Modeling

Master's course – M1 (Option: IVA)

Mohamed Khider University Biskra

2025– 2026

Dr : Zerari Abd El Moumene

Basic Concepts and Techniques

2D versus 3D

- Human beings live in a three-dimensional world, but when they draw, they usually use sheets of paper that have only two dimensions.
- He is therefore faced with a problem of representing a three-dimensional world in two dimensions.
- Two solutions are then available:
  - ✓represent only a flat face of objects, for example the front facade of a house or the top of a table
  - ✓try to draw the chosen scene taking into account projection laws such as perspective.

Basic Concepts and Techniques

Polygon modeling

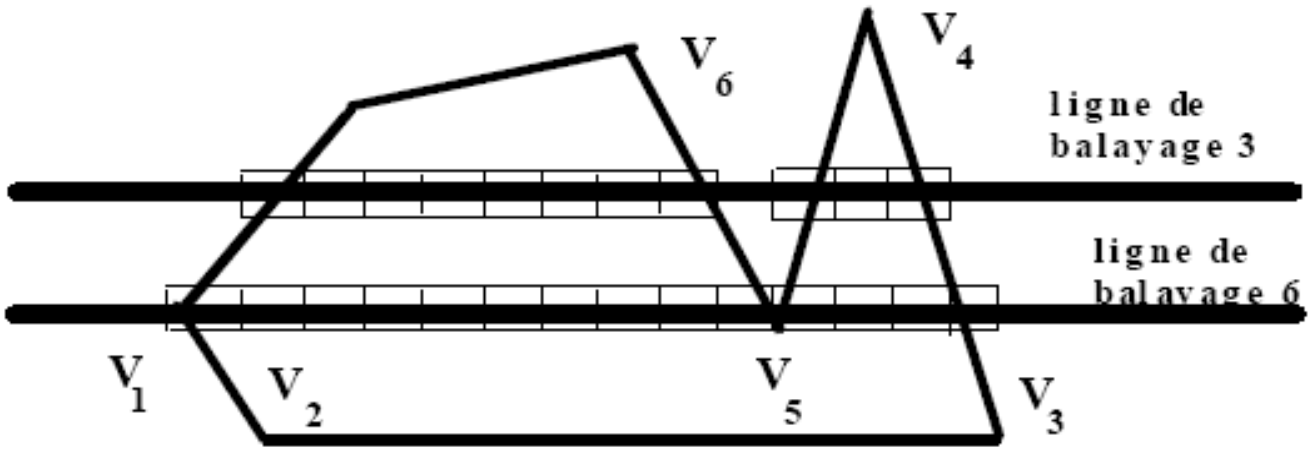
The concept of polygon

- Whether working in two or three dimensions, the polygon plays an extremely important role.
- By polygon we mean a plane figure defined by a list of points (the vertices) connected by straight line segments (the edges).
- The vertices are assumed to be all different, the edges must not intersect, and an edge connects the last vertex to the first.
- A polygon is concave if there is at least one internal angle greater than 180^{o};
- It is convex, if not concave.
- Polygons are most commonly used in two dimensions, because filling them with colors quickly builds attractive images.

Basic Concepts and Techniques

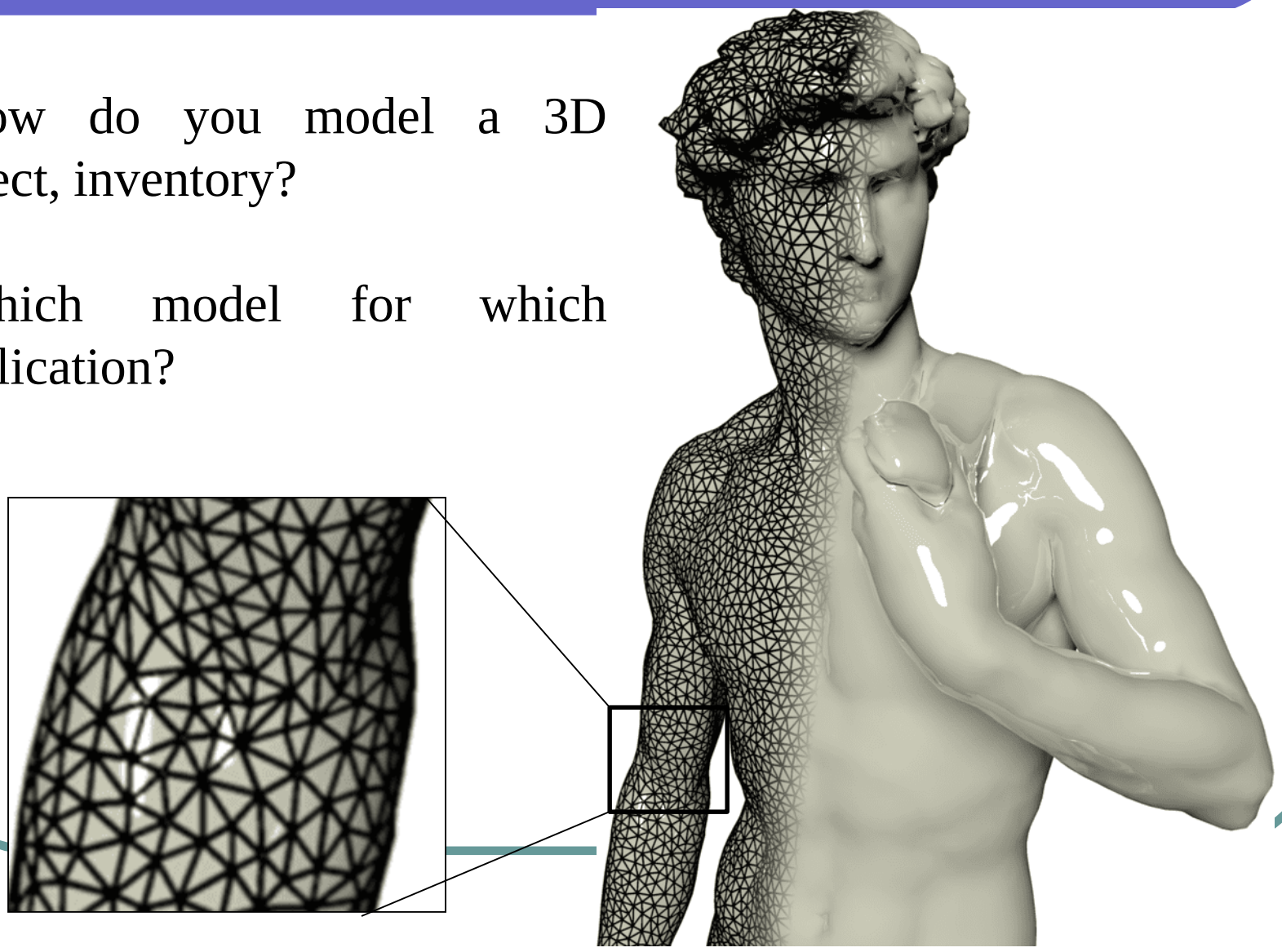
Polygon Filling (1)

- The most popular basic algorithm for filling a polygon is to scan the polygon with horizontal lines (scan lines).
- On each line, repeat the following steps:
  1. find the intersections of the scan line with all edges of the polygon
  2. sort these intersections in ascending order of x coordinates
  3. set all pixels between the intersection pairs to the given value.



Modeling : Course Goal

- How do you model a 3D object, inventory?
- Which model for which application?



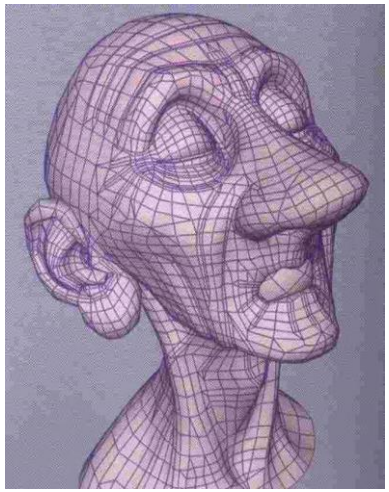
Modeling approaches

Concept of “geometric model”

- Mathematical model of the virtual object (equation of its surface)

How to create this model?

- How quickly does it appear?
- That it doesn't take up too much memory?
- That it can be easily modified?



Modeling

- Two types of objects to model
  - virtual objects / invented
  - objects real
- Multiple modes of creation
  - **interactive** by graphic designers
  - **automatic** from the real
    - 3D scanner , medical data , etc...
  - **procedural**
    - complex scenes, terrain, etc...
  - Or a mixture of these fashions…
- Different uses
  - display, animation, physical simulation, etc...

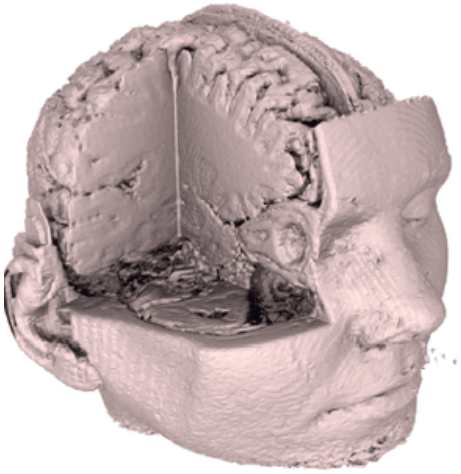
Modeling

- Building the scene objects
  - Primitives geometric
  - Transformation operations (placing objects)
- Difficulties
  - 3D scene / 2D screen
  - Placement of objects
  - Editing objects (meshes)

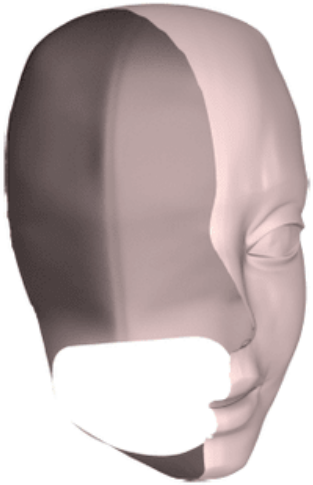
The models geometric

- Wire
- Models **not structured**
  - points
- Models **surface**
  - meshes
  - surfaces parametric
  - surfaces of subdivision
  - surfaces implicit
- Models **volumetric** (Solids)
  - octrees
  - CSG
- Models **procedural**
  - Fractals
  - Grammars / L-systems
  - System of particles
- Models **based on of images**
  - Acquisition and rendered

■ Modélisation volumique



■ Modélisation surfacique



The models geometric

Wireframe modeling

- This model only keeps the coordinates (x, y, z) of the vertices and the edges that join them.
- It makes the object appear as a set of points connected by lines,
- Advantages:
  - Requires minimal computing power.
  - Offers a saving of memory space.
  - Provides very fast visualization (real time).
  - Represents the general shape of the object.
  - It is very simple. .

The models geometric

Wireframe modeling

- ***Disadvantages***
  - Ambiguity, nothing distinguishing emptiness from fullness.
  - The inability to obtain realistic images.
  - The ability to create objects without any physical sense.
  - The difficulty of solving the problem of eliminating hidden parts.

The models geometric

Unstructured models

Clouds of points

- Models obtained
  - by manual digitalization or by scanner
  - by reconstruction from of images
- Advantages
  - “natural” ie design atomistic
  - simple to display
  - powerful theory for antialiasing
  - rendered adaptive
- Disadvantages
  - cost memory
  - with difficulty editable
  - Calculation of the normal

The models geometric

Modeling surface

- Meshes
- Surfaces parametric
- Surfaces of subdivision
- Surfaces implicit

The models geometric

Modeling surface

❑This model allows objects to be defined by associating pieces of surfaces with contours delimited by edges.
❑The surface approach is superior to the wireframe approach. However, it highlights the weaknesses of the previous model.

Advantages:
modeling :

- Is more developed than the wireframe approach.
- Is widely used in CAD/CAM.
- Allows you to create smooth objects.
- Allows you to build very complex objects

- ***Disadvantages:***
  - More expensive in memory space
  - The elapsed time is greater than that of the wireframe model.
  - "Attachment" problem, each surface is described independently of the others.

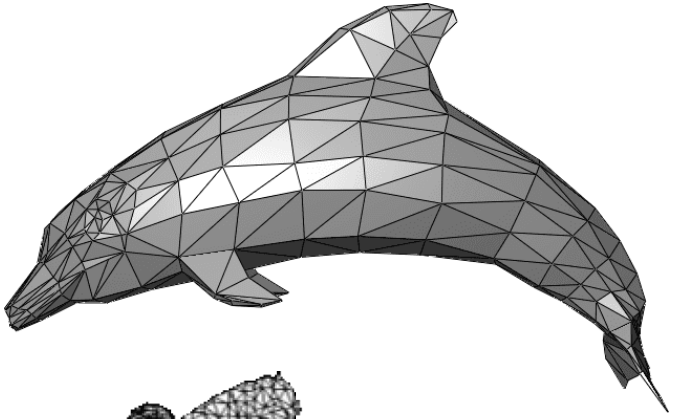
Surface modeling :
Polygon Meshes

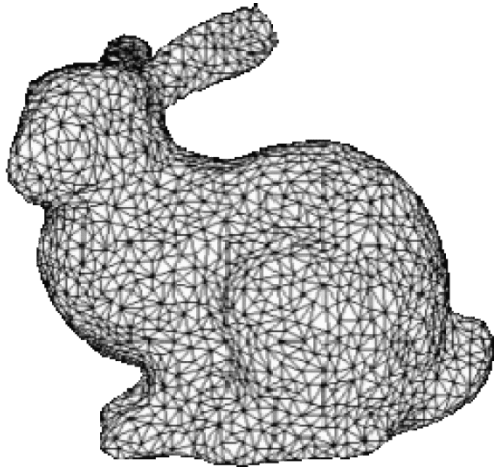
Meshes of polygons

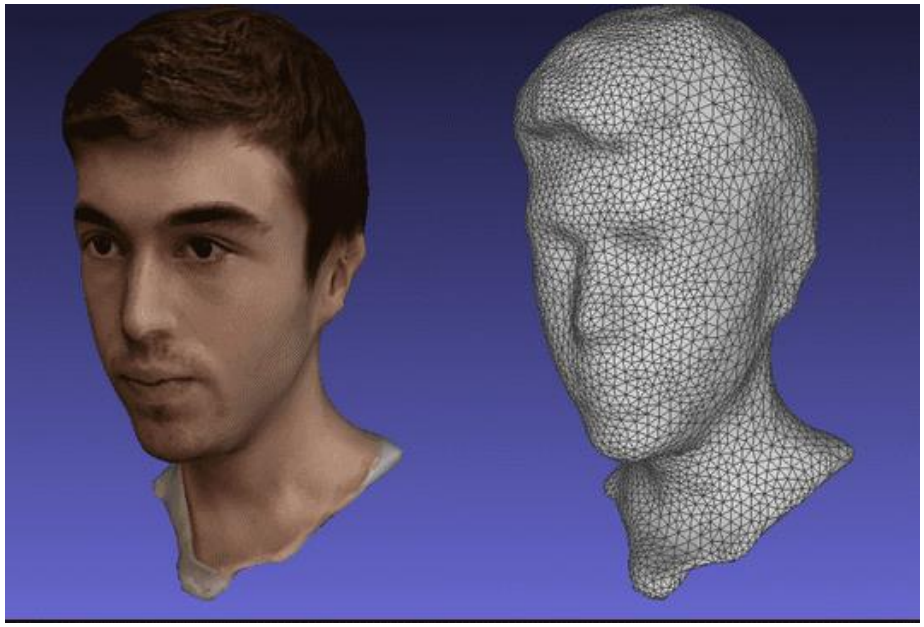
- A ***mesh*** is a three-dimensional object made **up** of vertices, edges and faces organized into polygons forming a wireframe structure.
- Faces usually consist of triangles, quadrilaterals, or other simple convex polygons, because this simplifies the rendered.
- Faces can be combined to form more complex concave polygons, or polygons with holes.
- Many operations can be performed on the *meshes* , such as boolean logic, smoothing, etc.
- Easy display with OpenGL
- Graphical modeling tools such as Blender or 3D Studio Max allows you to create these objects wired.

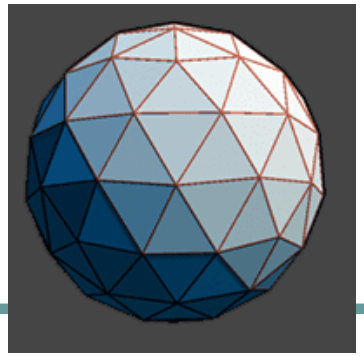
Surface modeling :
Polygon Meshes

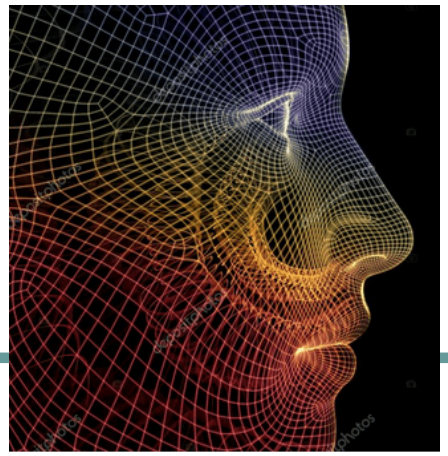
Meshes of polygons





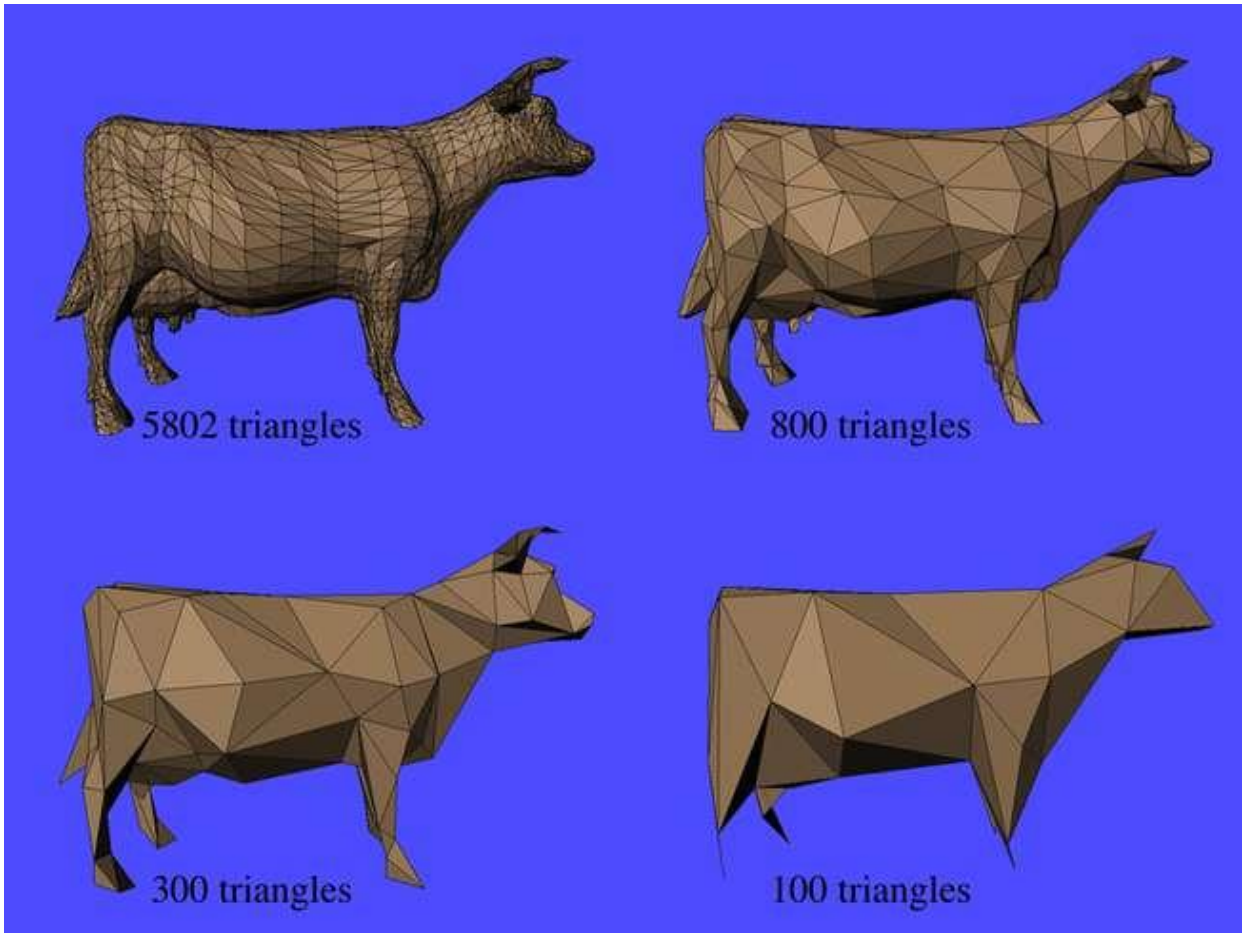






Surface modeling : Levels of detail of triangulated surfaces LOD (Level Of Details)

Different representations of the same object are computed, each with a different level of detail. During application execution, one representation of the object is selected and visualized.



Surface modeling : Levels of detail of triangulated surfaces LOD (Level Of Details)

Objective

- **Observations:**

– detailed object → lots of polygons

– rendering cost = function of the number of polygons

– size of polygons in image space = function of the distance from the observer.

- **Idea :**

– adapt the resolution (the number of polygons) of the mesh according to the point of view

- **Challenges:**
  - how to calculate simplified versions of the mesh?
  - how to choose the right resolution?
  - how to make this efficient from a GPU perspective?

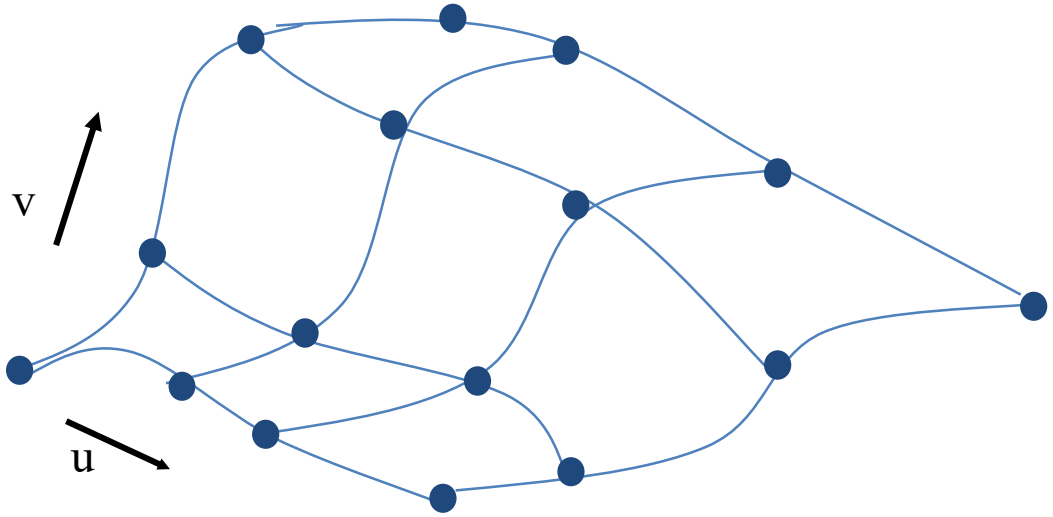
Surface modeling :
Parameterized surfaces

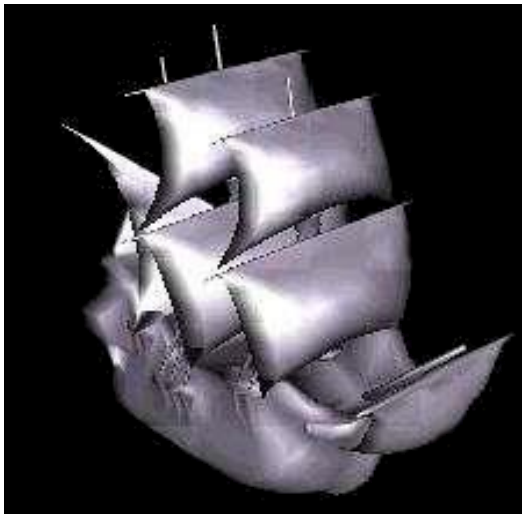
Parametric modeling is a type of explicit modeling that uses functions.
This is historically used to describe curves and surfaces. Continuity properties can be easily identified by differential analysis of the functions used. The functions used are often polynomials because of their smoothness. The higher the degree of the polynomial, the smoother the curve or surface.

Surface modeling :
Parameterized surfaces

Facets: Set of tiles

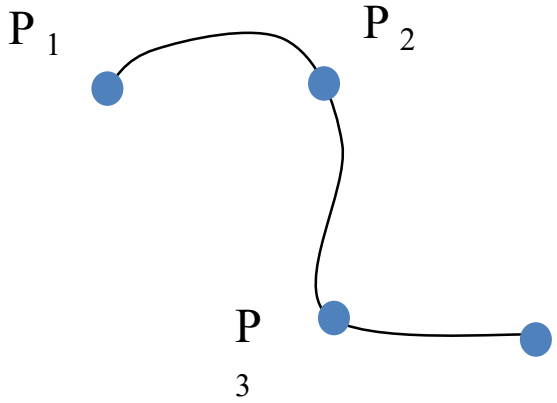
Tile: a piece of parametric surfaces (Bezier , Bsplines , COONS…)

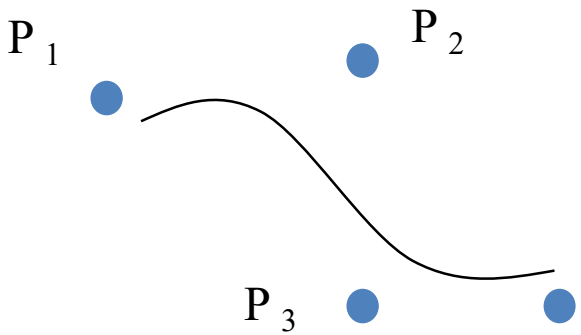




Surface modeling :
Parameterized surfaces

- definition from control points
- Interpolation or approximation

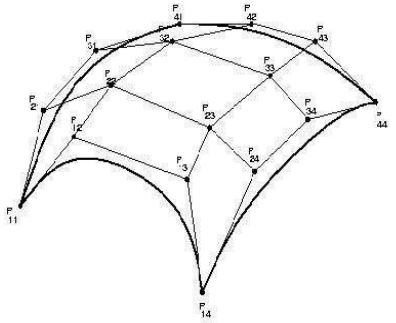




Surface modeling :
Parameterized surfaces

Freeform surfaces

- The parametric representation of a curve in space is of the form :
  - **C(t)=(X(t),Y(t),Z(t))**
  - where $t \in [\, t_{min}, t_{max} \,]$
- As well as that of a surface :
  - **S(u,v)=(X(u,v),Y(u,v),Z(u,v))**
  - where $u \in [\, u_{min}, u_{max} \,]$, $v \in [\, v_{min}, v_{max} \,]$
- Each value of t (for curves) and u and v (for surfaces) corresponds to a point in the curve (surface).



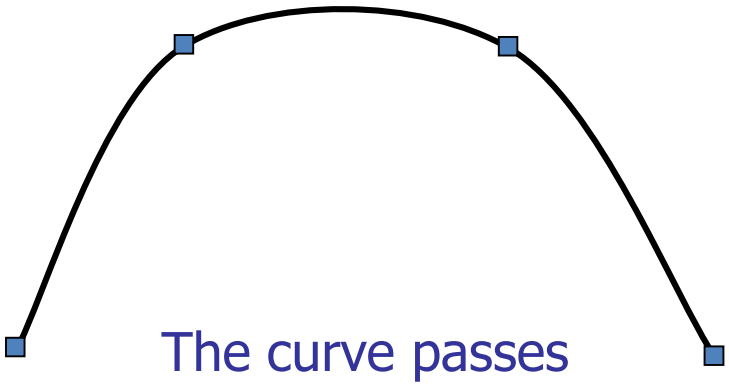
Surface modeling :
Parameterized surfaces

A curve or a surface is generally given by a set of points. We then distinguish 2 types of methods:

Approximations *: the curve or surface is defined by the points, but does not* necessarily pass through them. These are the adjustment and control point methods. (Bezier approximations - B- spline approximations)

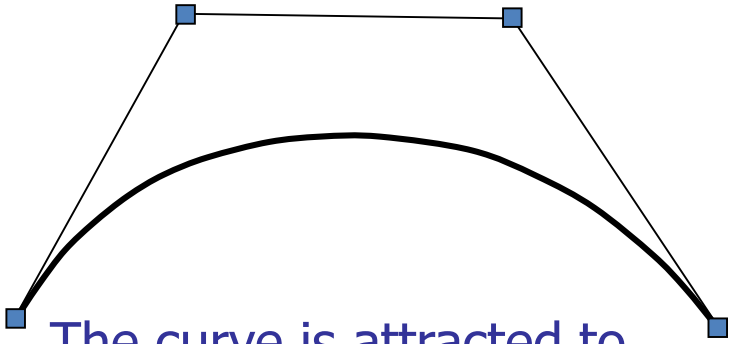
•***Interpolations*** *: the curve or surface passes through the points.*

Interpolation



The curve passes through the control points

Approximation

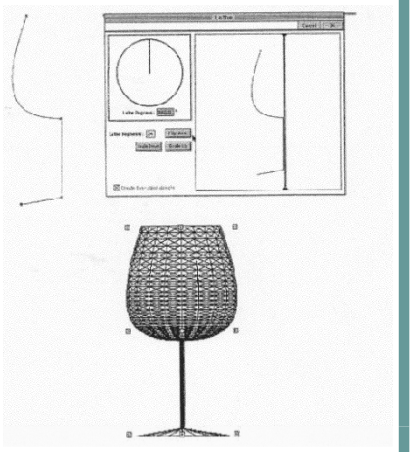


The curve is attracted to the control points

Surface modeling

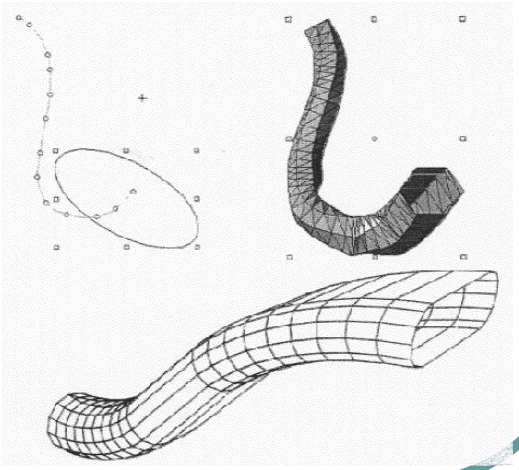
Surfaces parameterized: Surfaces created from curves

- Surfaces of revolution: Surface created at leave
  - of a curve
  - an axis of rotation
  - position of the curve relative to the axis of rotation
  - an angle of rotation



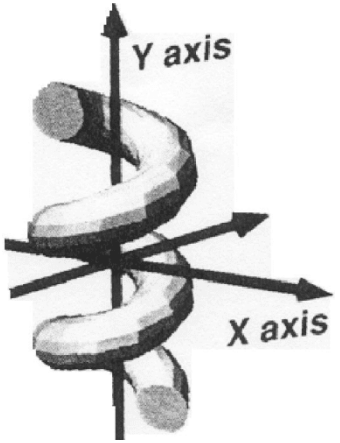
- surfaces : Surface created from a plane curve by giving it the thickness

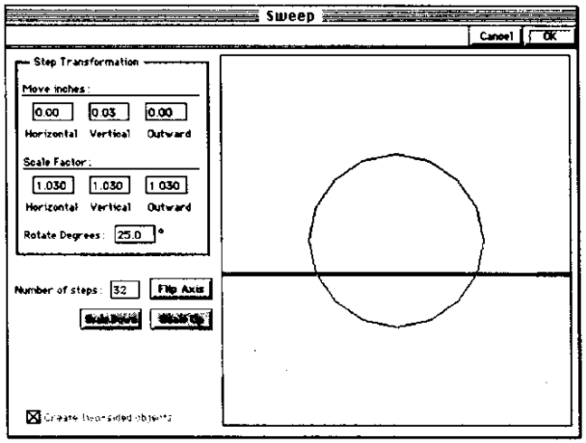
  - **Extrusion generalized**
    - **A closed plane curve**
    - **a path**
    - **position and modification of the plane curve along the path**



Surface modeling:
Surfaces parameterized: Surfaces created from curves

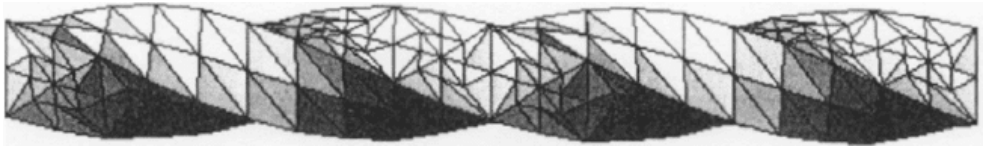
- Sweeping: Construction by shift
  - A curve plane
  - An axis of rotation
  - An angle of rotation
  - A shift



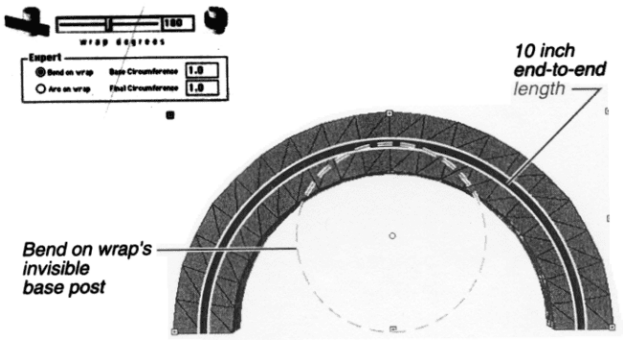


Modeling surface:
Surfaces parameterized: Construction by deformation (warping)

- Construction by Deformation
  - Twist

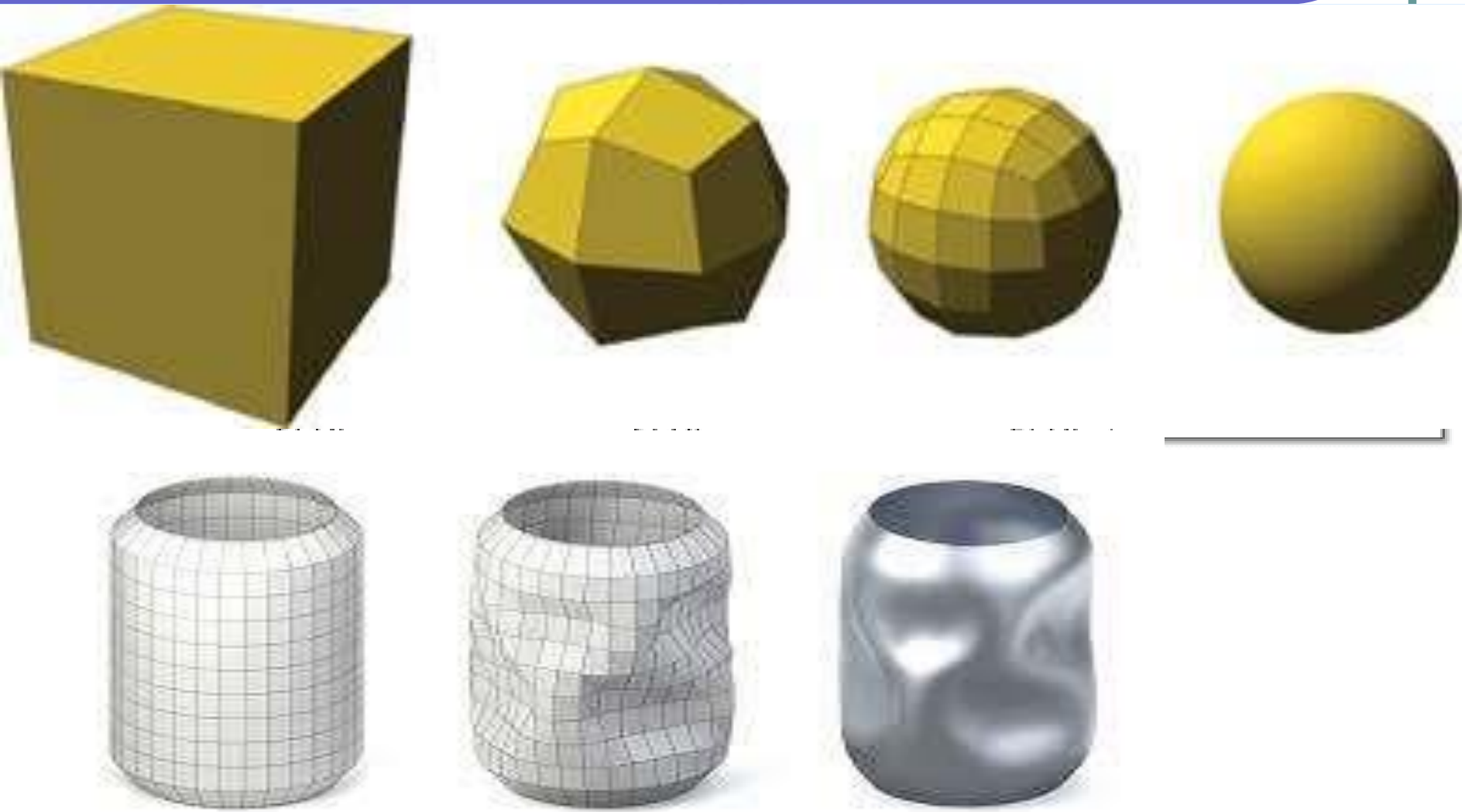


  - Winding



Surface modeling

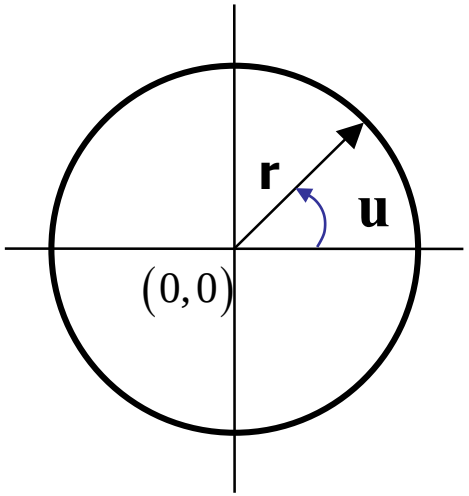
Surfaces of subdivision



Surface modeling
Implicit

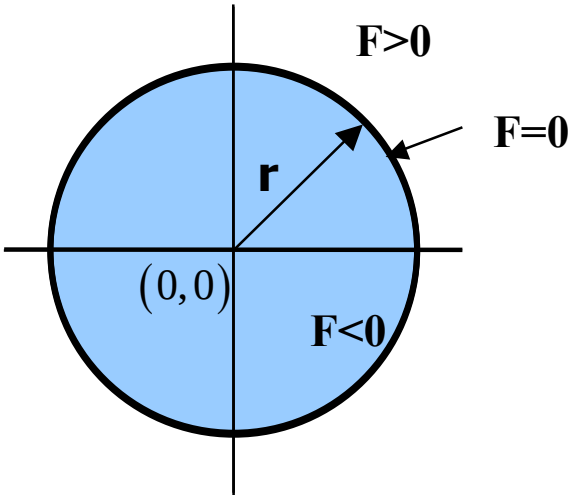
- A three-dimensional implicit surface is a surface consisting of points P(x, y, z), satisfying the implicit function f(x, y, z)=0.

Parametric



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

Implicit



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

Surface modeling
Implicit

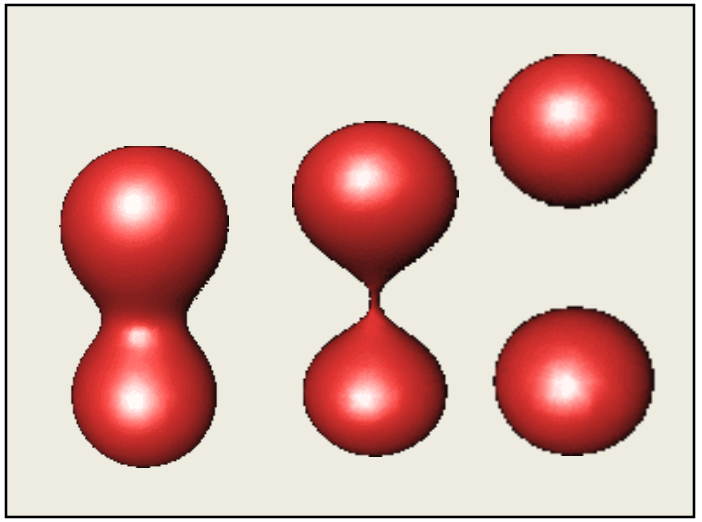
$$S = \{ P(x,y,z) / f(x,y,z) = iso \}$$

Interest: Combining elements

union: $f = max(f_1, f_2)$

Intersection: $f = min(f_1, f_2)$

"mixture": $f = f_1 + f_2$



Surface modeling

Implicit

Advantage:

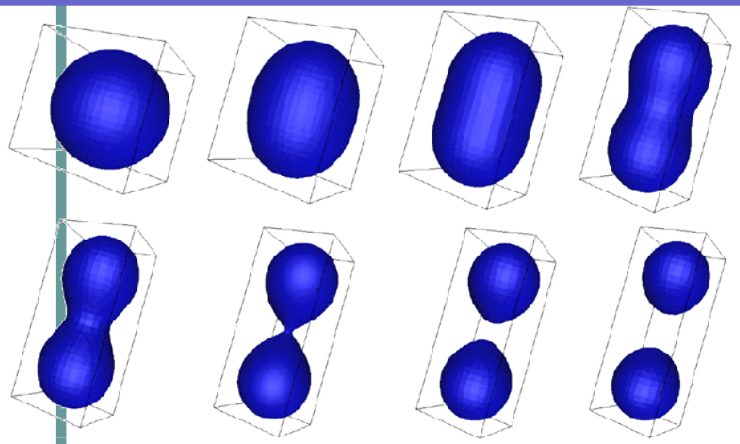
- The possibility to create smoother objects.
- Allows you to merge multiple shapes.
- Defines a surface with few parameters, compared to the parametric surface.

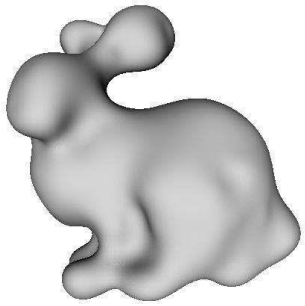
Disadvantages:

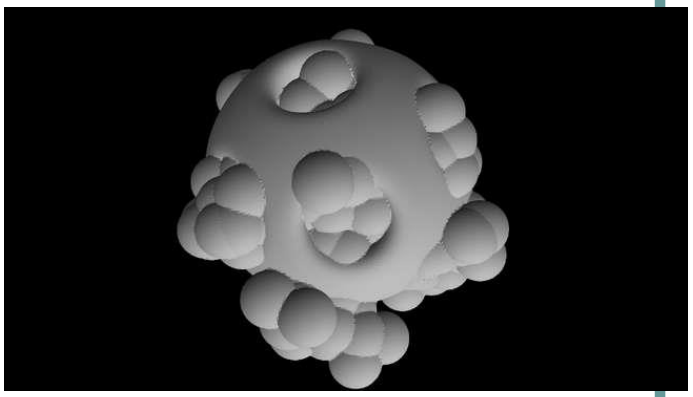
- Expensive rendering (in computational time).
- Difficulty of texture mapping (requires surface polygonation).

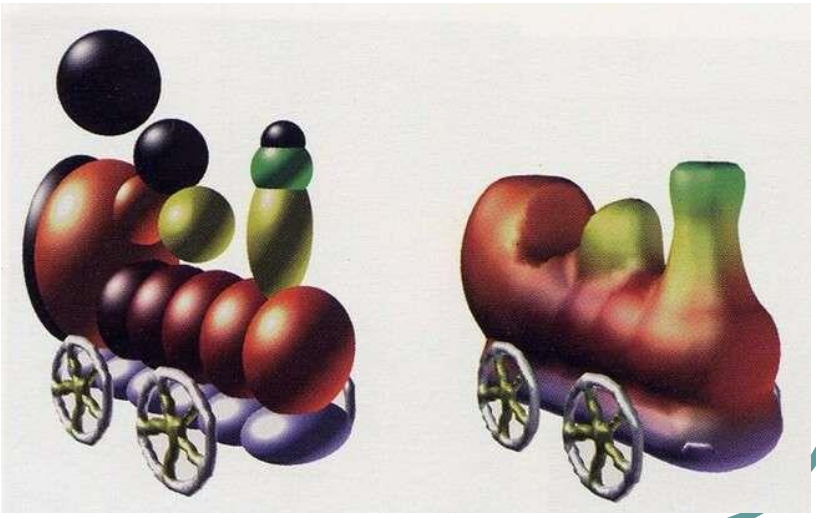
Surface modeling

Implicit









Volumetric modeling

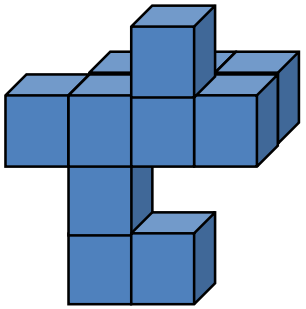
- 3D objects are not treated as a collection of polygons, but as entire entities (volumes).
- It consists of providing information on the space occupied by the object.
- To define and manipulate a volume object, it must translate their properties (vertices, edges and radius) into mathematical formulas.
- This approach describes and visualizes solid objects.
- Among the types of volume modeling we distinguish the methods (CSG, BREP, Hybrid).

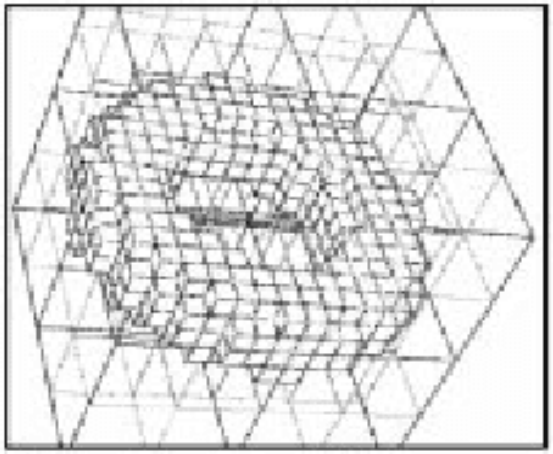
Volumetric modeling

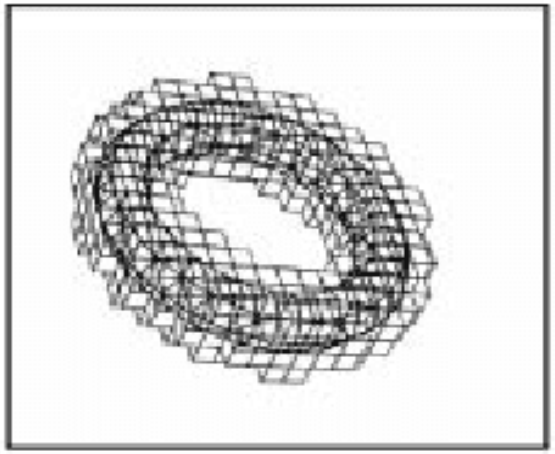
Discrete volumes: voxels

Voxels = elements of a 3D grid
Presence or absence of matter
a definition – called voxalized – of *V* is given by:

$$V = \sum_{i=1}^{N} v_i$$





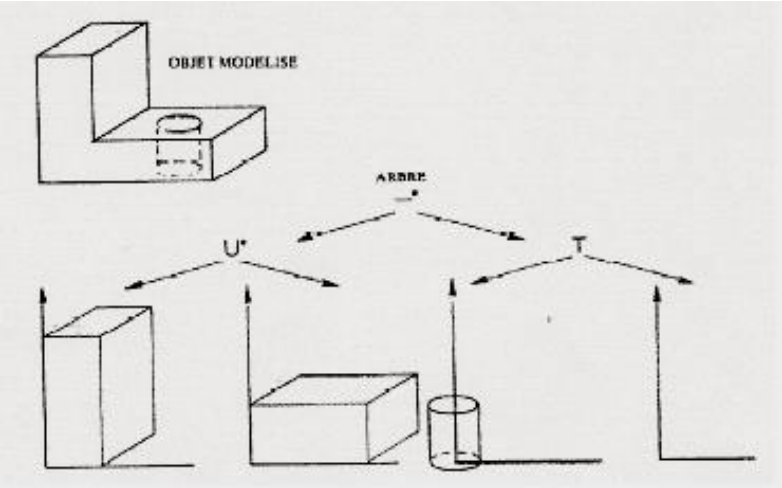


Volumetric modeling

- Constructive Solid Geometry (CSG)
- Octrees

Volumetric modeling:
Constructive Solid Geometry (CSG)

- Tree representation of construction:CSG



- Representation by a grammar
  Exp -> prim / transf prim / op exp exp
  Prim -> cube / sphere / cone / ...
  Transf -> translation / homothety / rotation
  Op -> union / intersection / difference

Volumetric modeling:
Constructive Solid Geometry (CSG)

Generate complex shapes using primitives.

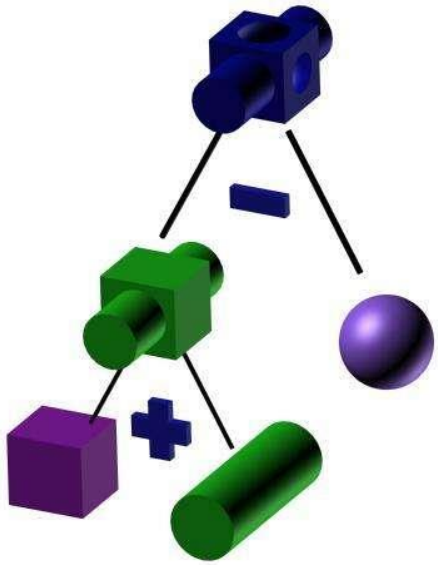
Drawing an object: cropping parts, drilling holes, etc.

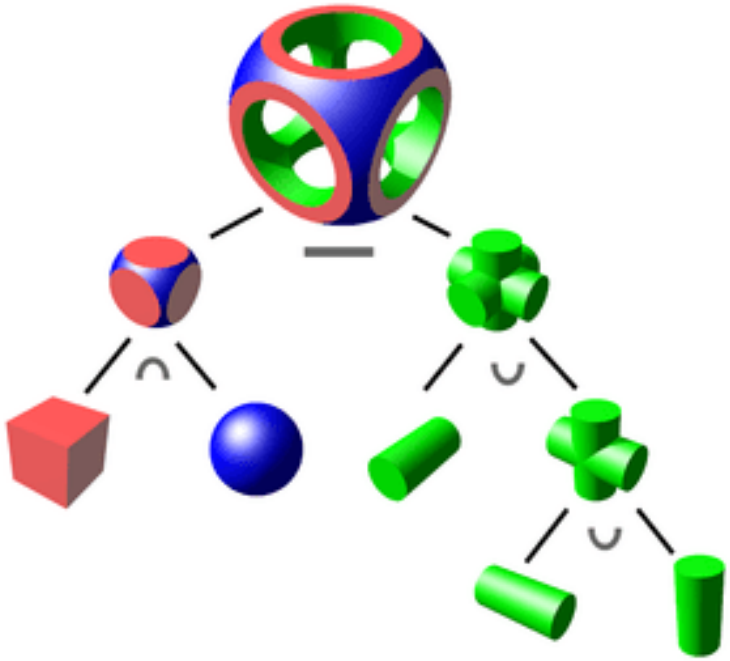
Glue parts together

Commonly used in CAD.

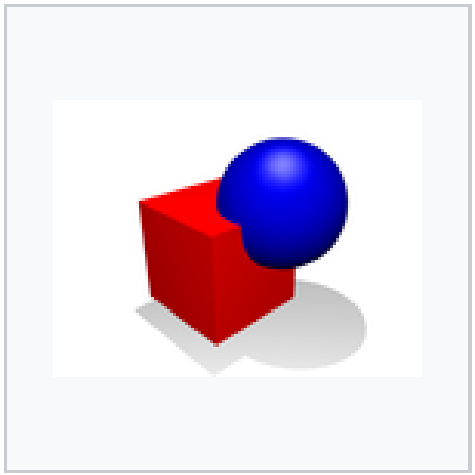
Possible operations

- union
- intersection
- subtraction

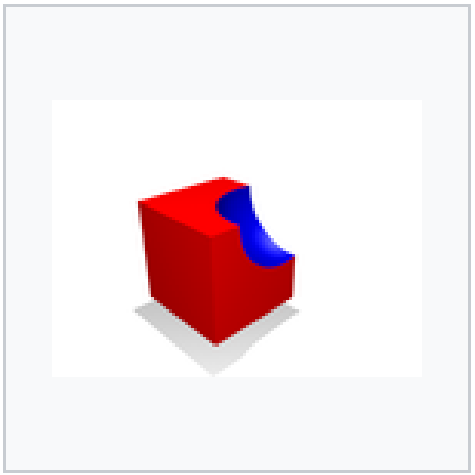




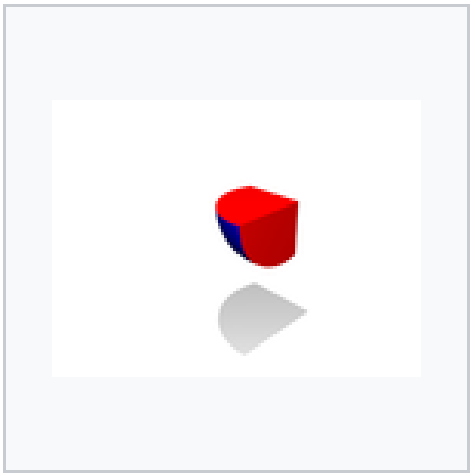
Volumetric modeling:
Constructive Solid Geometry (CSG)



Union
Merger of two objects into one

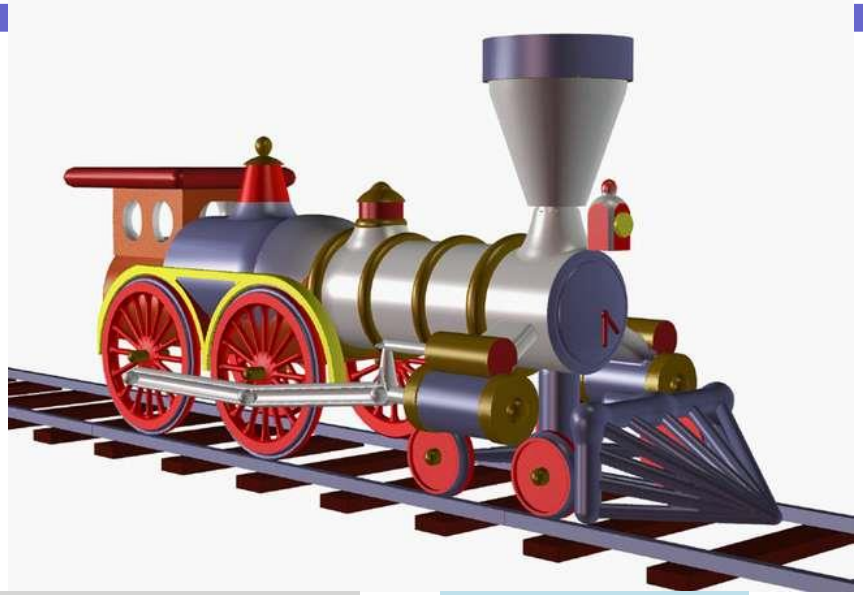


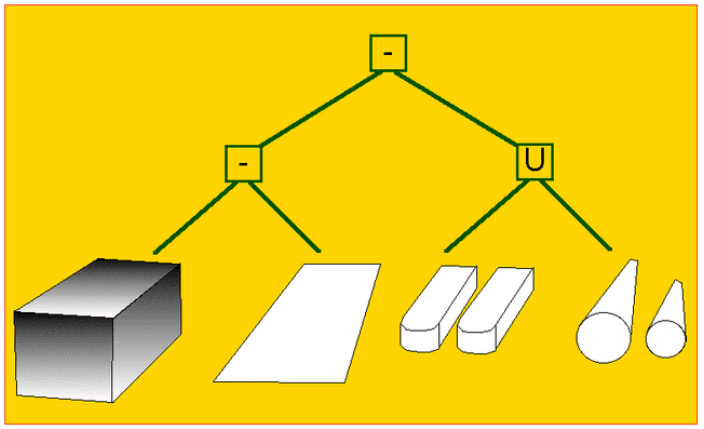
Difference
Subtraction of one object from another

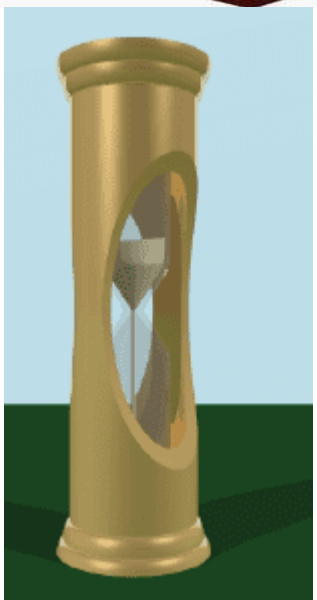


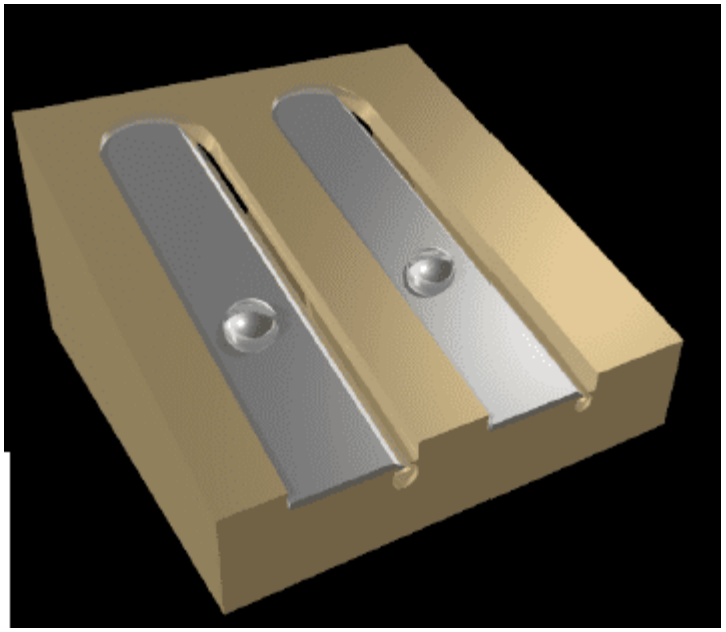
Intersection
Portion common to both objects

Volumetric modeling:
Constructive Solid Geometry (CSG)









Volumetric modeling: Octrees

➢Solids are represented by a list of voxels (volume elements). Voxels are spatial cells occupied by the solid; they are usually fixed-size cubes distributed according to a grid.

➢The main method of cell decomposition is the **Octree** which is a generalization of **Quadtrees** .

➢A **quadtree** is a representation of a 2D object based on the recursive subdivision of a square into 4 squares of the same size (quadrants)

Advantages of Octrees

•Ease of validation.

•Simplicity of access to a given point and spatial uniqueness ensured.

Disadvantages of Octrees

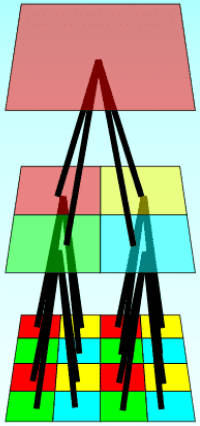
•No structure between different parts of an object.

•Expensive in terms of memory.

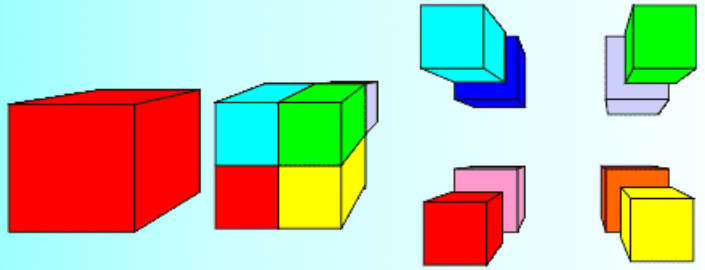
•It is not possible to represent very complex objects.

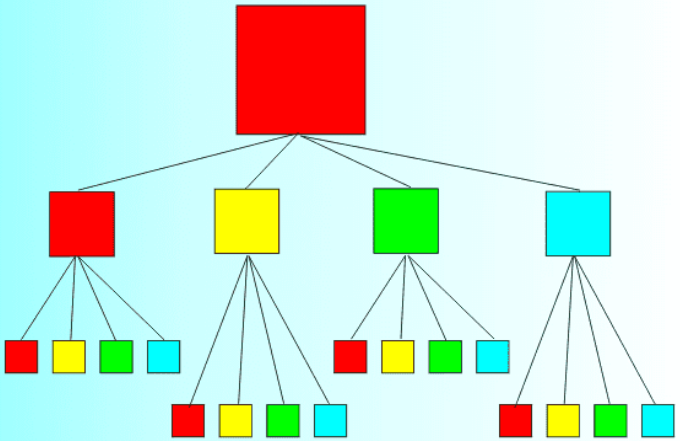
Volumetric modeling: Octrees

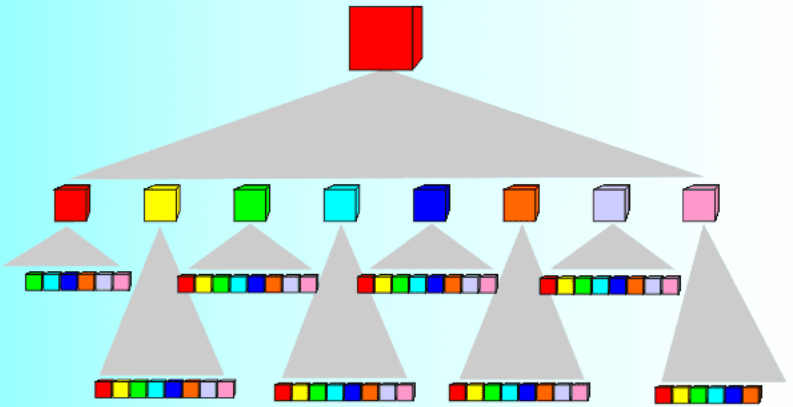
Quadtrees



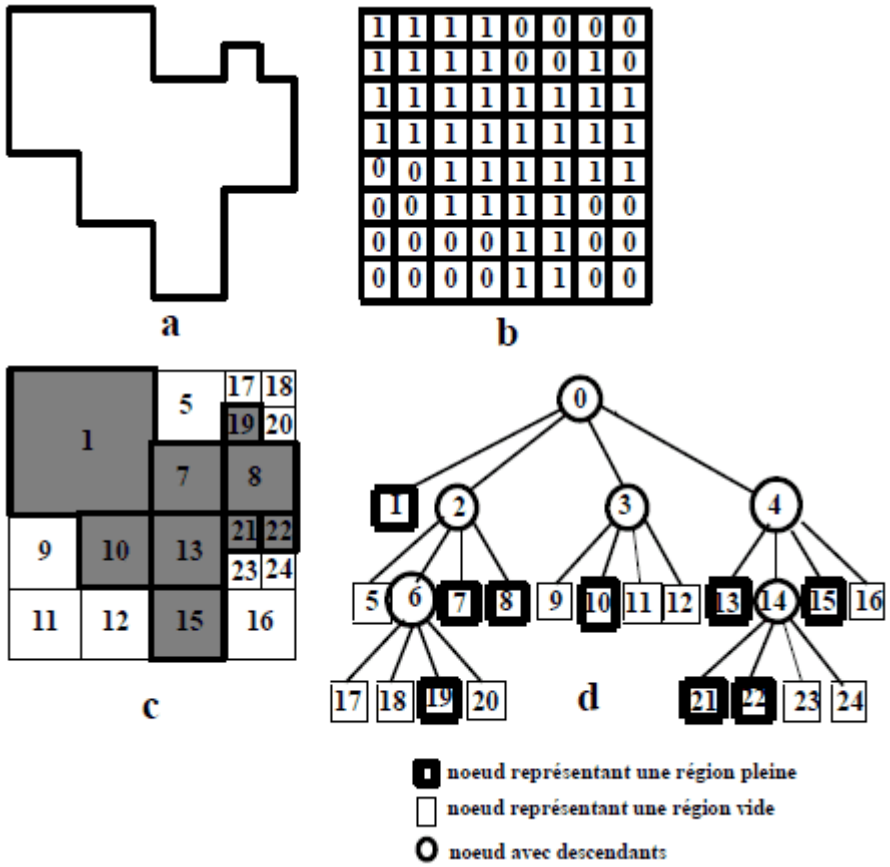
Octrees



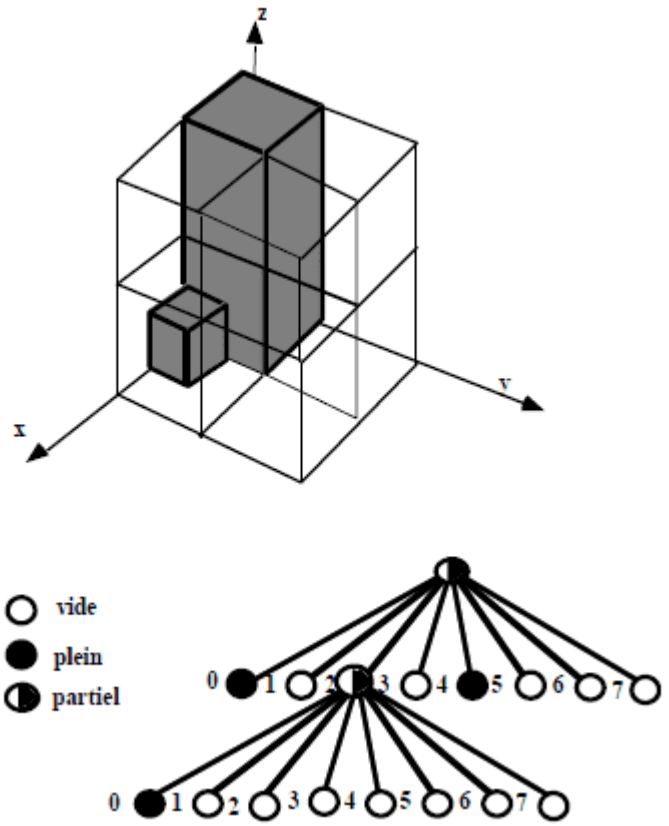




Volumetric modeling: Octrees



Example of a quadtree . a: the 2D object; **b** : the corresponding binary matrix representation **; c:** the quadrant decomposition; **d:** the quadtree representation



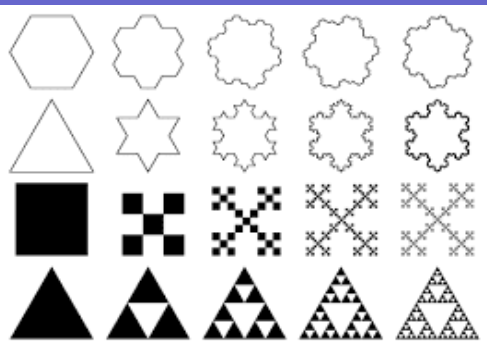
Octree representation of a 3D object

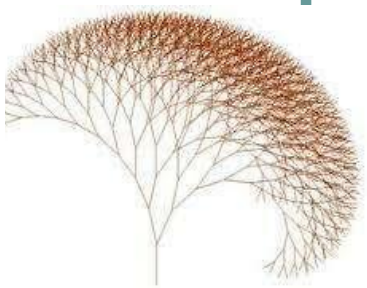
Procedural modeling:

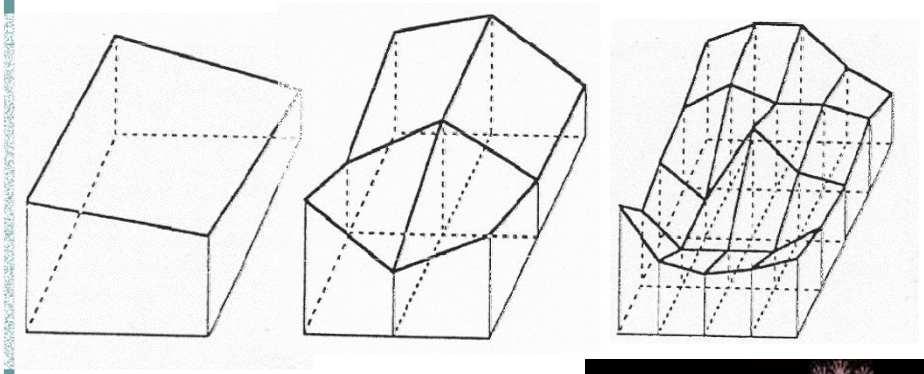
- Consists of representing complex natural objects such as trees, clouds, plants, fire, etc.
- Its objects are designed by using one of the following new models: recursive (Fractals, Graftales), deformations, Physics….etc.).

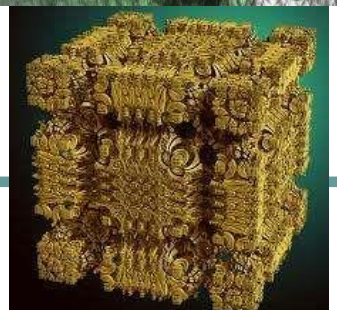
Procedural modeling:
Fractals

- Mountains fractals
  - Recursive construction of the ground
  - Model statistical
    - axiom
    - generator random





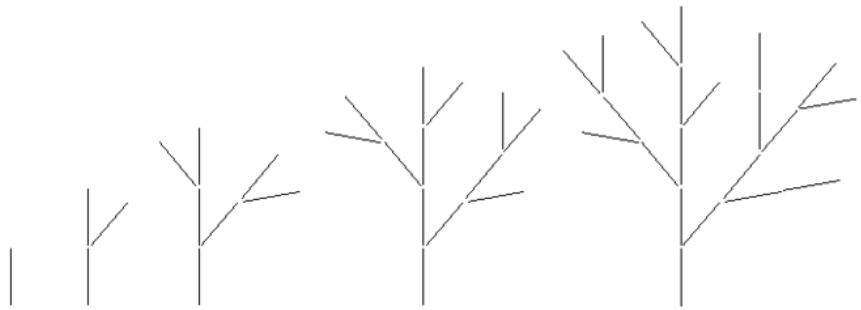


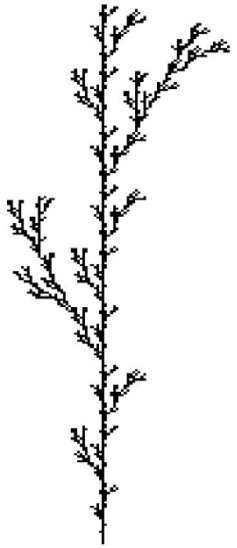


Procedural modeling:
L-systems

- Modeling of plants
  - Grammar describing the growth
    - The axiom
    - the rules of production
    - A corner
    - the number of iterations

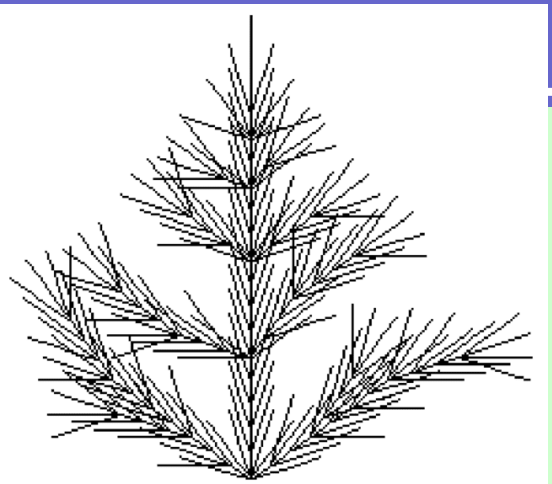
- Example:
  - axiom : F
  - ruler: F → F[+F]F [-F]F
  - $\alpha = \pi /7$

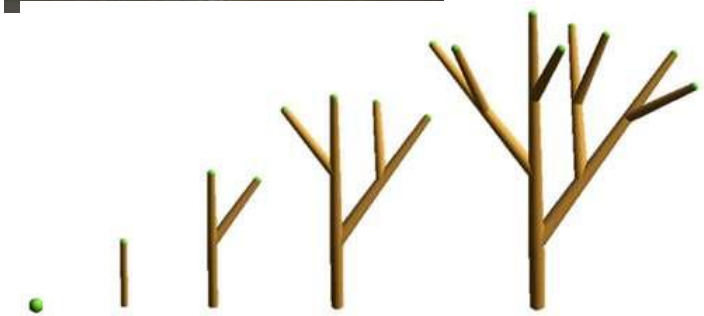




Procedural modeling:
L-systems

- axiom: SLFFF
- rules:
  - S [+++G][---G]TS
  - G + H [+G] L
  - H -G [+H]L
  - T TL
  - L [-FFF][+FFF]F
- $\alpha = \pi / 10$

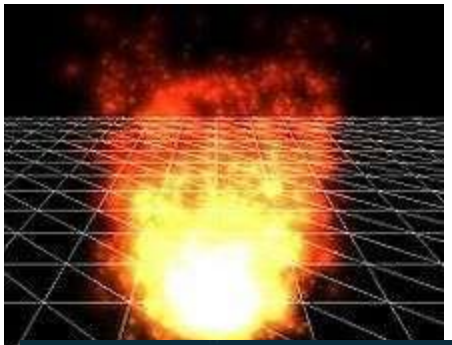


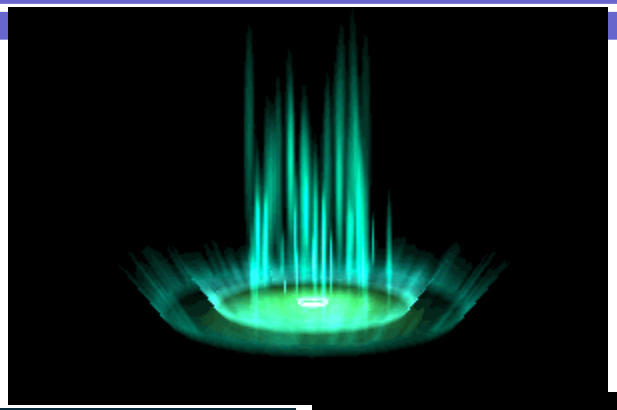


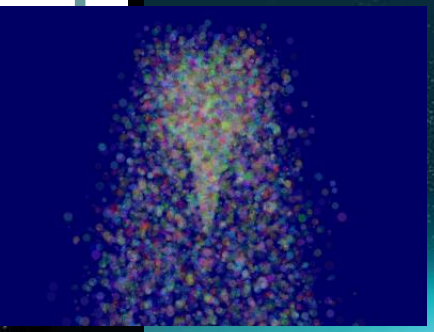
Some determinstic 3D branching plants.

Procedural modeling:

Particle







Procedural modeling:
Particle

Particle attributes: deterministic or random

- Particle set
  - Position, speed, mass , shape parameters, color, transparency
- Displacement: equations of dynamics
  - Sum of forces = mass * acceleration
- *Generator* : particle source
- Limited *lifetime*
- Very useful for:
  - Dust, smoke, sparks, flames, liquids…

Procedural modeling:
Particle

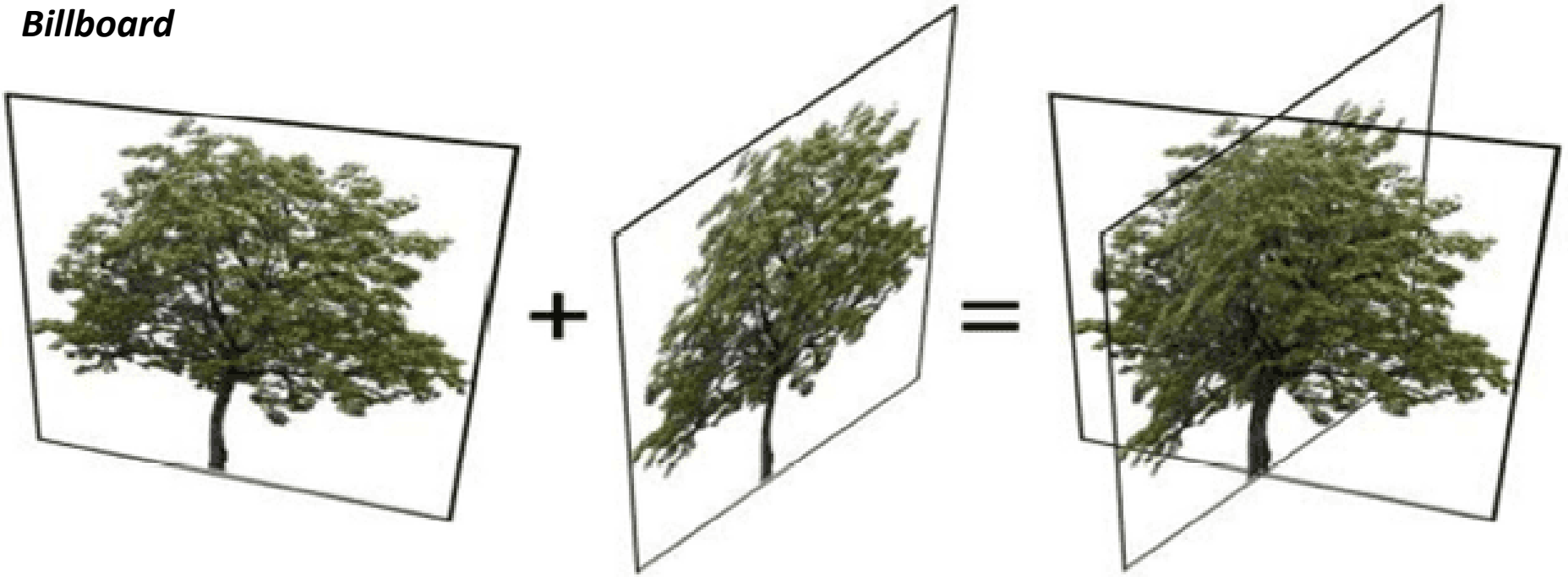
Particle life cycle :

for each **time range** do:

- 1. generate N new particles
- 2. give attributes to new particles
- 3. destroy particles whose lifetime has expired
- 4. animate the remaining particles: collision + shadow management
- 5. collisions can be destructive,
- 6. draw the remaining particles

Image-based models

Billboard



panoramic views

Geometric transformations

Coordinate system of a 3D scene

3D scene and coordinate systems

➢For the design of a 3D scene and its 3D objects, we consider the following **coordinate systems**:

- **Local coordinate system** (or **object reference point**): we design an object independently of the rest of the scene.
- **World coordinate system** (or **scene reference frame** : this is the coordinate system frame for all positioning (local reference frames and the camera will be placed directly or indirectly relative to World).
- **Camera coordinate system** (or **view coordinate system**; noted **Eye** in the following): it is in this coordinate system that we define the visualization volume (i.e. placement of the screen and definition of the projection parameters).

Coordinate system of a 3D scene

3D scene and coordinate system (continued)

- We will place the coordinate systems relative to each other with changes of coordinate systems (generally by translations, rotations, changes of scale).
- The positions (x , y , z) of the vertices of 3D objects are given in the Local coordinate system of the object they define, and the object is placed in the scene by moving its local coordinate systems (and not directly/explicitly its points.)

Coordinate system of a 3D scene

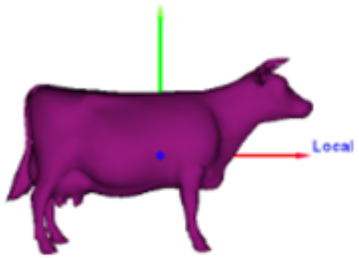
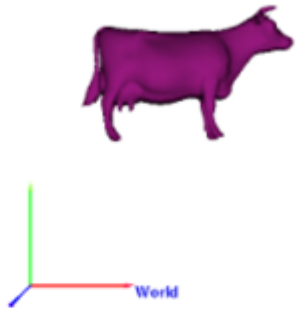
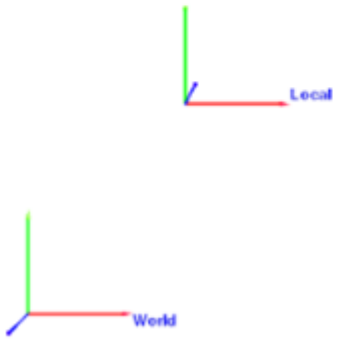
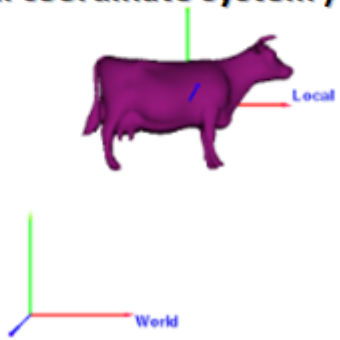
3D scene and coordinate system (continued)

- Three main coordinate systems :
  - ✓local: 3D model
  - ✓world:user
  - ✓camera: projection

- $V_{local} \rightarrow V_{monde} \rightarrow V_{camera}$

Coordinate system of a 3D scene

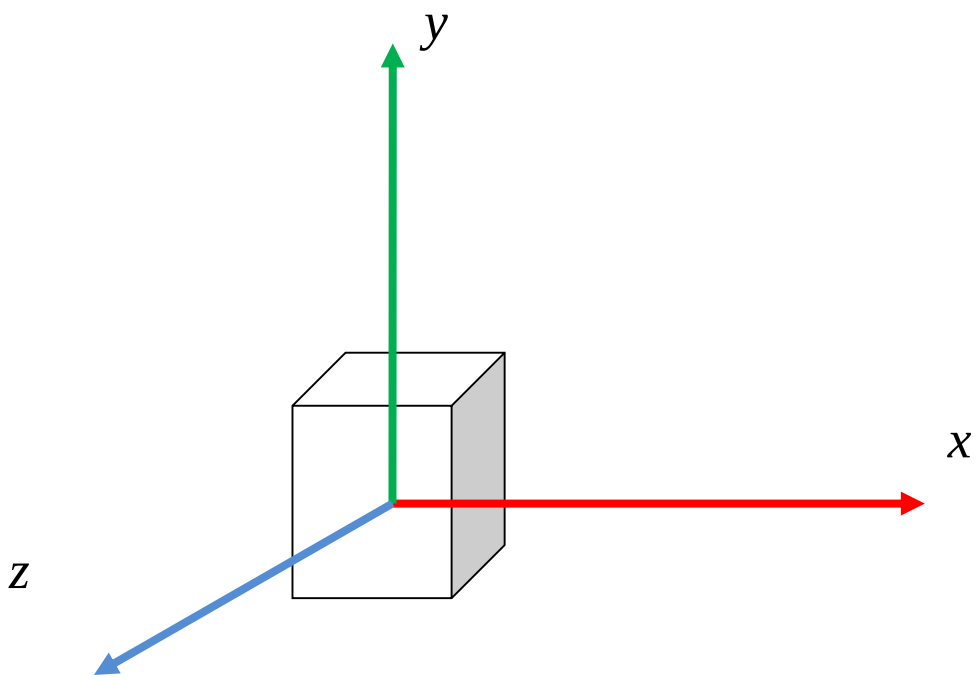
Example

1)We have an object defined in its coordinate sys tem 	2) We wish 
3) We place Local in relation to World 	4) We obtain (points always expressed in Local coordinate system) 

Model, view and projection matrices

The model matrix

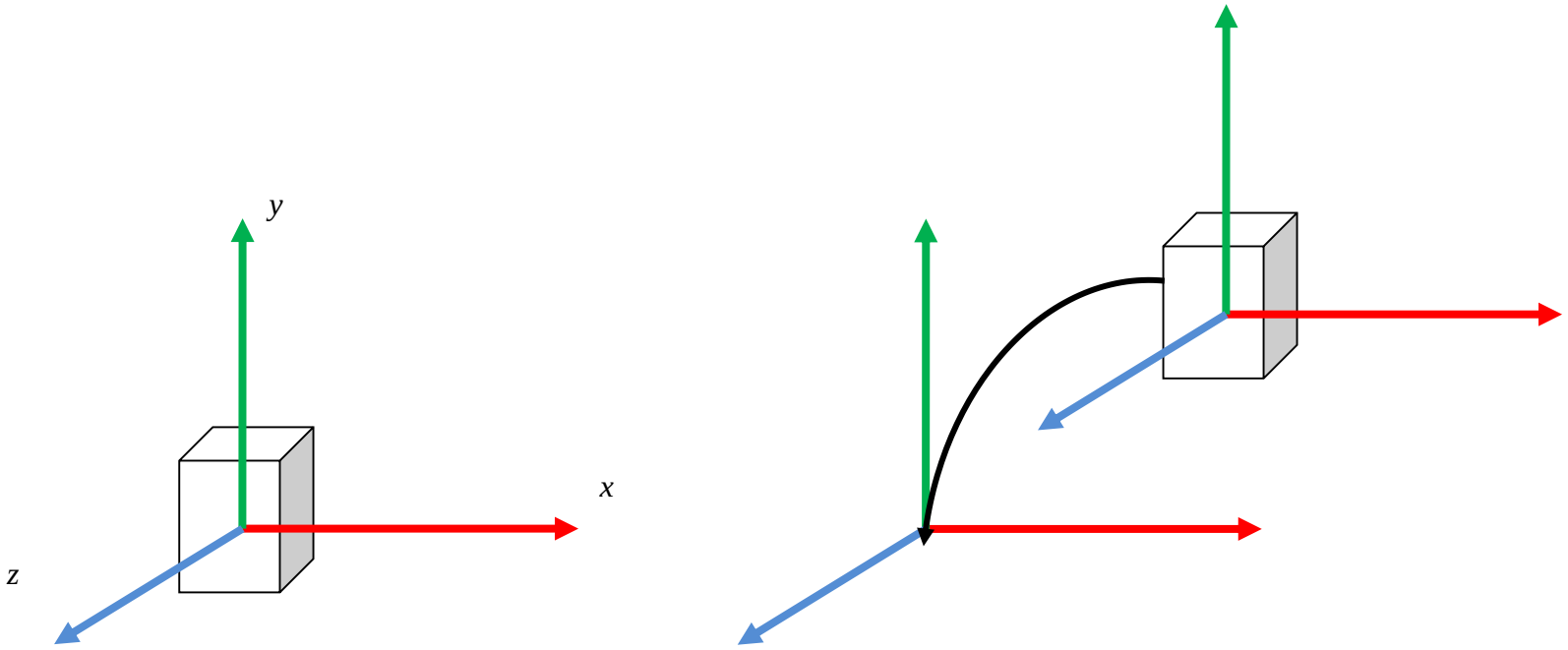
The model is defined by a set of vertices. The X, Y, Z coordinates of these vertices are defined relative to the center of the object.



Model, view and projection matrices

The Model Matrix (continued)

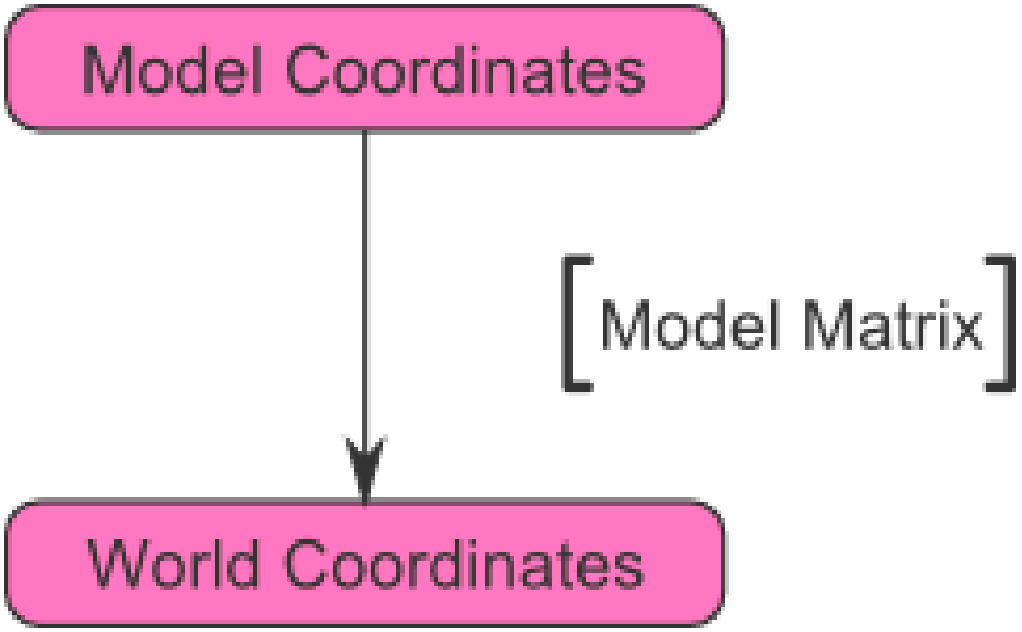
Our vertices are now in the world coordinate system. This is the meaning of the black arrow in the image below: we have moved from the model space (all vertices are defined relative to the model center) to the world space (all vertices are defined relative to the world center).



Model, view and projection matrices

The Model Matrix (continued)

We can summarize this with the following diagram:

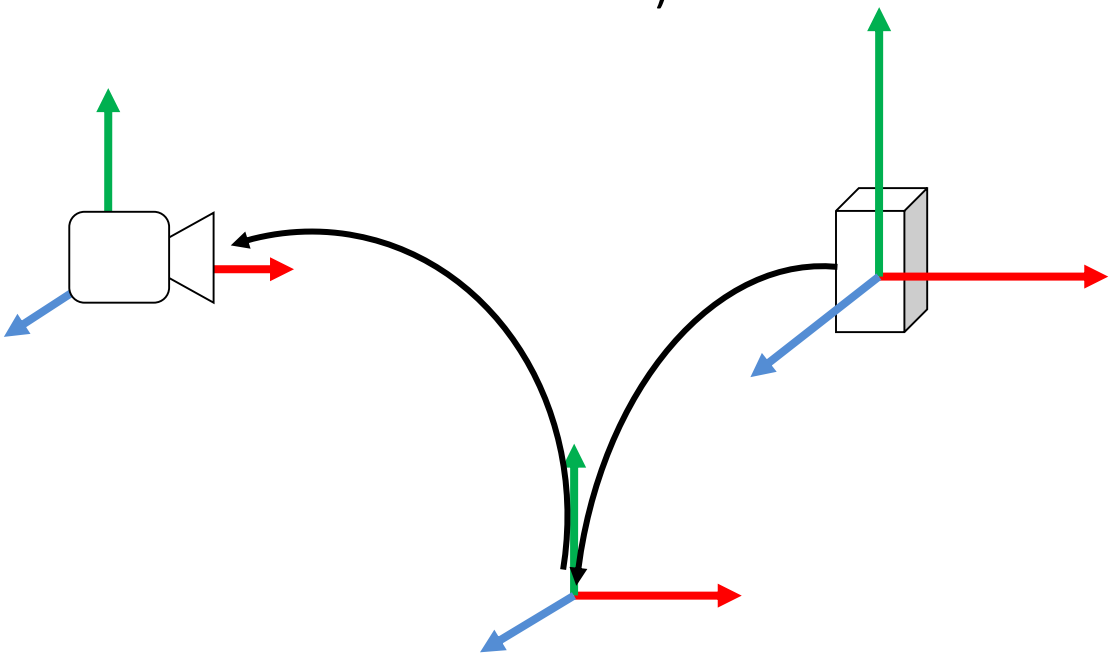


Model, view and projection matrices

The view matrix

At first, our camera is at the origin in the world frame. In order to move the world, you simply introduce a new matrix. Example: move the camera t(-3,0,0) .

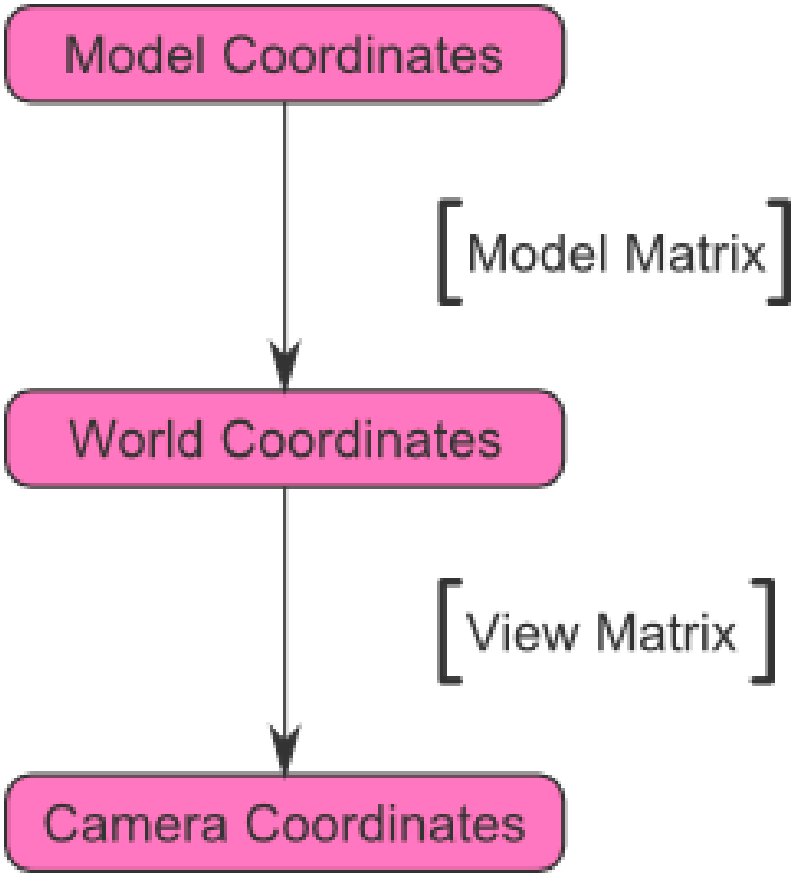
We moved from the world coordinate system (all vertices are defined relative to the center of the world) to the camera coordinate system(all vertices are defined relative to the camera).



Model, view and projection matrices

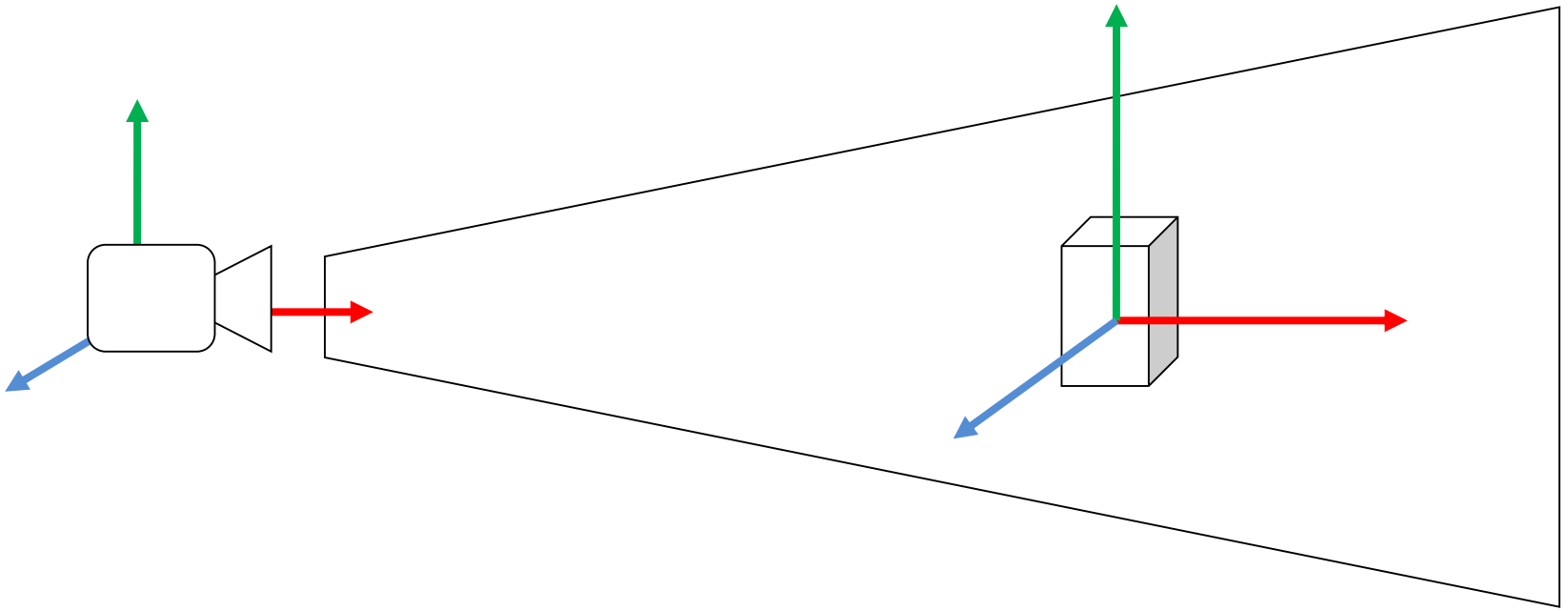
The view matrix

Here is the obligatory diagram:



Model, view and projection matrices

The projection matrix



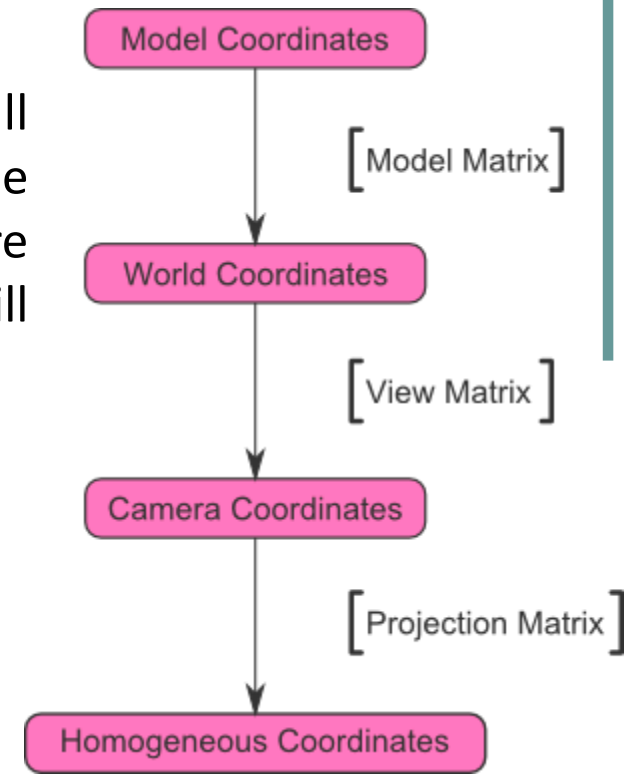
Model, view and projection matrices

The projection matrix

And here is the final diagram:

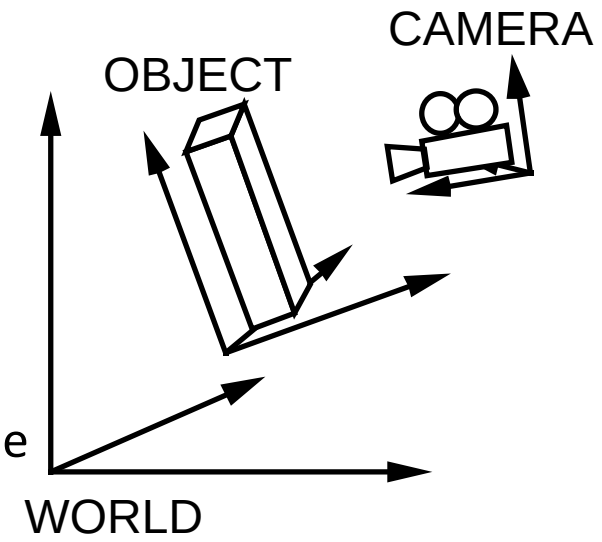
We moved from the camera coordinate system (all vertices are defined relative to the camera) to the homogeneous coordinate system (all vertices are defined in a small cube). Everything in the cube will appear on the screen.

And here is the final diagram:



Transformations geometric

- World coordinate system
  - placement of objects
  - camera placement (point of view)
- Transformations
  - translation, rotation, scaling
  - build complex scenes by positioning simple objects (successive transformations)
  - transform object coordinates into world coordinates
- Projection
  - pyramid view
  - perspective projection
  - **Clipping :** consists of not calculating objects outside the cone of vision of a scene,



Transformations geometric

- **Model:** A set of objects organized to represent a scene to be displayed. Each model is a set of points (x,y,z).
- Vector representation of points
  - Points attached to graphic primitives
  - Vertices, centers, volume data…
- Transformations on this data
  - **Transformation:** operation on vertices
  - Translation, rotation, change of scale…
  - **Projections** :
    - Perspective, parallel…
  - **Unified notation?**

Point transformations
Basic transformations

Once we have constructed graphical objects, we generally want to manipulate them, that is, change their attributes.

The attributes of a graphic object are:

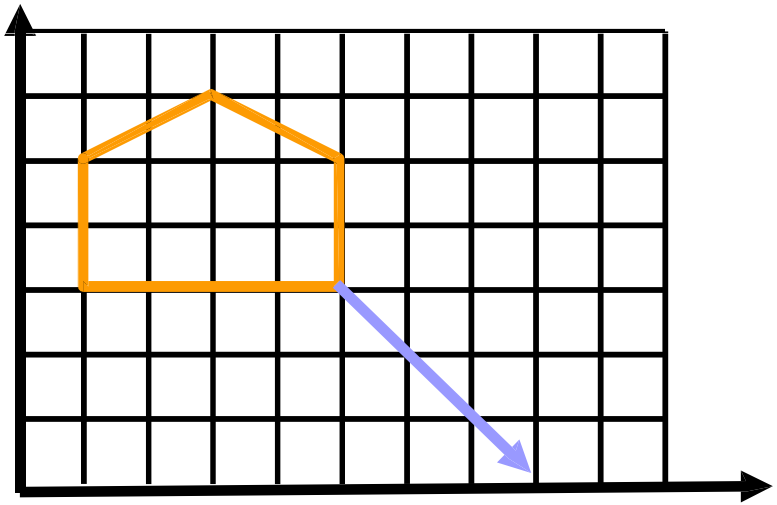
- the position
- orientation
- the size
- the form
- the color
- transparency
- the texture

In 2

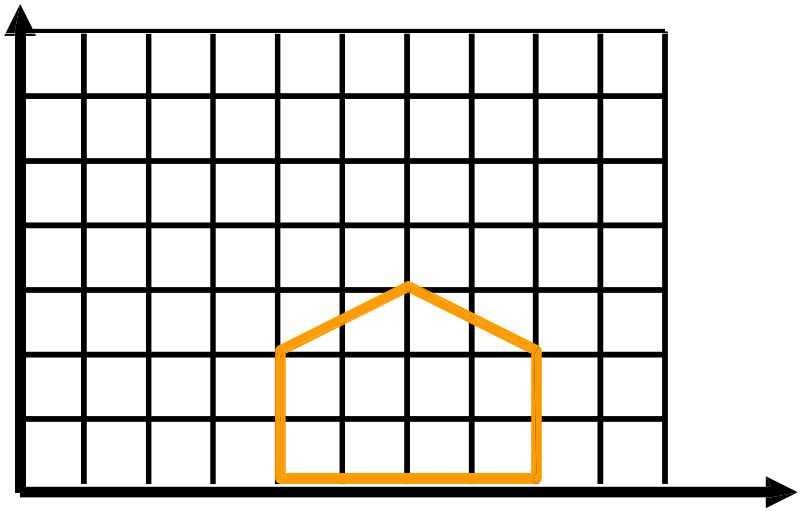
- We begin in 2D
  - Easier to represent
- Each point East transformed :
  - x' = f(x , y)
  - y' = g(x,y)
- How to represent transformation?

Translations

- Modification simple :
  - $x' = x+t_x$
  - $y' = y+t_y$



Before



After

Vector notation

- We uses vectors for the representation
  - It is more simple

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

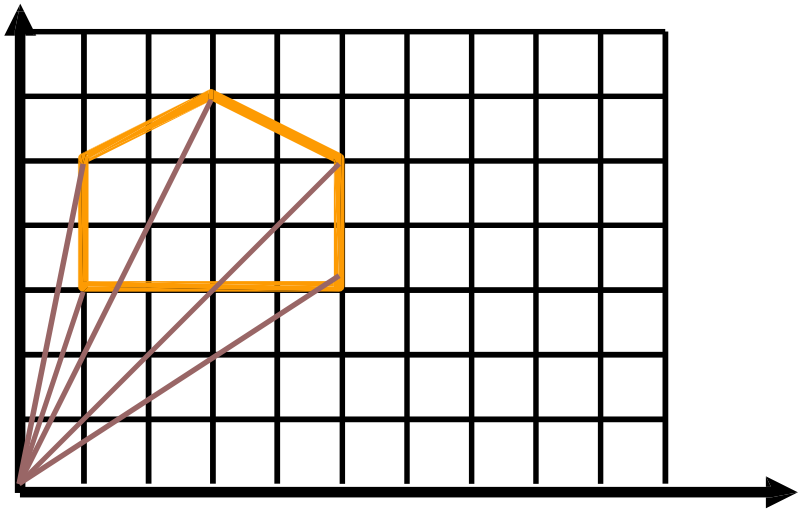
- A point is a vector

- A translation is a vector sum:

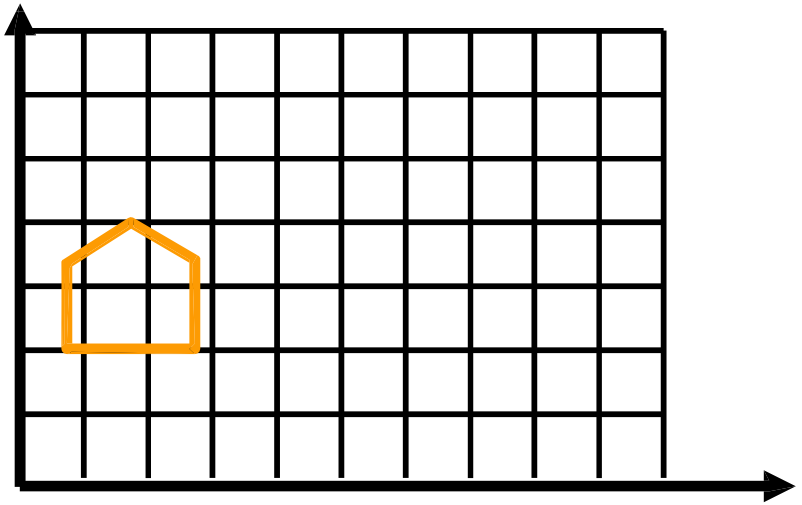
$$P' = P + T$$

Change of scale

- The coordinates are multiplied by the scale change factor:
  - $x' = s_x x$
  - $y' = s_y y$



Before



After

Notation matrix

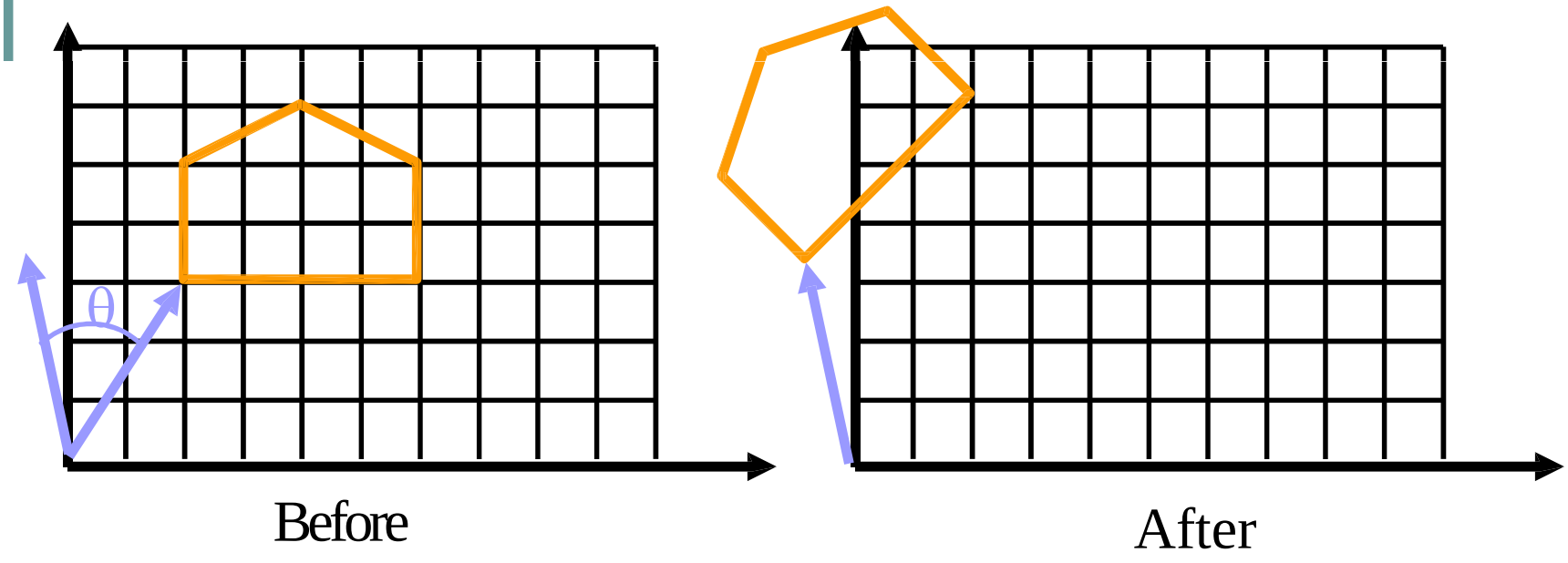
- This is a matrix multiplication:

  P’ = SP

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotation

- Rotation in 2D:
  - $x' = \cos\theta\, x - \sin\theta\, y$
  - $y' = \sin\theta\, x + \cos\theta\, y$



Notation matrix

- Rotation = matrix multiplication:

$$P' = R\ P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Unification

- Notation simple , concise
- But not Really unified
  - Addition or else multiplication
  - **How to concatenate several transformations ?**

- **We want a unique notation**
  - Which also allows us to note the combinations of transformations
  - How to do it?

Homogeneous coordinates

Idea working in a space of dimension (n+1)
– called projective space
– represent affine geometric transformations by matrices

- Very powerful geometric tool:
  - Used everywhere in Computer Graphics
  - also projective geometry
- We add a third coordinate, W
- A 2D point becomes a 3-coordinate vector:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Transition from usual coordinates to homogeneous coordinates

(x, y, z) —> [x, y, z, 1]

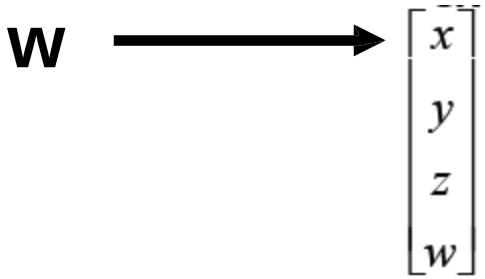
[x, y, z, w] —> [x/w, y/w, z/w], if w ≠ 0

Homogeneous coordinates

- Two points are equals if and only if :
  - x ' /w' = x/w And y ' /w'= y/w
- w=0: points " has infinity " then the vector (x, y, z,0) is a direction .
  - Very useful for projections, and for some splines .
- w = 1, then the vector (x, y, z, 1) is a position in space .

And in 3 dimensions?

- We introduces a fourth coordinate,



  - Two vectors are equals if :
    x/w = x ' /w ' , y/w = y ' /w' And z/w=z'/w'

- All of transformations are of matrix 4x4

Translations in homogeneous c.

In 2D

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$\begin{cases} \frac{x'}{w'} = \frac{x}{w} + t_x \\ \frac{y'}{w'} = \frac{y}{w} + t_y \end{cases}$$

$$\begin{cases} x' = x + wt_x \\ y' = y + wt_y \\ w' = w \end{cases}$$

Change of scale in homogeneous c.

In 2D

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$\begin{cases} x' = s_x x \\ y' = s_y y \\ w' = w \end{cases}$$

$$\begin{cases} \frac{x'}{w'} = s_x \frac{x}{w} \\ \frac{y'}{w'} = s_y \frac{y}{w} \end{cases}$$

Rotation in homogeneous c.

In 2D

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

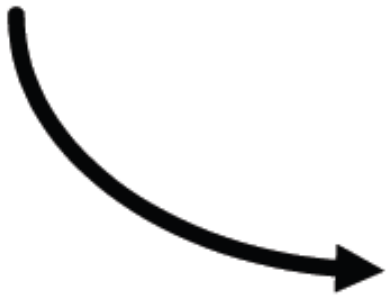
$$\begin{cases} \frac{x'}{w'} = \cos\theta \frac{x}{w} - \sin\theta \frac{y}{w} \\ \frac{y'}{w'} = \sin\theta \frac{x}{w} + \cos\theta \frac{y}{w} \end{cases}$$

$$\begin{cases} x' = \cos\theta x - \sin\theta y \\ y' = \sin\theta x + \cos\theta y \\ w' = w \end{cases}$$

Translations in homogeneous c.

In 3D

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

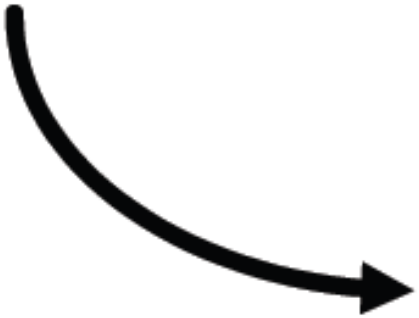


$$\begin{cases} x' = x + wt_x \\ y' = y + wt_y \\ z' = z + wt_z \\ w' = w \end{cases}$$

Change of scale in homogeneous c.

In 3D

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\begin{cases} x' = s_x x \\ y' = s_y y \\ z' = s_z z \\ w' = w \end{cases}$$

Rotations in 3D

- Rotation : A axis and an angle
- The matrix depends on the axis and the angle
- Direct expression possible, starting from the axis and the angle, and some vector products
- Do by the graphic bookstore
  - glRotatef(angle, x, y, z) :

Rotation in homogeneous c.

In 3D : Rotation around the x axis

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The x axis is not modified

$$R_x(\tfrac{\pi}{2}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation in homogeneous c.

In 3D : Rotation around the y axis

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The y axis is not modified

$$R_y(\tfrac{\pi}{2}) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation in homogeneous c.

In 3D : Rotation around the z axis

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The z axis is not modified

$$R_z(\tfrac{\pi}{2}) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

All 3D transformations

- Any 3D transformation is expressed as a combination of translations, rotations, changes of scale
  - And so as a matrix in homogeneous coordinates

- Provided by the graphics library :
  - glTranslatef(x, y, z);
  - glRotatef(angle, x, y, z);
  - glScalef(x, y, z);

Composition transformations

- Just multiply the matrices:
  - composition of a rotation and a translation:

$$\mathbf{M = RT}$$

(in OpenGL you have to invert **M = TR**)

- All 2D transformations can be expressed as coordinate homogeneous matrices.
  - Notation very general
- Rotation around a point Q:
  - Translate **Q** to the origin ($\mathbf{T_Q}$),
  - Rotation around of the origin ($\mathbf{R_\Theta}$)
  - Translate back to Q ($\mathbf{-T_Q}$) .

$$\longrightarrow \quad P' = (-T_Q)\, R_\Theta\, T_Q\, P$$

3D Transformations (continued)

- We can do the same:
  - glLoadIdentity();
  - glLoadMatrixf(pm);
  - glMultMatrixf(pm);
- Stack of transformations:
  - glPushMatrix(); :
  - glPopMatrix();

Rasterization

Rasterization

– For each primitive P_i, find the rays intersecting P_i

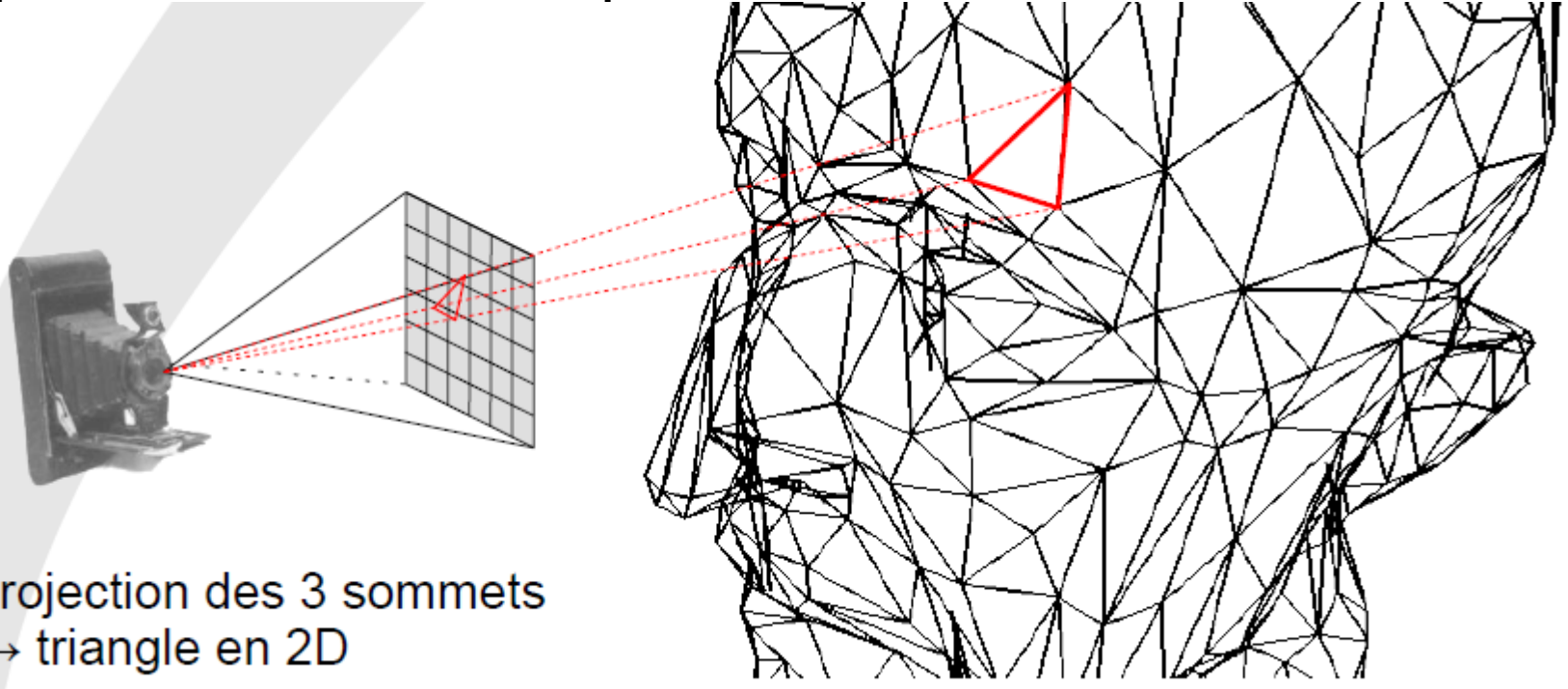
– Two-step rendering

• projection of primitives onto the screen (*forward projection)*

• discretization (conversion of 2D primitives into pixels)

– scene = set of " rasterizable " primitives

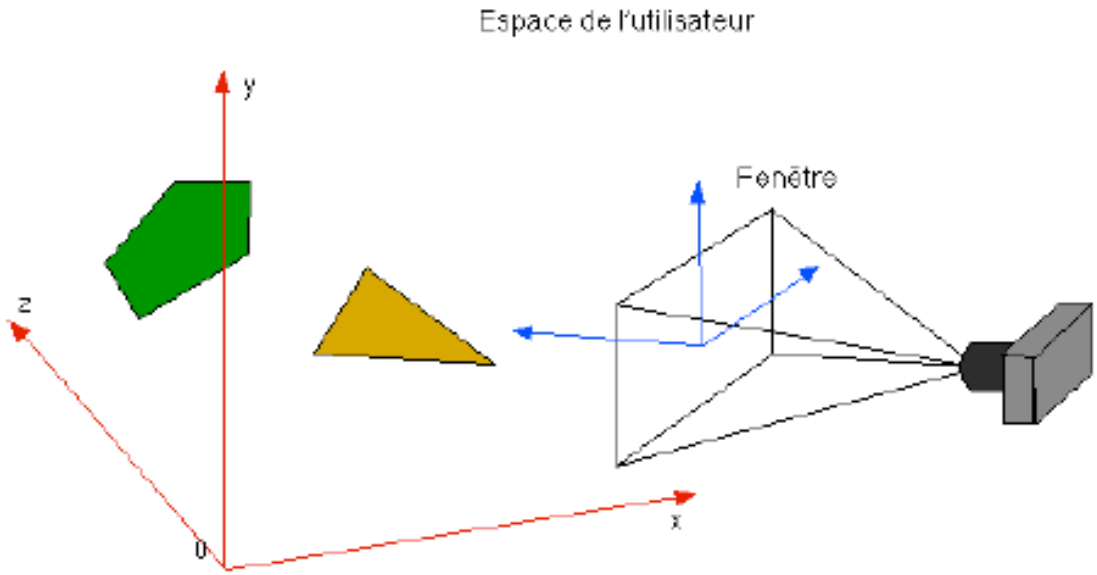
→ problem: elimination of hidden parts

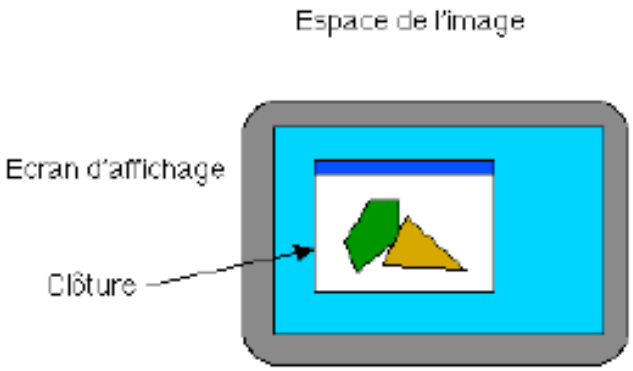


projection des 3 sommets
→ triangle en 2D

Moving from scene to screen

User space and image space

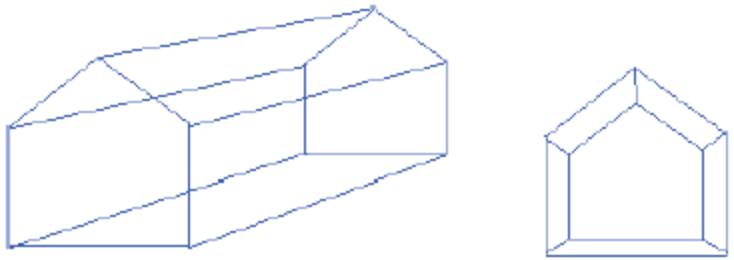




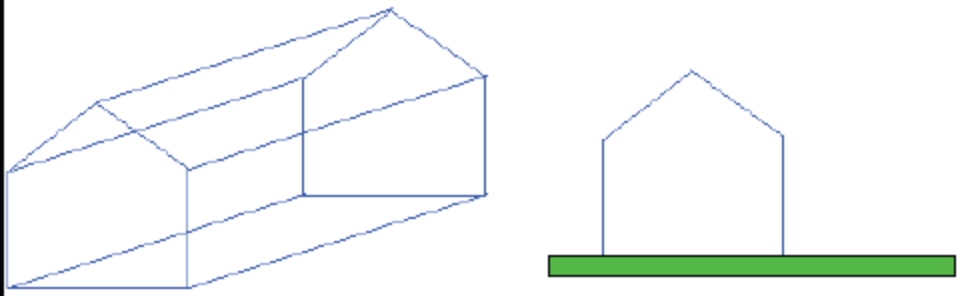
Moving from scene to screen

Transition from 3D space to 2D space

- The mathematical operation is a projection.
- Two types of projection are commonly used:
  - projection with perspective (more realistic)



  - orthogonal projection (sometimes more convenient when creating and in areas such as CAD)



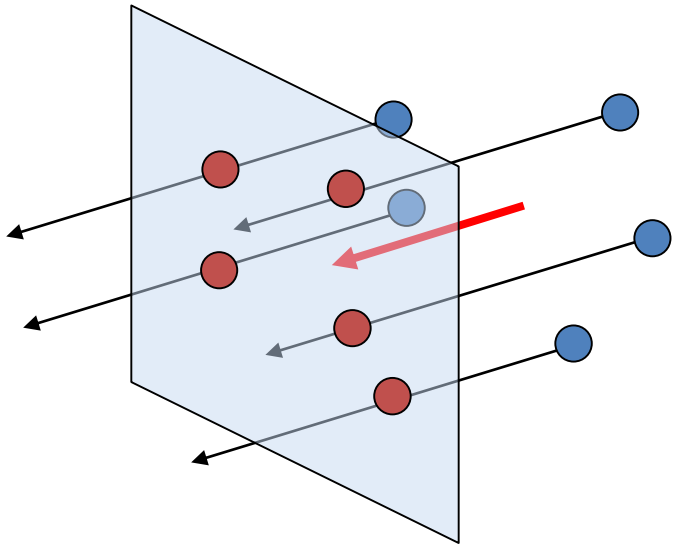
Projections

- Projection reduced the domain $P_n \in \Re^n \rightarrow \Re^m$ où $m < n$
  - typically in computer graphics , n=3 and m =2
- A projector is a segment connecting P_n to a projection center

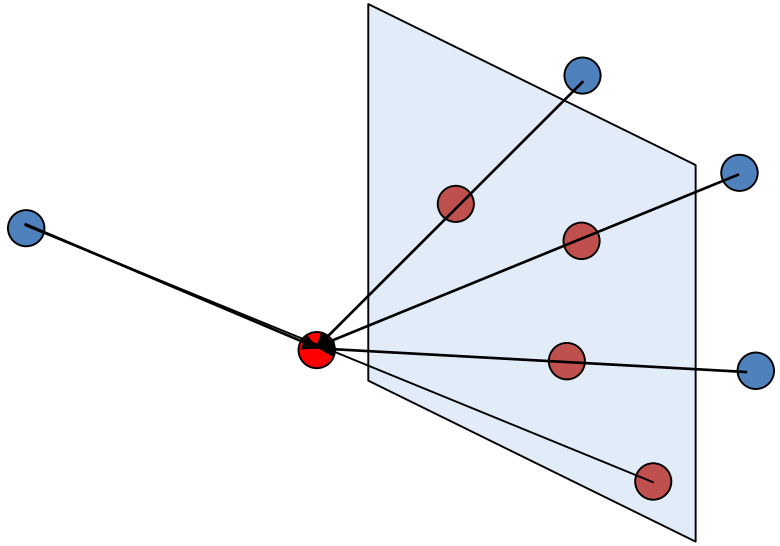
- The intersection of a projector with the projection surface corresponds to P_m
- When this surface is planar, we speak of planar projection

Planar projection

- We split the planar projections in:
  - projection parallel
  - projection perspective



Parallel



Perspective

Parallel projection

- Projection center is at infinity
- Direction of projection

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} - \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 0 \end{bmatrix}$$

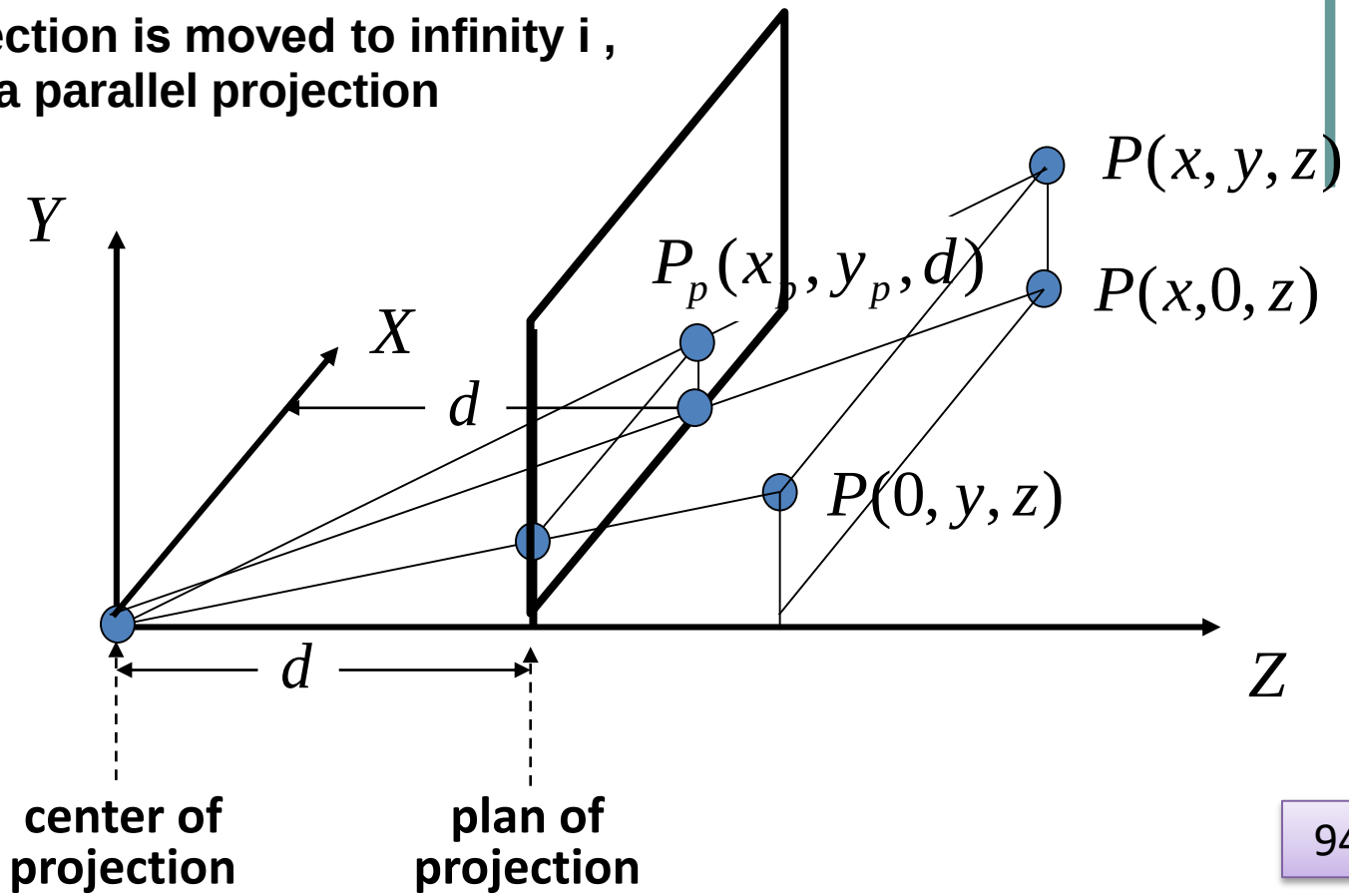
direction ou point à l'infini

- Projectors are parallel to each other
- Parallel lines in 3D remain parallel after projection

Perspective projection

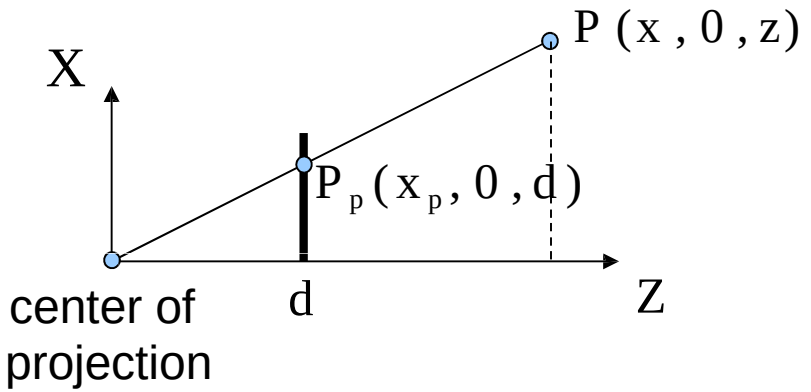
- **Projection center is at a finite distance**
- **Size of an object increases when the distance to**
- **Parallel lines in 3D are no longer parallel after screening**

- **If the center projection is moved to infinity i , we obtain a parallel projection**

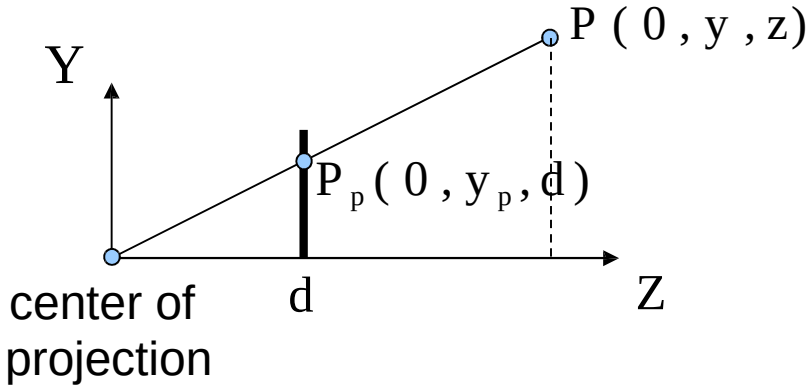


Perspective projection

- Thales 's Theorem : Triangles similar



$$\frac{x_p}{d} = \frac{x}{z} \quad : \quad x_p = \frac{x}{z/d}$$



$$\frac{y_p}{d} = \frac{y}{z} \quad : \quad y_p = \frac{y}{z/d}$$

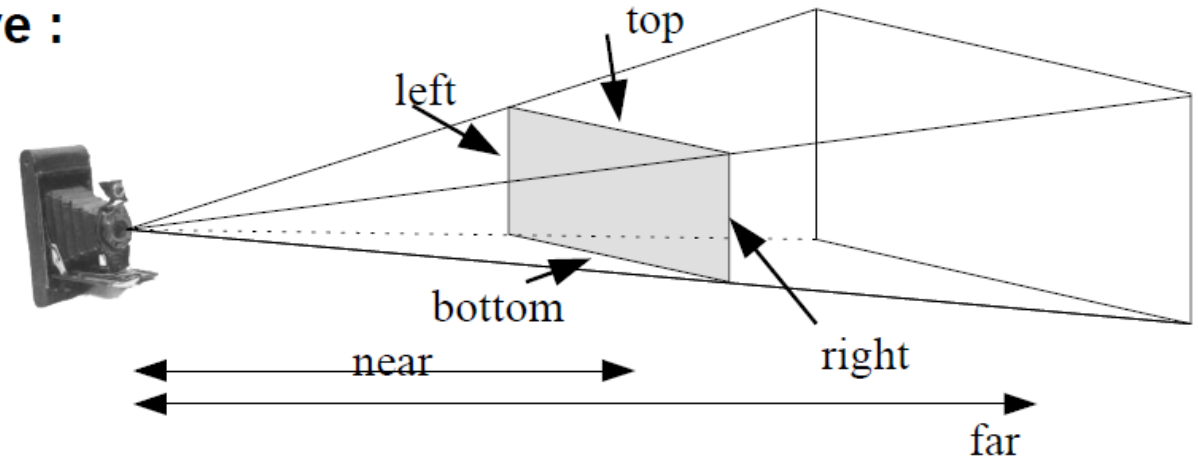
Perspective projection

- Projection on the z= 0 , with the center of projection place has z = - d :

$$\begin{bmatrix} x_p \\ y_p \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

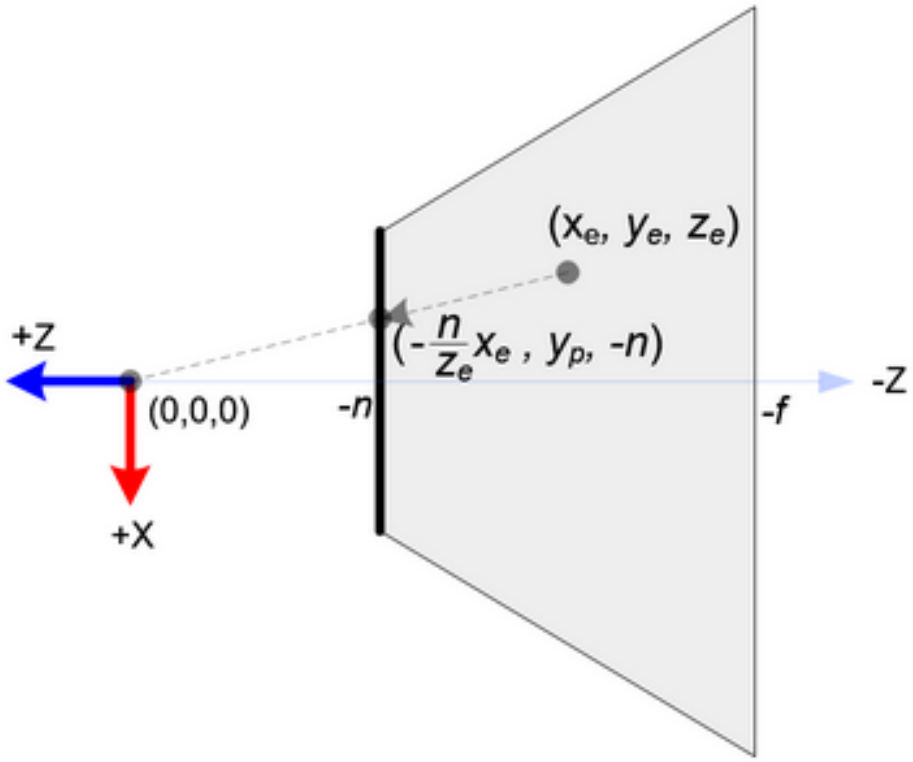
Perspective projection

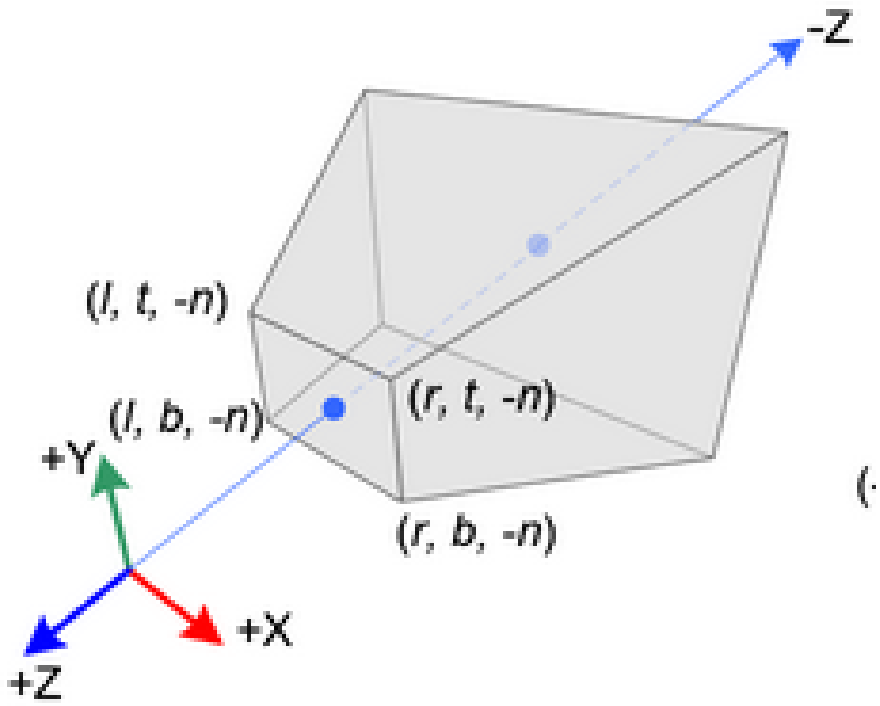
- **Projection perspective :**



$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Perspective projection



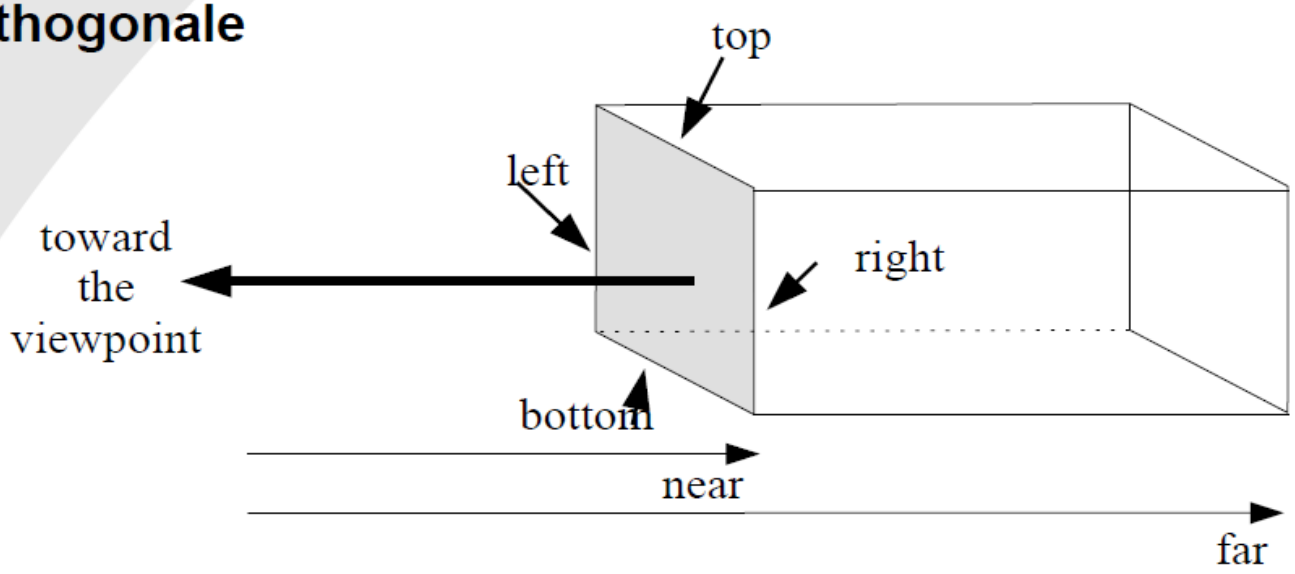


For more details, consult the site:

http://www.songho.ca/opengl/gl_projectionmatrix_mathml.html

Orthogonal projection

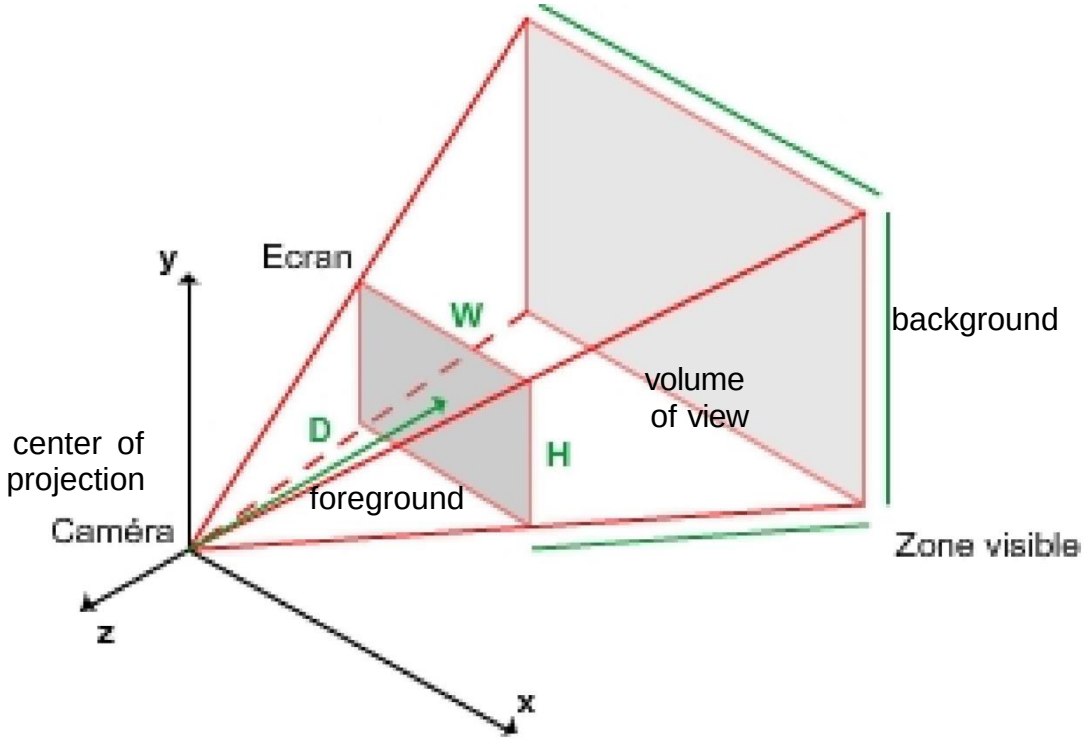
- **Projection orthogonale**



$$\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

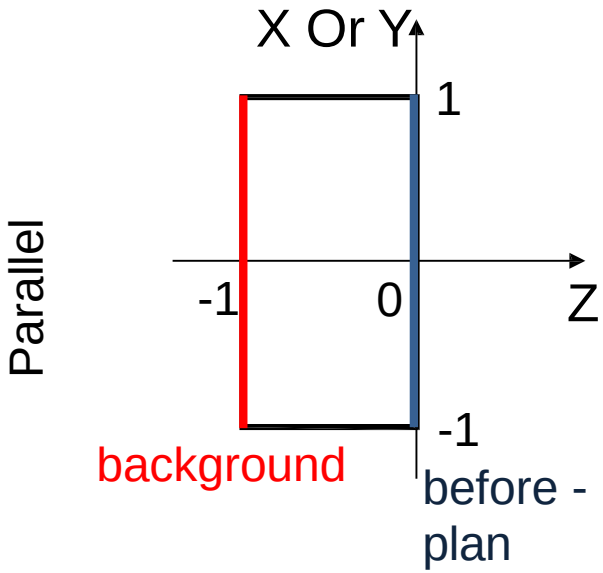
Pyramid of from view

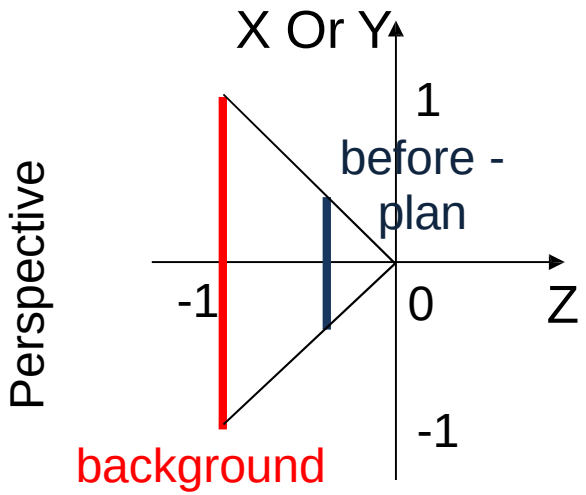
- Clipping with the six defining plans the volume from view
- Projection survivors At clipping on the window
- T ran s formations into display data coordinates



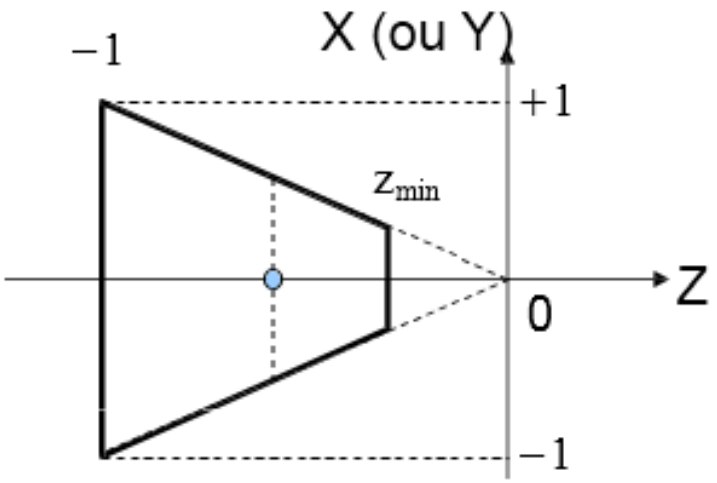
Canonical Form of the View Pyramid

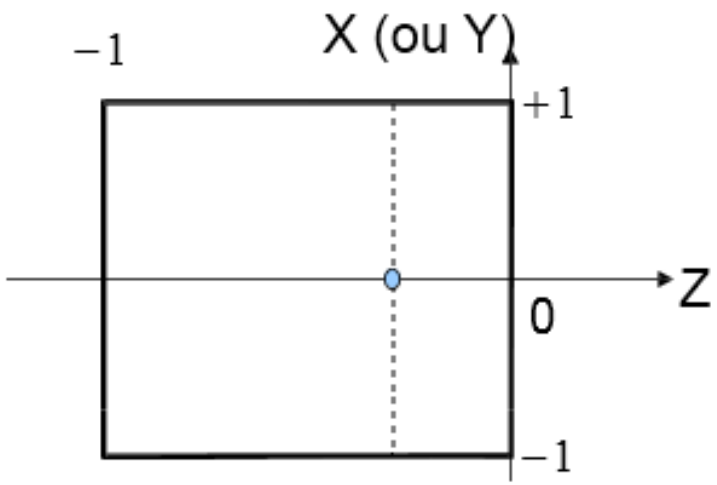
Canonical view volume : the near and far faces of the parallelepiped are parallel to *x* O *y* . The camera is at O.





Canonical Form of the View Pyramid





$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{min}} & \frac{-z_{min}}{1+z_{min}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad \text{pour } z_{min} \neq -1$$

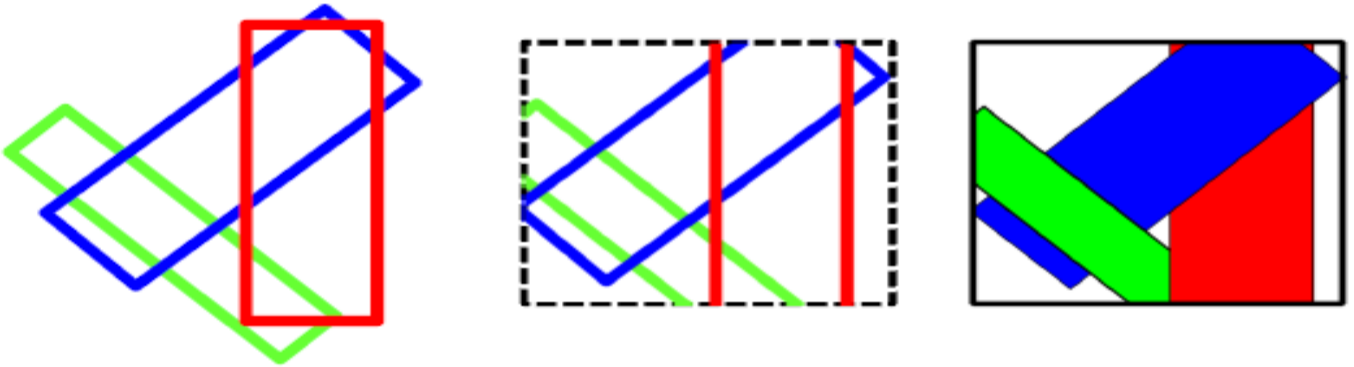
Cohen-Sutherland en 3D (6 bits)

$$x < -1 \;\; ; \;\; x > 1$$
$$y < -1 \;\; ; \;\; y > 1$$
$$z < -1 \;\; ; \;\; z > 0$$

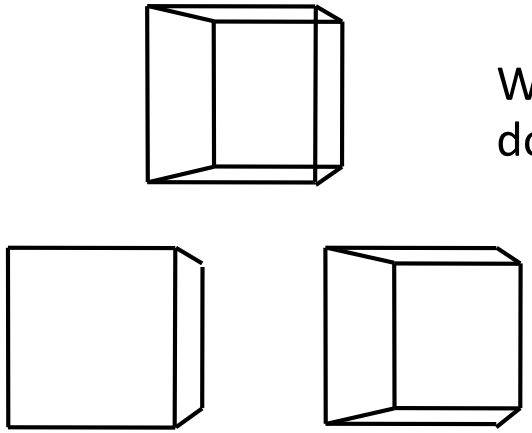
Visibility

Visibility

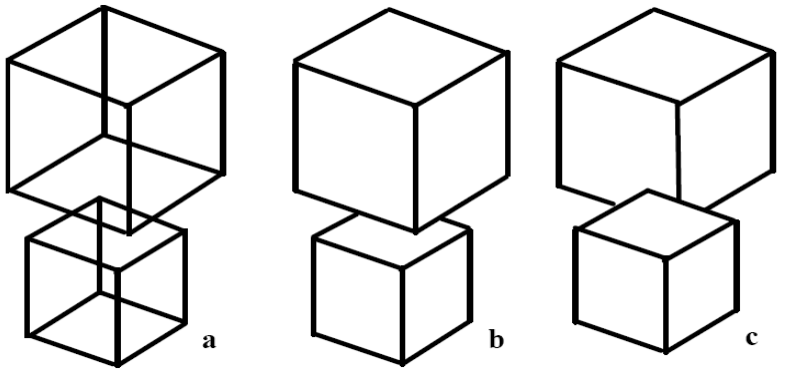
1. Determine visible surfaces or eliminate hidden surfaces
2. To create an image, we must first perform the projection and clipping
3. For a scene consisting of opaque, non-reflective objects, we must then determine what is in front for each element of the image.
4. Finally, we need to display the image element with the correct color



Eliminating Hidden Parts
Why eliminate hidden parts?



What view of the object do we want to give?



- Determine the lines, edges, surfaces, volumes, visible to an observer

105

Elimination of hidden parts
Problem position

Hidden part elimination methods aim to establish an order (called priority order) between the different elements (face, edge, vertex, etc.) of the space to be visualized according to their depths and this for each pixel.

The difficulty of solving hidden part elimination problems is related to the complexity of the scene.

Existing hidden part elimination methods fall into two categories depending on the coordinate system in which the calculations are performed:

- Algorithm in object space.
- Algorithm in image space.

106

Algorithms in image space

The processing is performed on the screen coordinates,

i.e. the calculation of hidden parts is only performed after the visualization transformation (projection of the scene from 3D space to the 2D plane).

Determines for each image element (of ***p*** pixels), the object (of ***n*** objects) which is closest to the projection center while being in the viewing volume. The computation time is quite high.

O(***n p***) : **n** * **p** operations

Ex: 100 x (640x480) = 30,720,000 operations

It is better to choose an algorithm working in object space, but solving the problem in image space is more efficient.

107

Algorithms in image space

- Accuracy depends on image resolution
- Visibility must be recalculated every time the resolution and/or window positioning changes
+ Algorithms and their structures are usually simpler

Ex: depth buffer, line scan, recursive image subdivision, ray tracing

Algorithms in object space

- In this category, the problem is solved geometrically in the three-dimensional coordinate system without considering the display on a finite resolution device.
- In this case the calculation time increases with the square of the number of facets.
- Principle: Determine which of each object is in front of the other given the projection direction
- If we have **n** objects, the algorithms are in O(n^2)

Ex: 100 x 100 = 10,000 operations

Algorithms in object space

```
for (each graphics primitive P in the scene)
 {
if (no primitive hides P)
   {
display P in the correct color;
   }
 }
```

- Accuracy depends on the resolution of the objects.
- Visibility calculation is independent of image resolution.
- Algorithms and their structures are usually quite complex.

Ex : painter's algorithm, octree , BSP trees

Backface Object Space Algorithms culling

The backface Culling (removal of back faces) is based on a simple idea.

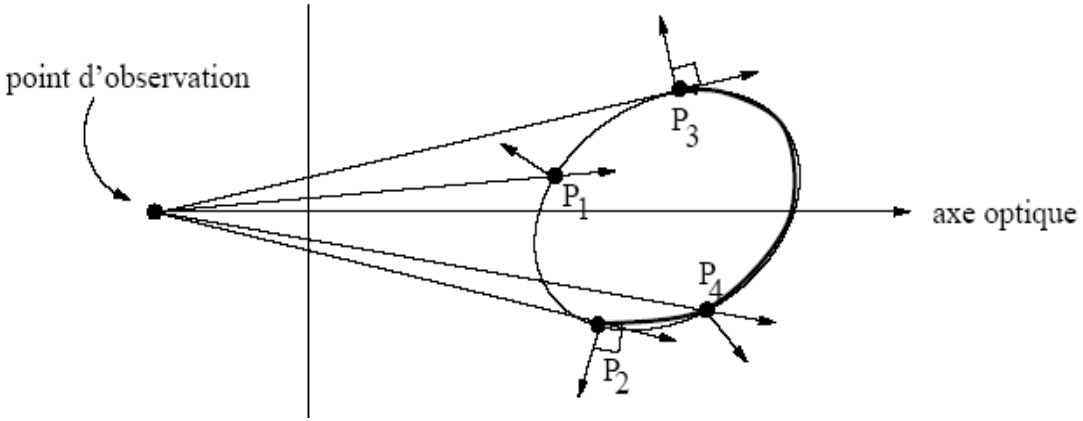
If we know the normals to the surface of the observed object,

it is then possible to eliminate the parts of the surface for which the normal points in the direction opposite to the observation point.

These parts are in fact hidden by the object itself.

111

Backface Object Space Algorithms culling



P1 *is visible,* P2 *and* P3 *are at the limit of visibility,* P4 *is not visible.*
For the points in the figure above, we can easily verify that:

N1. V1 < 0; N2. V2 = N3. V3 = 0; N4. V4 > 0;
where Vi is the direction of the line of sight at point Pi.

We deduce that the parts for which N . V > 0 are not visible and can be eliminated.
for a single convex object, the elimination of hidden parts boils down to backface culling.

112

Algorithms in image space
Depth buffer (z-buffer)

Initialize all pixels

rgb (*x,y*) = background color

z (*x,y*) = background distance

For each object in the scene to be represented

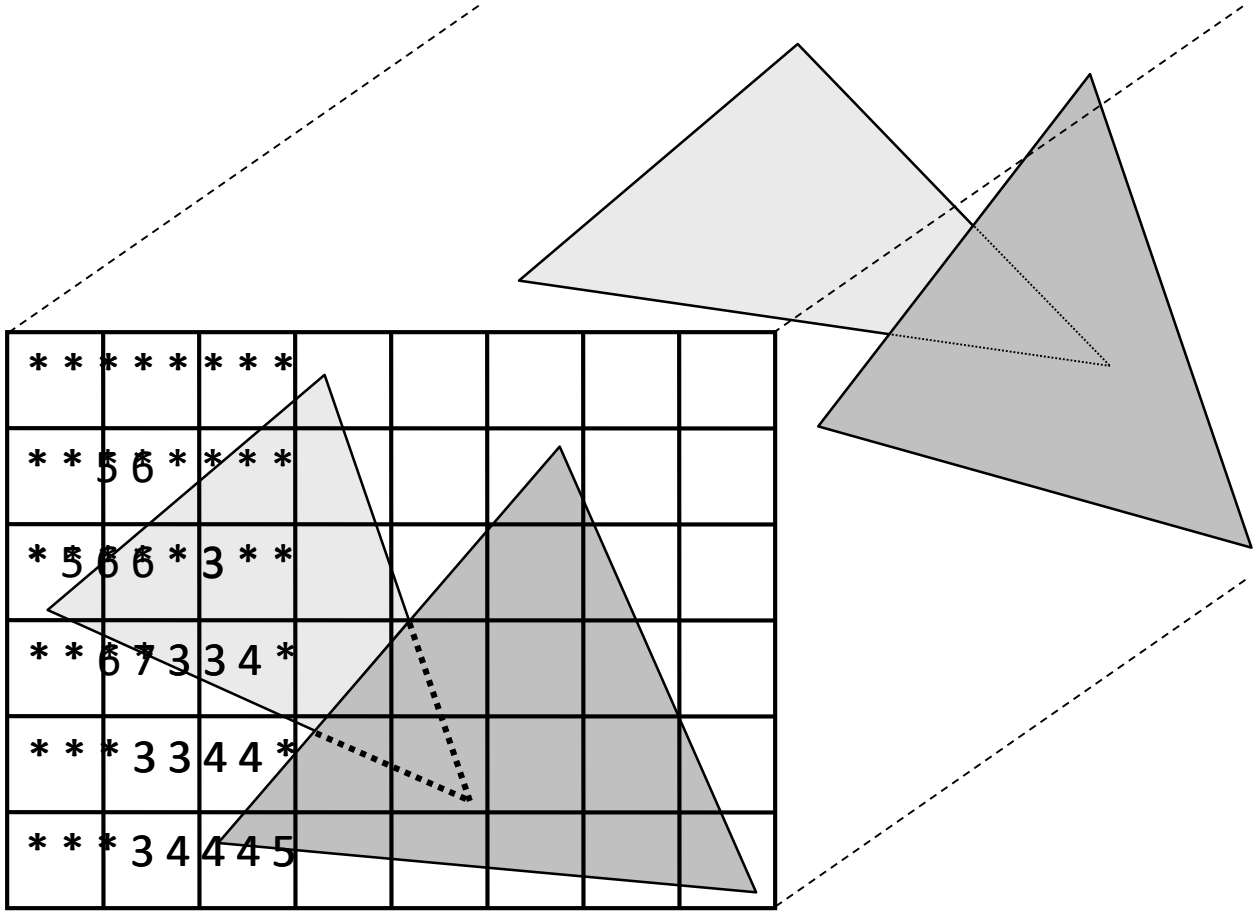
For each pixel (*x,y*) covered by the object

If object-distance (*x,y*) < *z* (*x,y*)

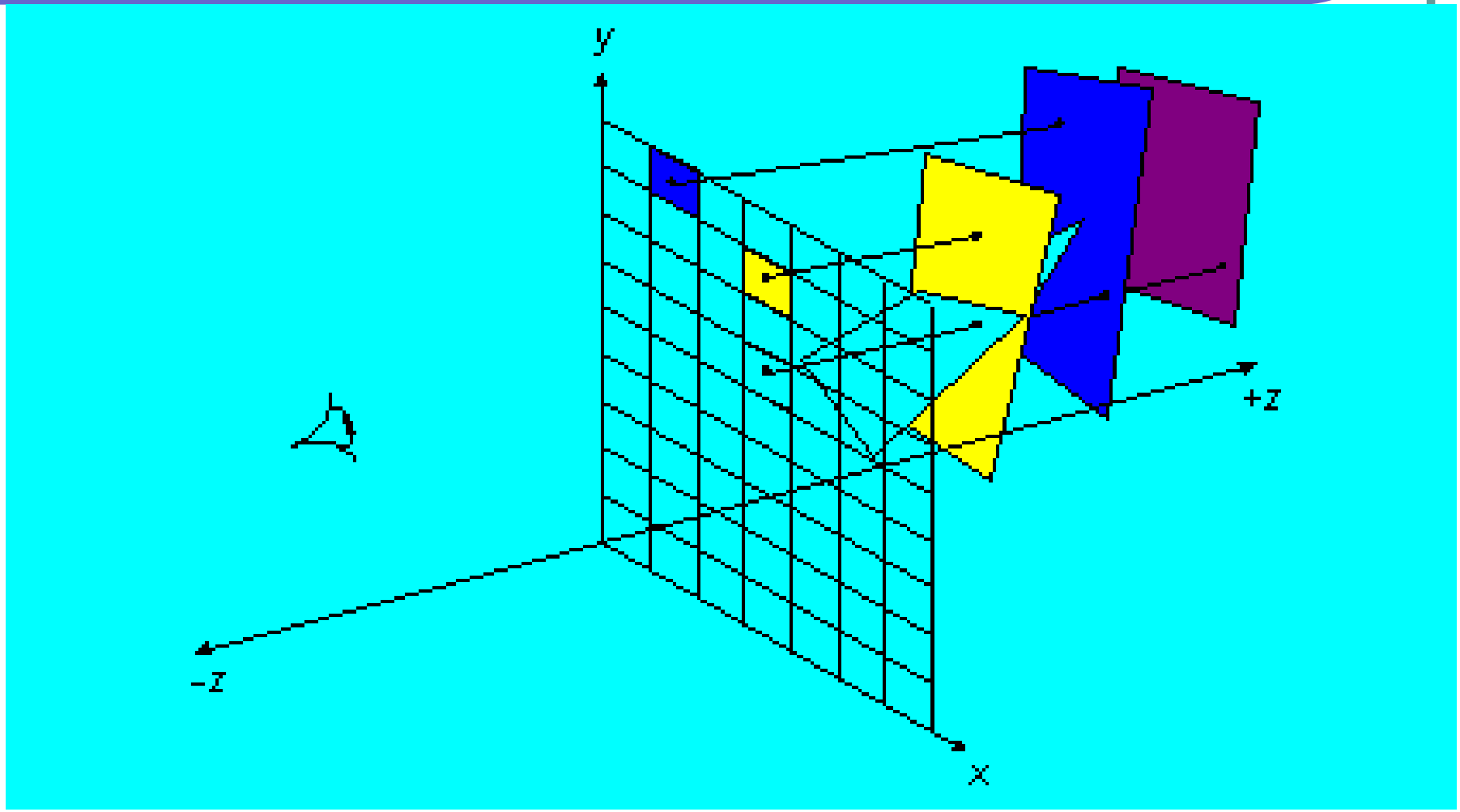
z (*x,y*) = object-distance (*x,y*)

rgb (*x,y*) = object color

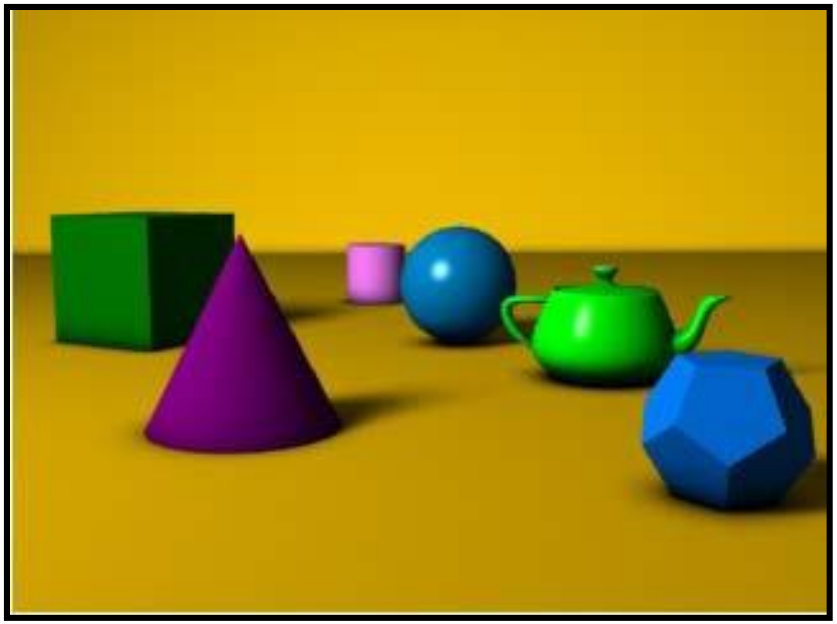
Algorithms in image space
Depth buffer (z-buffer)



Algorithms in image space
Depth buffer (z-buffer)



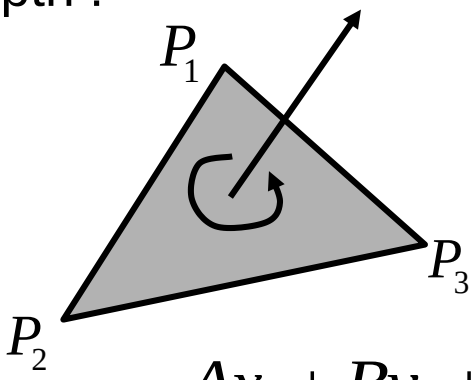
Algorithms in image space
Depth buffer (z-buffer)



Algorithms in image space
Depth buffer (z-buffer)

Scan- conversion

Depth :



$$\vec{N} = \frac{(P_2 - P_1) \times (P_3 - P_1)}{\left\| (P_2 - P_1) \times (P_3 - P_1) \right\|} = (A, B, C)$$

$$Ax_1 + By_1 + Cz_1 + D = 0 \quad \Rightarrow \quad D = -Ax_1 - By_1 - Cz_1$$

$$z(x, y) = \frac{-D - Ax - By}{C}$$

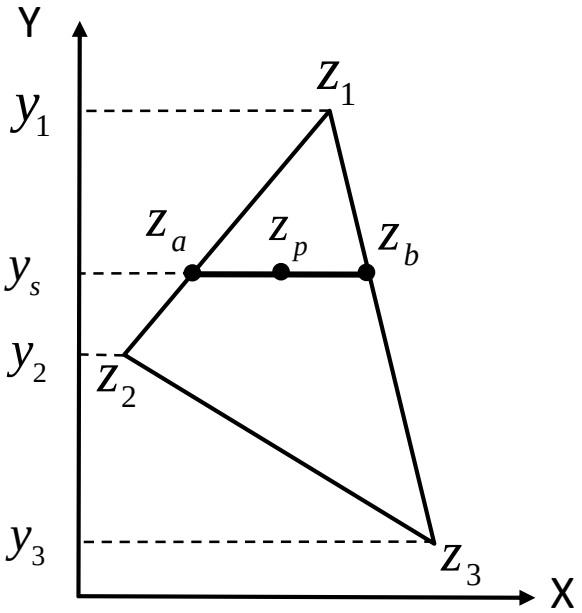
$$z(x + \Delta x, y) = \frac{-D - A(x + \Delta x) - By}{C} = z(x, y) - \frac{A\Delta x}{C} \left.\begin{array}{l} \Delta x = 1 \\ A/C = const \end{array}\right\}$$

$$z(x, y + \Delta y) = \frac{-D - Ax - B(y + \Delta y)}{C} = z(x, y) - \frac{B\Delta y}{C} \left.\begin{array}{l} \Delta y = 1 \\ B/C = const \end{array}\right\}$$

http://csis.pace.edu/~marchese/CG/Lect9New/cg_l9new.htm

Algorithms in image space
Depth buffer (z-buffer)

Bilinear interpolation (alternative)



$$y_s = y_1 + t(y_2 - y_1)$$

$$z_a = z_1 + t(z_2 - z_1)$$

$$\frac{z_a - z_1}{z_2 - z_1} = \frac{y_s - y_1}{y_2 - y_1}$$

$$z_a = z_1 + (z_2 - z_1)\frac{(y_s - y_1)}{(y_2 - y_1)}$$

$$z_b = z_1 + (z_3 - z_1)\frac{(y_s - y_1)}{(y_3 - y_1)}$$

Also interpolates:
color, texture, normal, etc.

$$z_p = z_a + (z_b - z_a)\frac{(x_p - x_a)}{(x_b - x_a)}$$

Algorithms in image space
Depth buffer (z-buffer)

Advantages/disadvantages:

+ simplicity, generality
+ hardware, parallelism
- additional memory for buffers
+ no memory to keep the scene
- aliasing (in XY and Z)