the tree will contain one union node that has five children representing the transformed primitives.



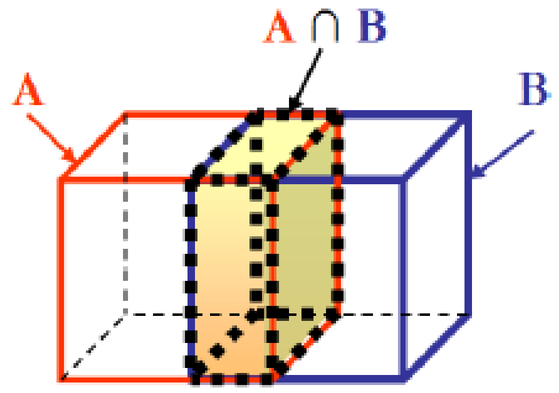
CSG: An Example (cont.)

3D Transformations

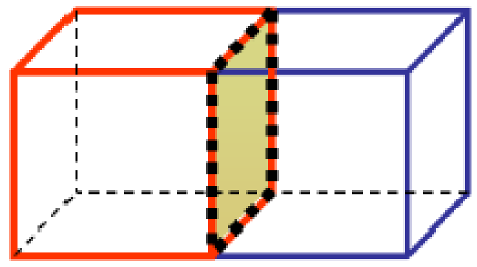
- Trans(box) =
  - **translate(scale(box, <2, 0.05, 1>), <0, 1, 0>)**
- Trans(cylinder1) =
  - **translate(scale(cylinder1, <0.1, 1, 0.1>), <1, 0,0.5>)**
- Trans(cylinder2) =
  - **translate(scale(cylinder2, <0.1, 1, 0.1>), <1, 0,-0.5>)**
- Trans(cylinder3) =
  - **translate(scale(cylinder3, <0.1, 1, 0.1>), <-1, 0,0.5>)**
- Trans(cylinder4) =
  - **translate(scale(cylinder4, <0.1, 1, 0.1>), <-1, 0,-0.5>)**

Regularized Boolean Operators

- Usually the Boolean operations applied to solids result in solids as well.
- Consider these two cubes. The intersection between them is a solid.

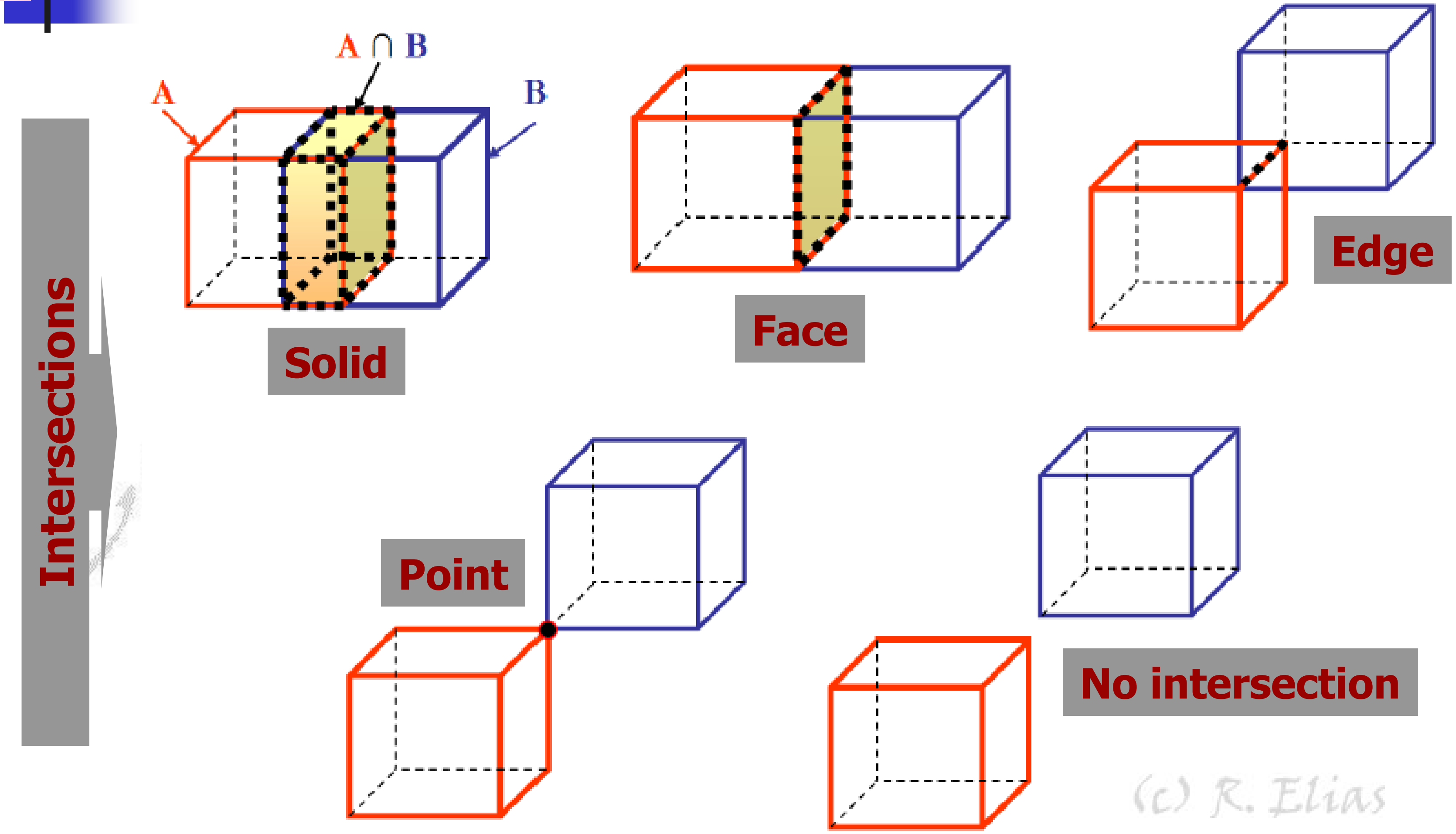


- Consider this case.
  - The intersection between the cubes is a rectangular face.
  - The rectangle is not a 3D object (i.e., not a solid).



➔ **We need to regularize the three set operations.**

Regularized Boolean Operators



Regularized Boolean Operators: Interior, Exterior and Closure

- We use the following concepts:
  - The **interior** of a solid contains all points inside the solid.
  - The **closure** of a solid contains the interior points plus the points on the surface of the solid.
  - The **exterior** of a solid contains points that do not belong to the closure.

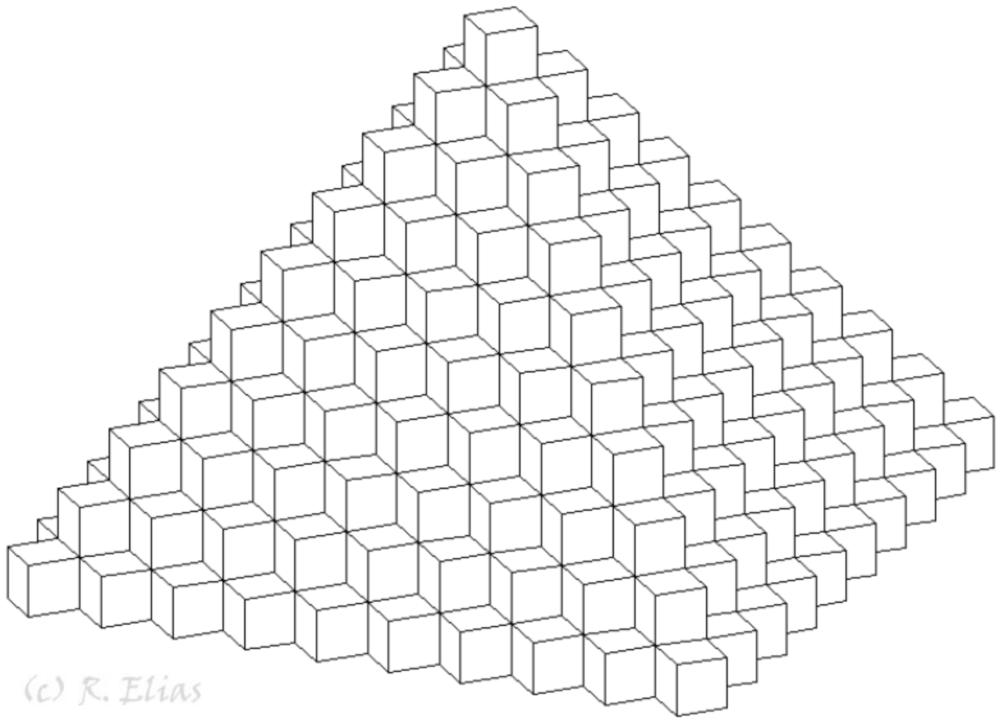
Regularized Boolean Operators

- Regularized operators are computed as follows:
  1. Compute the result of the Boolean operation as usual.
  2. Compute the interior of the result.
  3. Compute the closure of the result. This adds the boundaries back.

- How does this affect the result of a rectangle?
  1. The result of the first step is a rectangle.
  2. The interior of a rectangle is empty.
  3. The closure of an empty set is empty as well.

Spatial Enumeration

- In **spatial enumeration**, the space is split into equal-sized small *volume elements* or ***voxels***.
- A voxel may be vacant or occupied.
- Any 3D model can be represented as a list of occupied voxels.



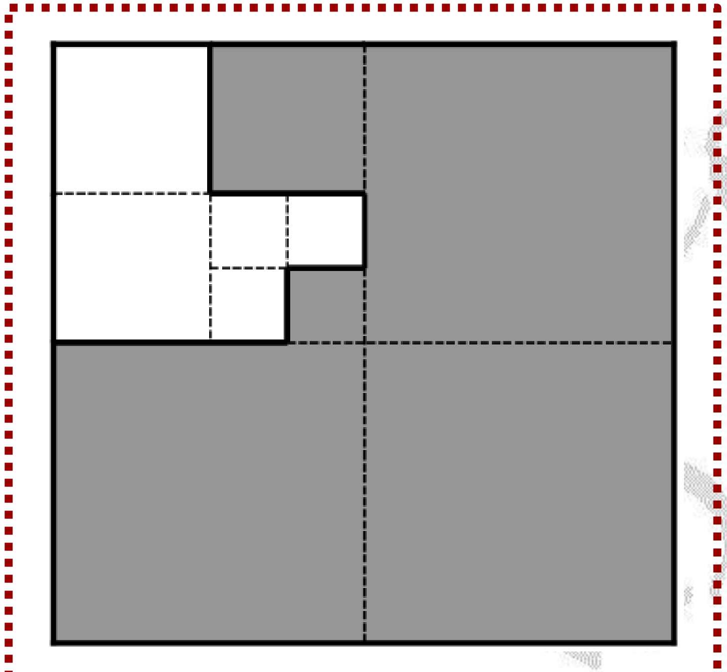
Quadtrees

- A **quadtree** is generated by dividing a 2D plane along two directions into **four quadrants**.

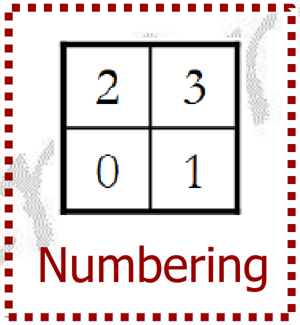
- Each quadrant could be full, partially full or empty.

- In case of partially full quadrants, subdivision continues recursively until quadrants are either full or empty (or until a specific tree depth or a minimum area is reached).

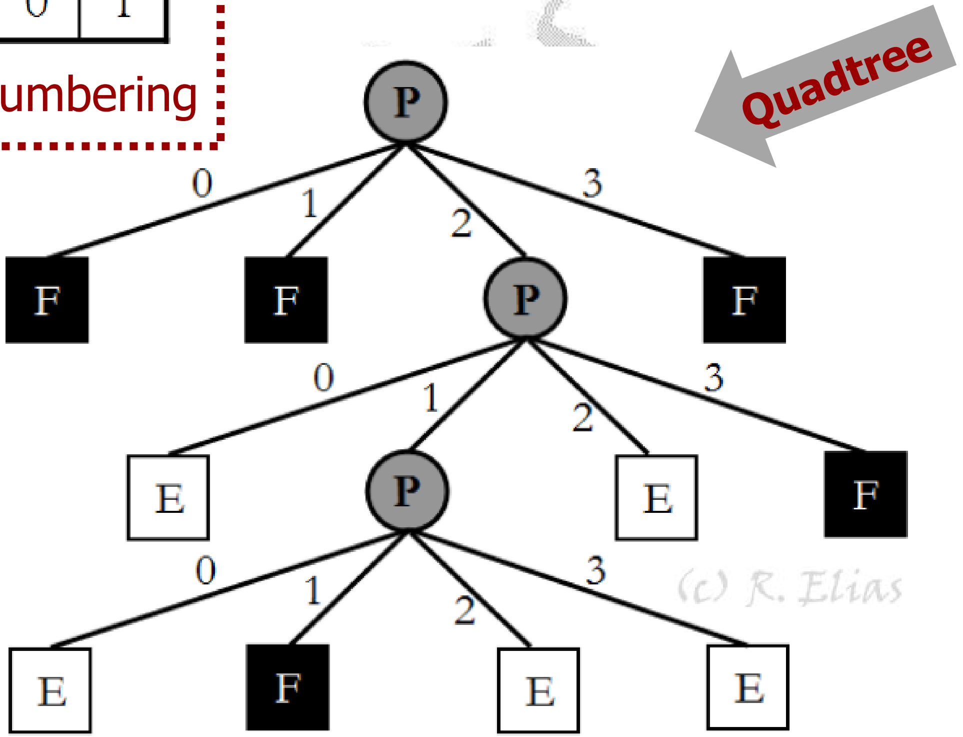
Quadtrees: An Example



A 2D pattern

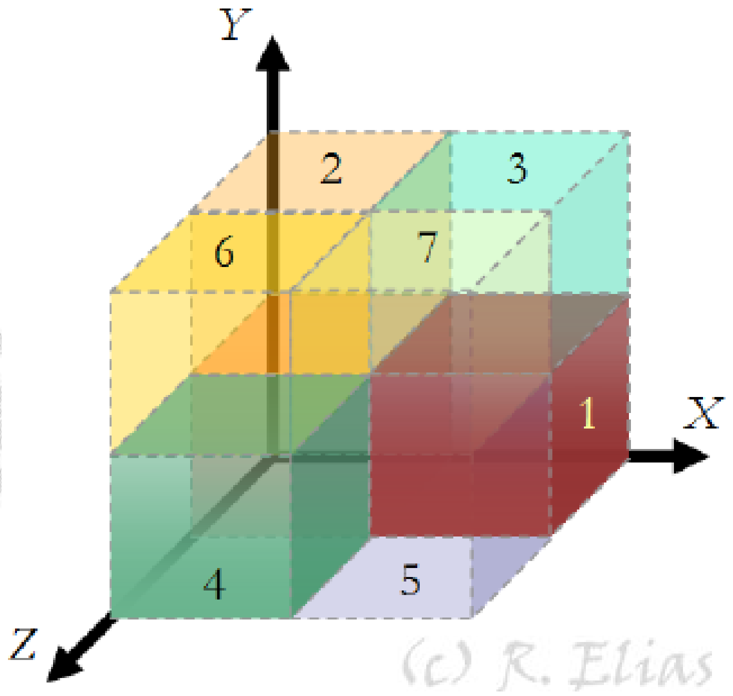


Numbering



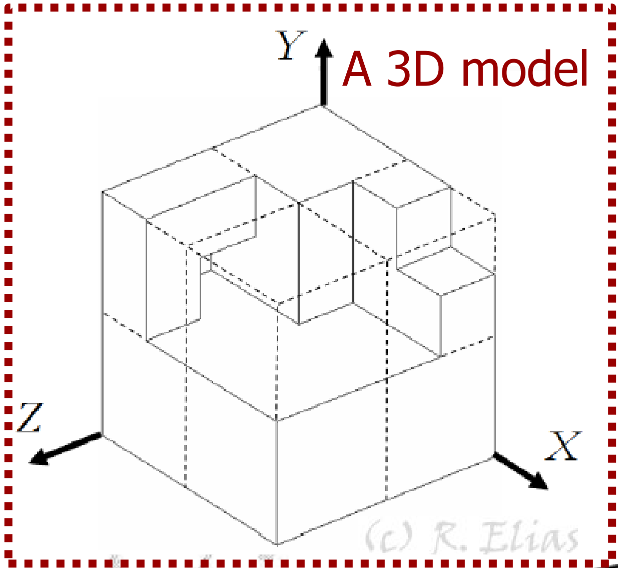
Octrees

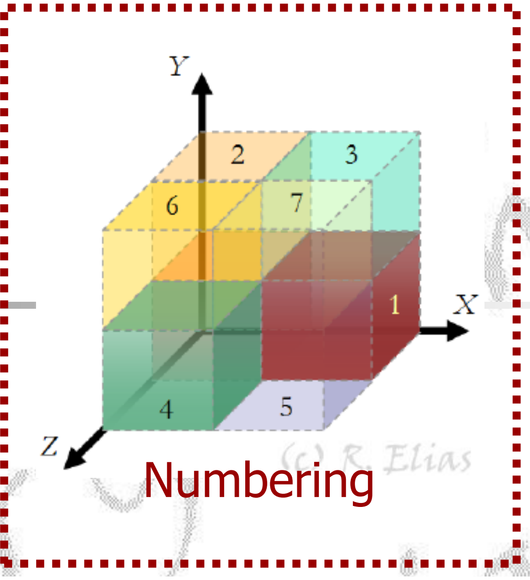
- **Octrees** can be regarded as the extension of quadtrees in 3D space.
- Splitting or division is performed in 3D space along all three directions so that **eight octants** are generated.
- Each octant could be full, partially full or empty.

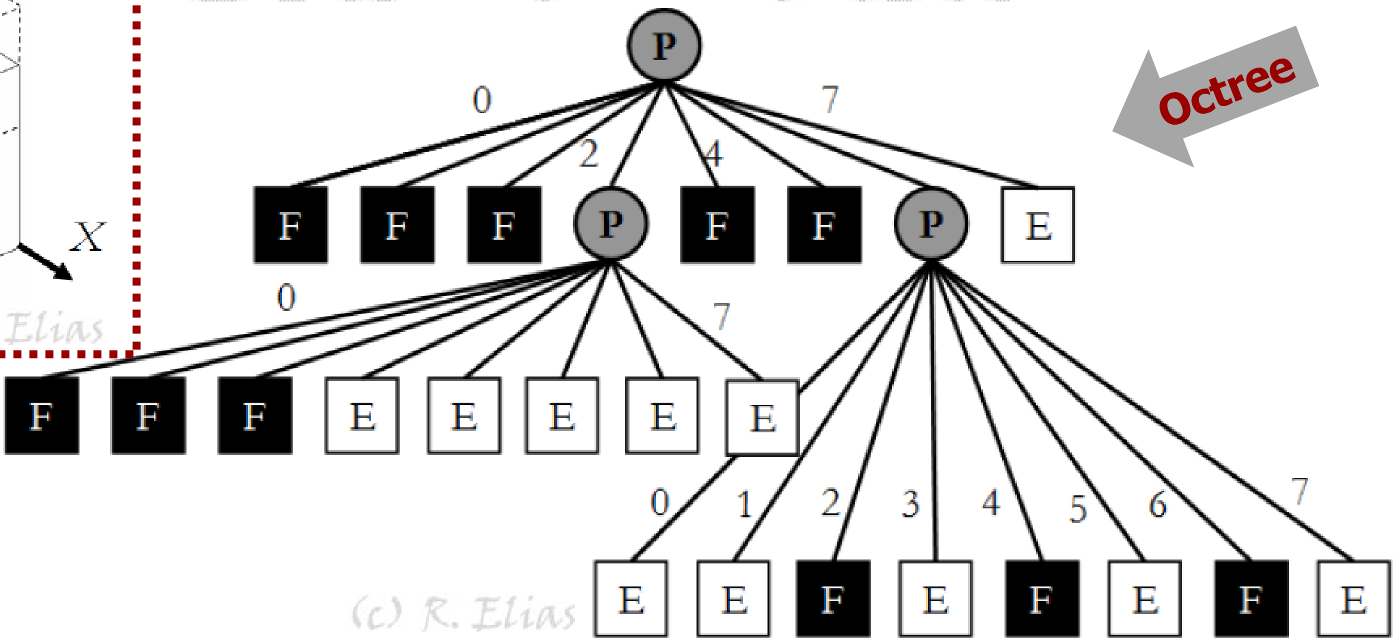


- In case of partially full octants, subdivision continues recursively until octants are either full or empty (or until a specific tree depth or a minimum volume is reached).

Octrees: An Example





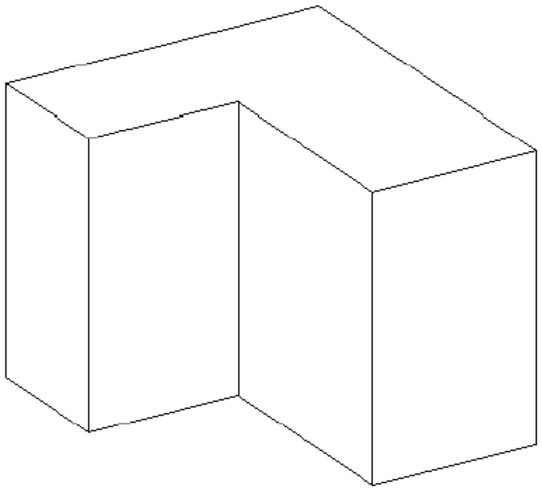


BSP Trees

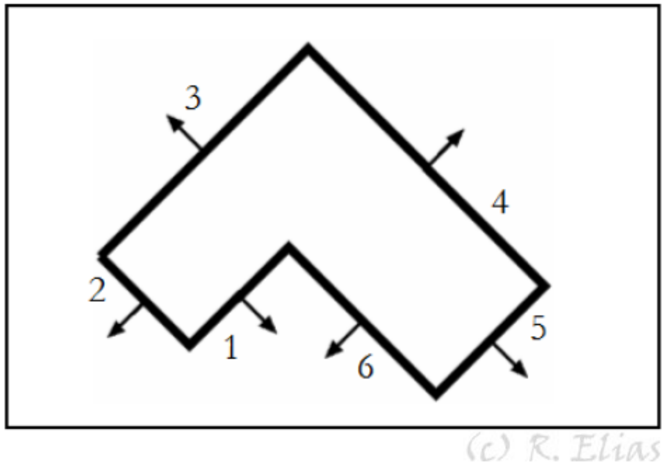
- A **binary space partitioning** (**BSP**) **tree** splits the space by surfaces of the solid being represented (i.e., as the dividing planes).
- A dividing plane splits the space into two sub-spaces representing two children; in front and in back of this plane (determined by the normal to the plane).
- The process is repeated recursively for each sub-space.
- It stops when the sub-space is entirely “in” or “out” the solid.

BSP Trees: An Example

- **Example:** A 3D model is shown below. The arrows shown indicate the outside directions of the faces. Represent this model using a BSP tree. For simplicity, you may work with the top view (i.e., ignore top and bottom surfaces). Also, you may start with face 3.



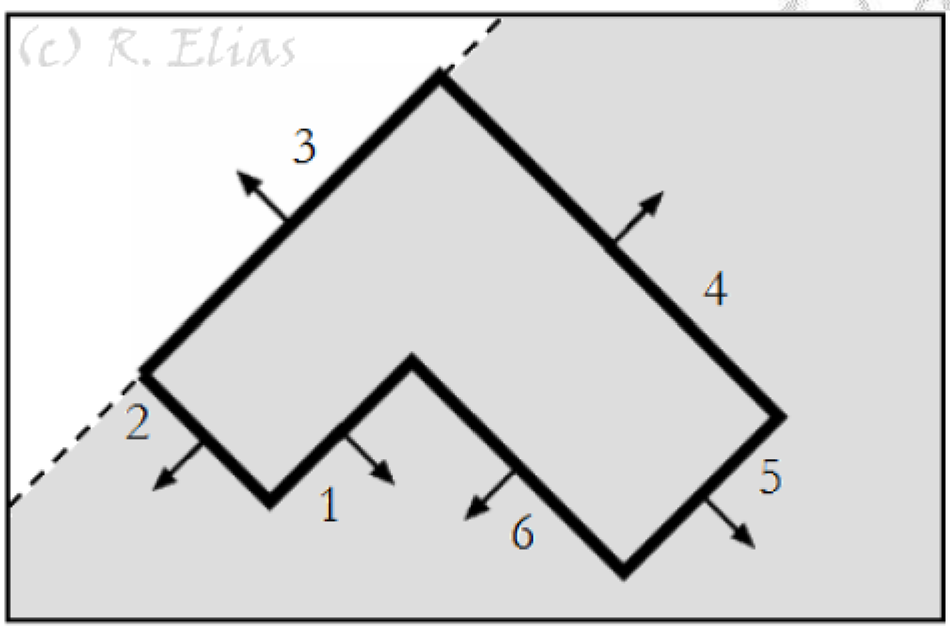
A 3D model

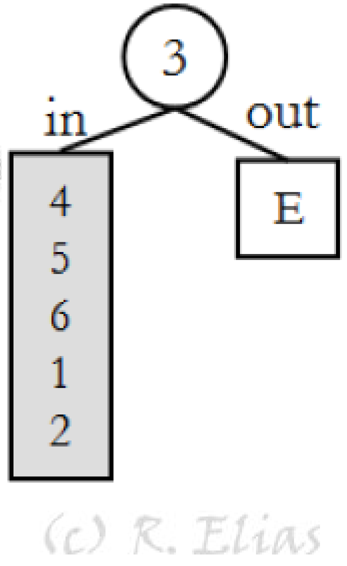


Top view

BSP Trees: An Example

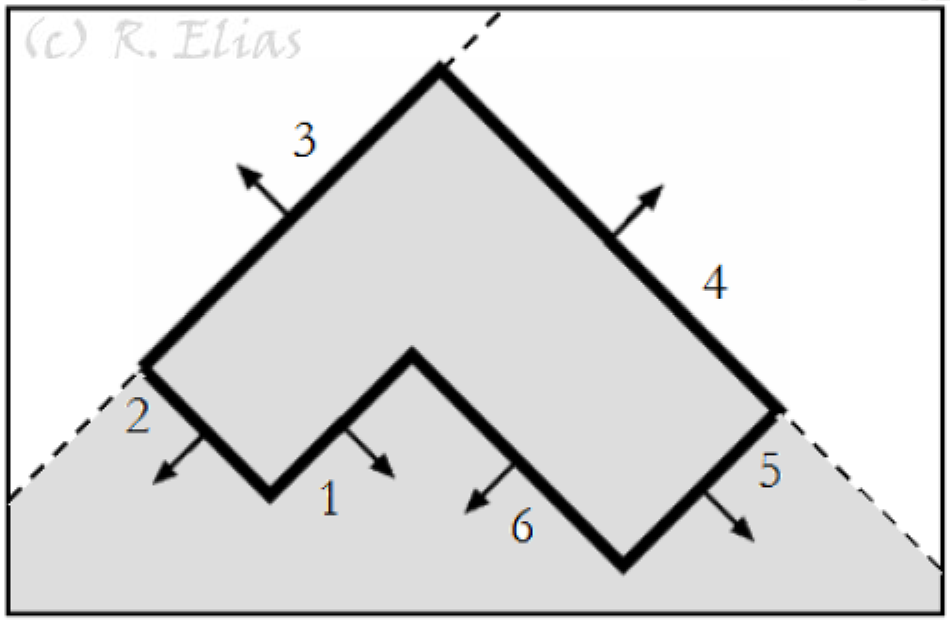
Face 3:

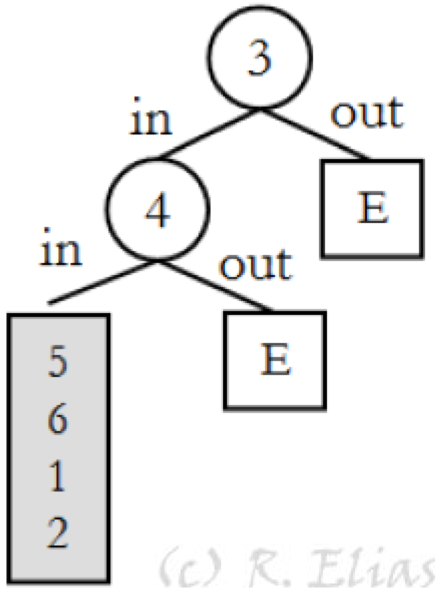




BSP Trees: An Example

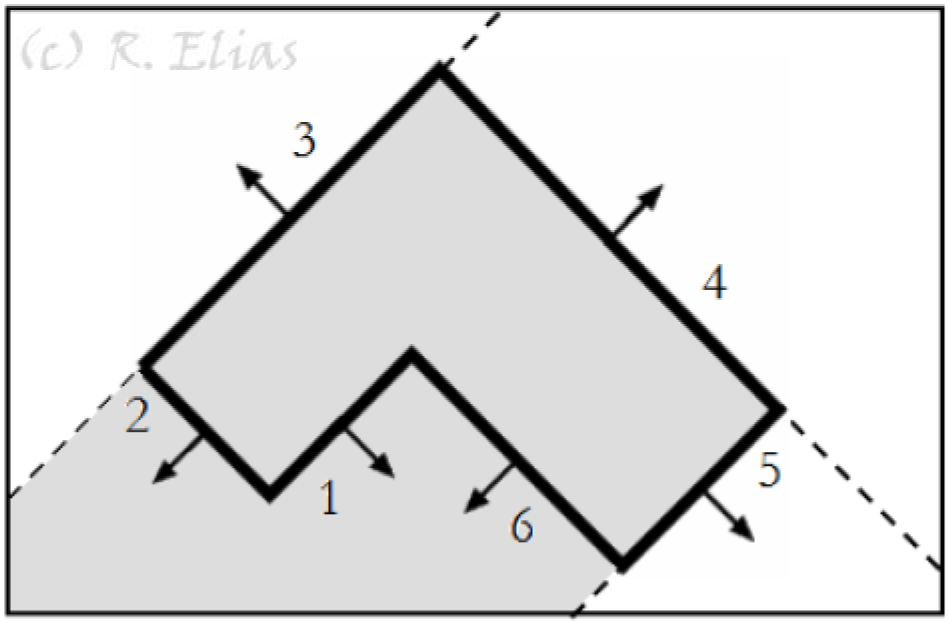
Face 4:

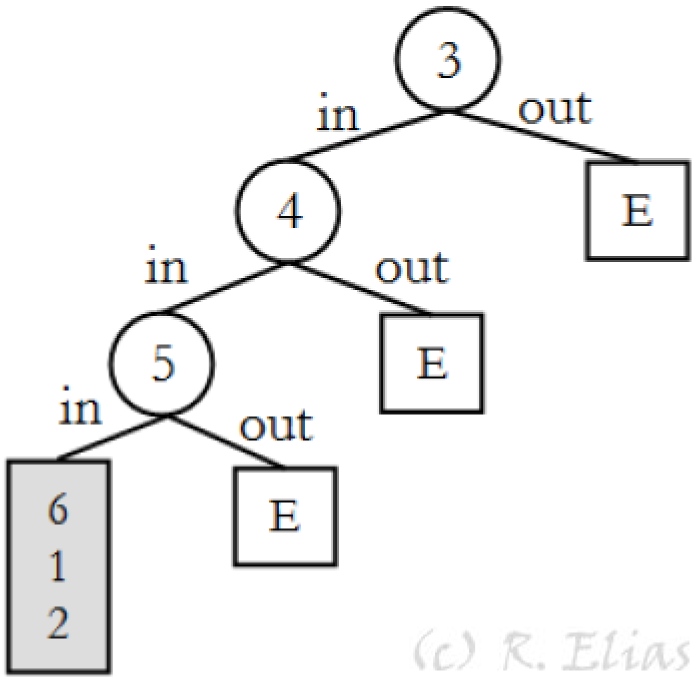




BSP Trees: An Example

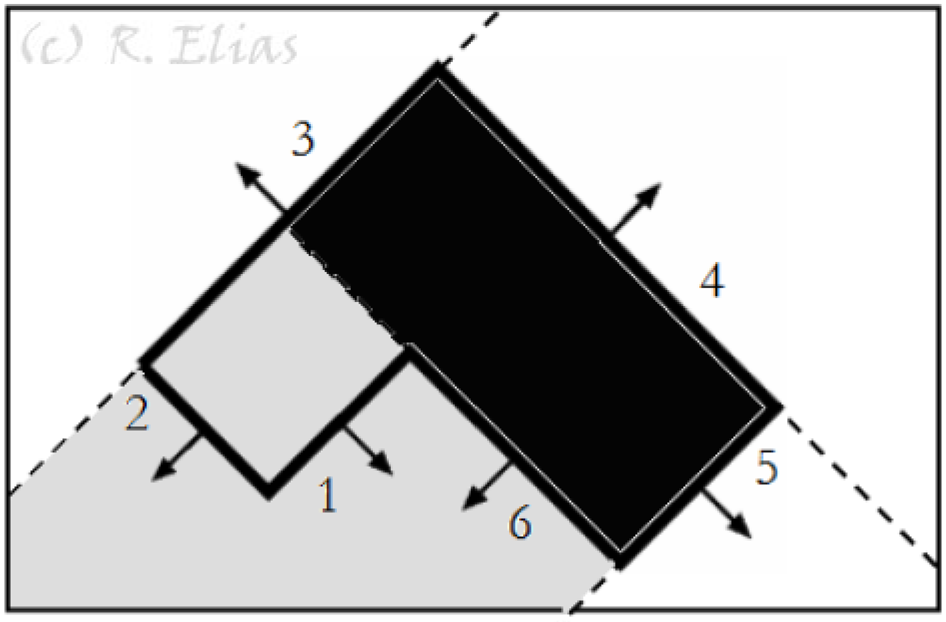
Face 5:

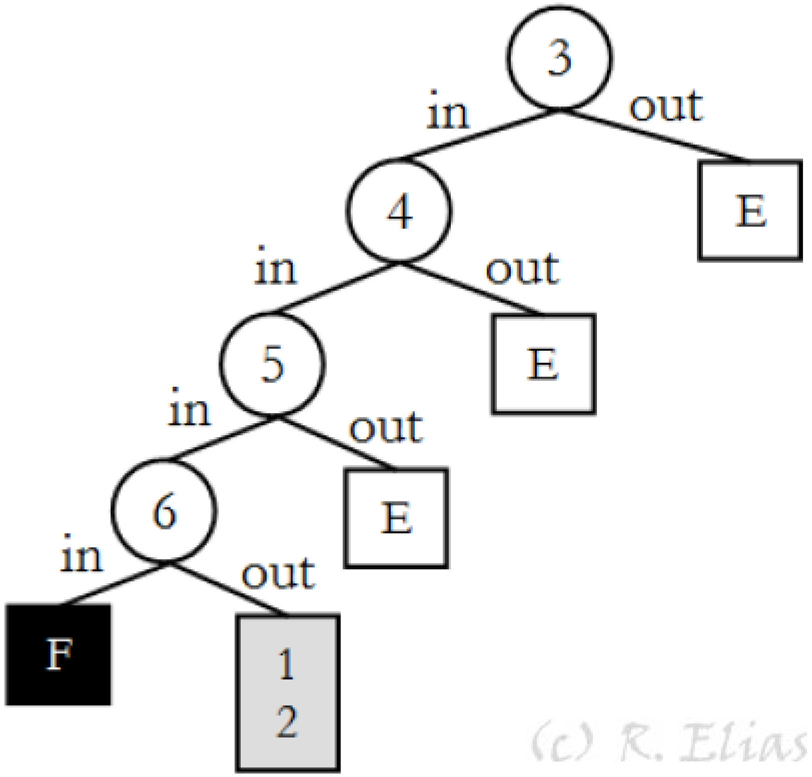




BSP Trees: An Example

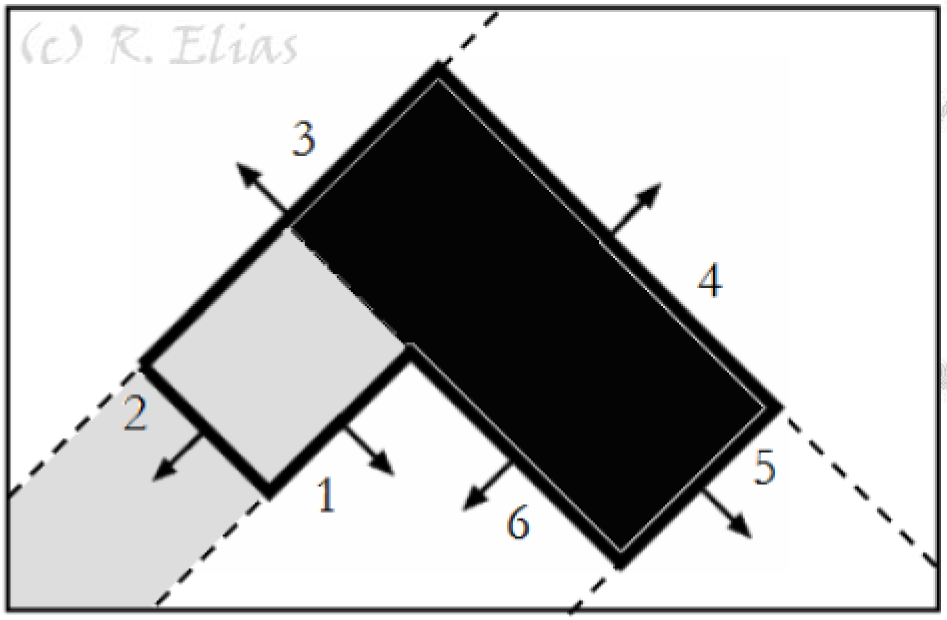
Face 6:

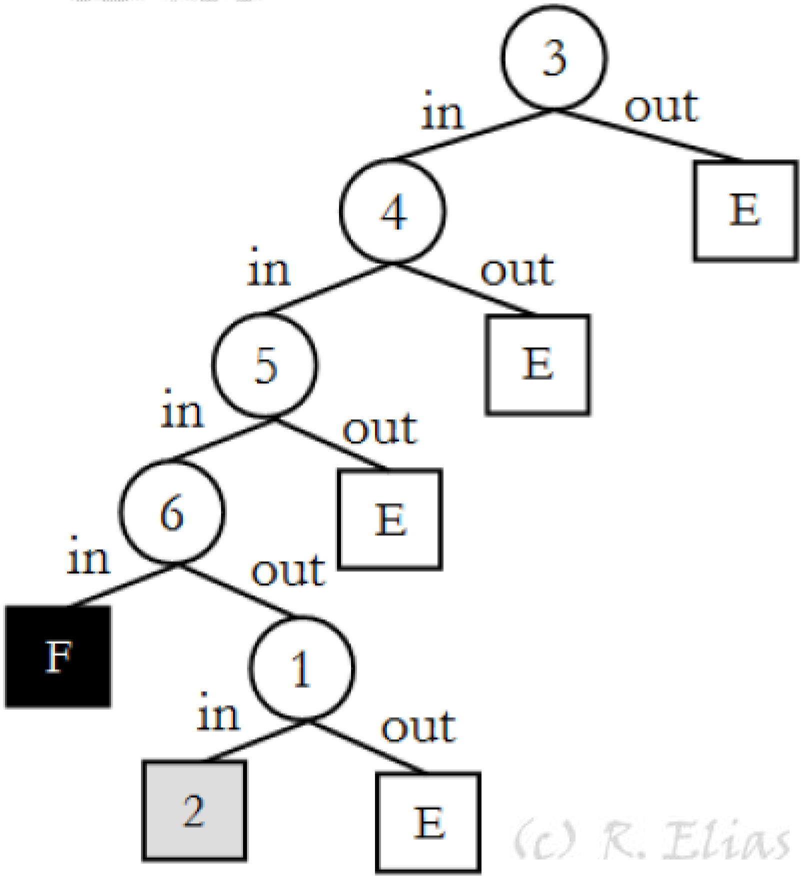




BSP Trees: An Example

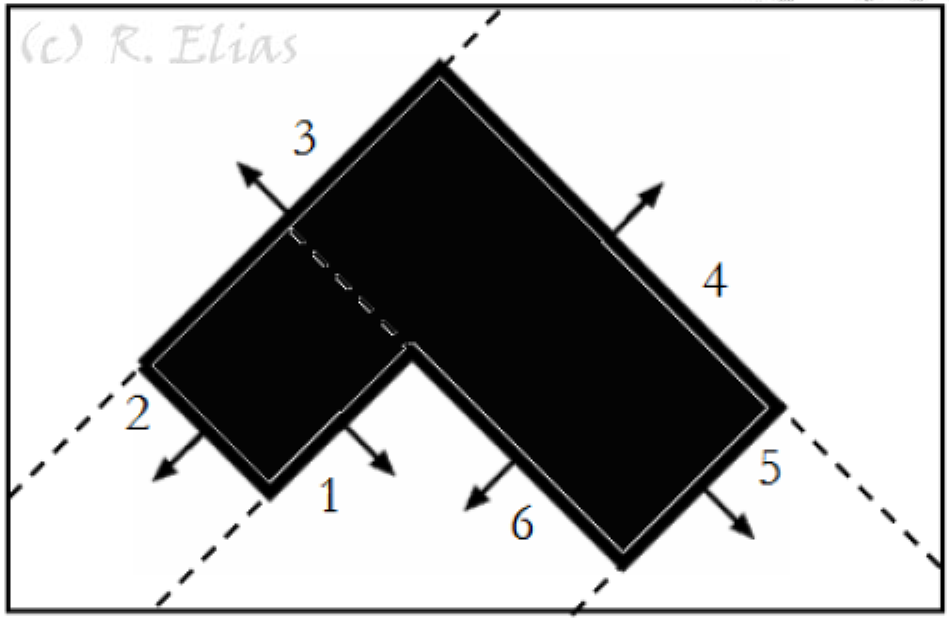
Face 1:

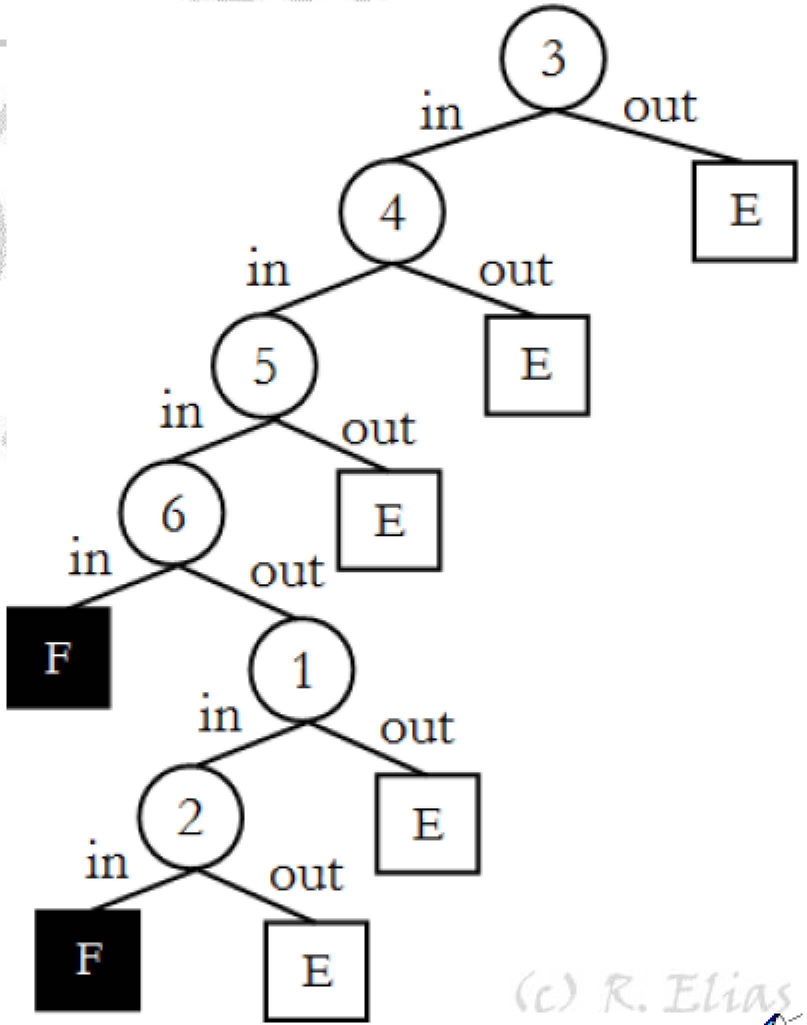




BSP Trees: An Example

Face 2:





Sweep Representations

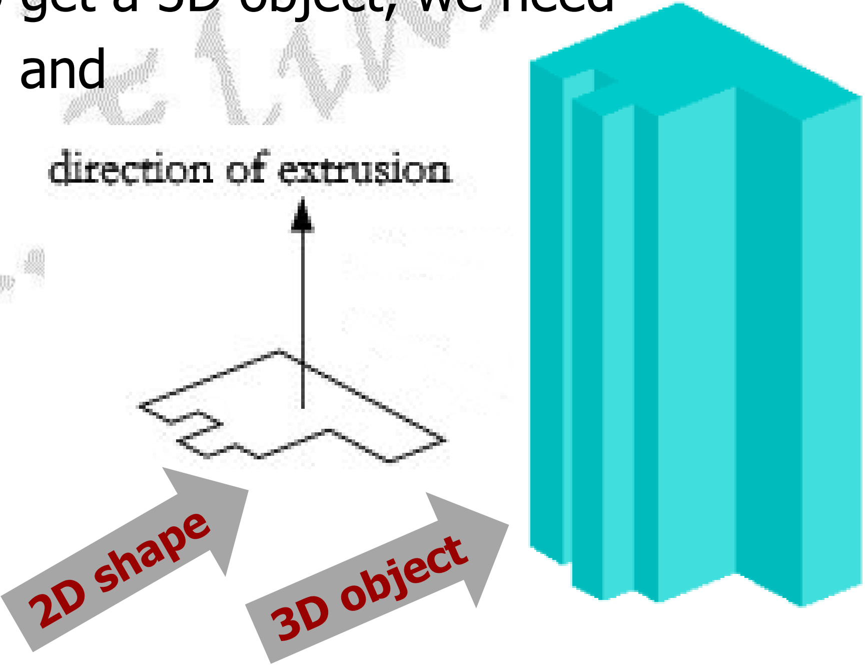
- Sweeping an object along a path or about an axis defines another object that is called a *sweep*.

- There are two main types:
  - **Translational sweeps:** Sweeping an object along a path creates a translational sweep.
  - **Rotational sweeps:** Sweeping or revolving an object about an axis creates a rotational sweep.

- Note that sweeping a 2D shape in its own plane does not generate a 3D object.

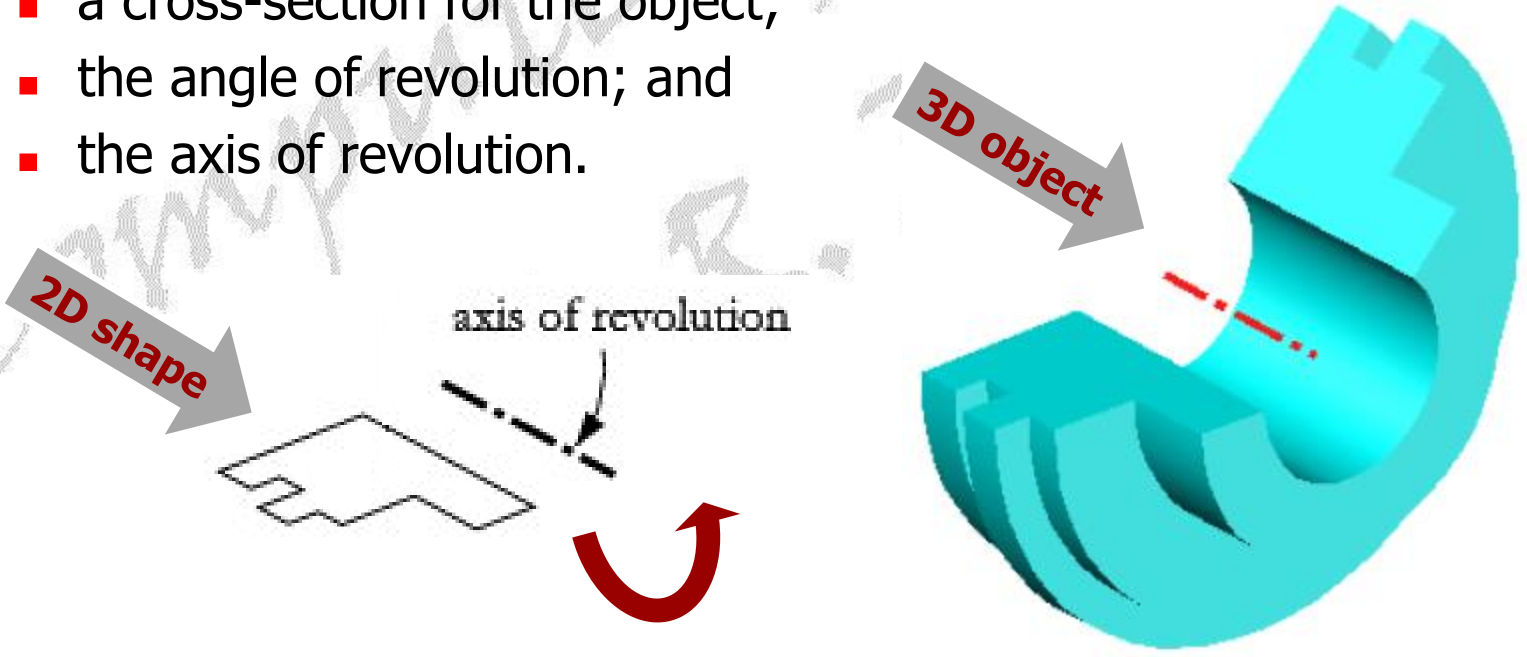
Extrusion: A Translational Sweep

- **Extrusion,** a *translational sweep,* is an important operation that converts a 2D shape located on a plane into a 3D solid model by adding thickness to it.
- In order to extrude a 2D shape to get a 3D object, we need
  - a cross-section for this object; and
  - the height and direction of extrusion, which is usually (but not necessarily) perpendicular to the plane of our profile.
- Needs may arise to use Boolean operations afterwards to subtract cavities or add more 3D features to the extruded objects.



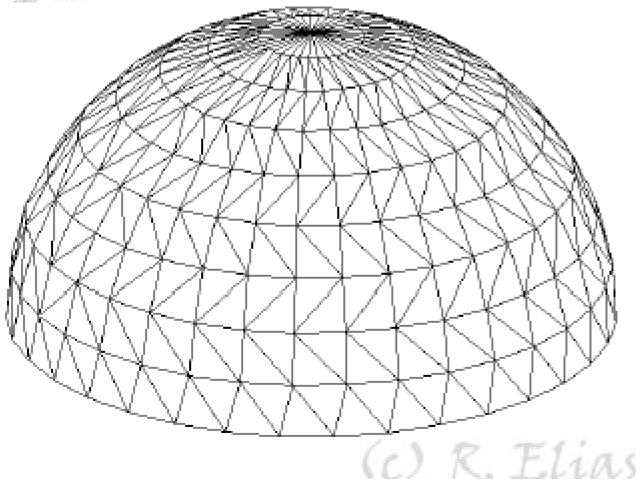
Revolution:
A Rotational Sweep

- **Revolution** converts a 2D shape into a 3D solid by revolving it about an axis.
- In order to revolve a 2D shape to get a 3D object, we need
  - a cross-section for the object;
  - the angle of revolution; and
  - the axis of revolution.



Polygonal Modeling

- In **polygonal modeling**, surfaces or faces are approximated by 3-sided or 4-sided polygons (i.e., triangles or quads).
- Each face is a planar surface (for which a normal vector can be calculated easily as a cross product of two edges).
- A group of adjacent faces comprises what is called a polygonal mesh.



Polygonal Modeling: Representations

- Polygonal meshes are represented through different ways.
  - **Independent faces:** In this representation, each face is indicated as a sequence of vertex coordinates.
  - **Vertex and face tables:** This representation uses two tables;
    - **Vertex table:** showing vertex numbers and coordinates and
    - **Face table:** showing faces and their surrounding vertex numbers.

Face table

Face	Vertices
A	$[x_1, y_1, z_1]^T,\ [x_2, y_2, z_2]^T,\ [x_3, y_3, z_3]^T$
B	$[x_2, y_2, z_2]^T,\ [x_4, y_4, z_4]^T,\ [x_3, y_3, z_3]^T$
⋮	⋮

Vertex table

Vertex	Coordinates
1	$[x_1, y_1, z_1]^T$
2	$[x_2, y_2, z_2]^T$
⋮	⋮

Face table

Face	Vertices
A	1, 2, 3
B	2, 4, 3
⋮	⋮

Polygonal Modeling: Representations

- **Adjacency lists:** In this representation:
  - A face is associated with three lists of adjacent vertices, edges and faces.
  - An edge is associated with three lists of adjacent vertices, edges and faces.
  - A vertex is associated with three lists of adjacent vertices, edges and faces.
  - Efficient way for traversal at the cost of more storage space.
- **Triangle meshes:** In this representation:
  - Only triangular faces are considered.
  - A vertex may be shared among many faces.
  - For each face, surrounding vertices and adjacent faces are stored.

Summary

- A 3D modeling
  - Wireframes
  - Boundary representations (B-rep)
  - Constructive solid geometry (CSG)
  - Spatial Representations
    - Spatial Enumeration
    - Trees
  - Sweep Representations
    - Translational Sweeps
    - Rotational Sweeps
  - Polygonal Modeling