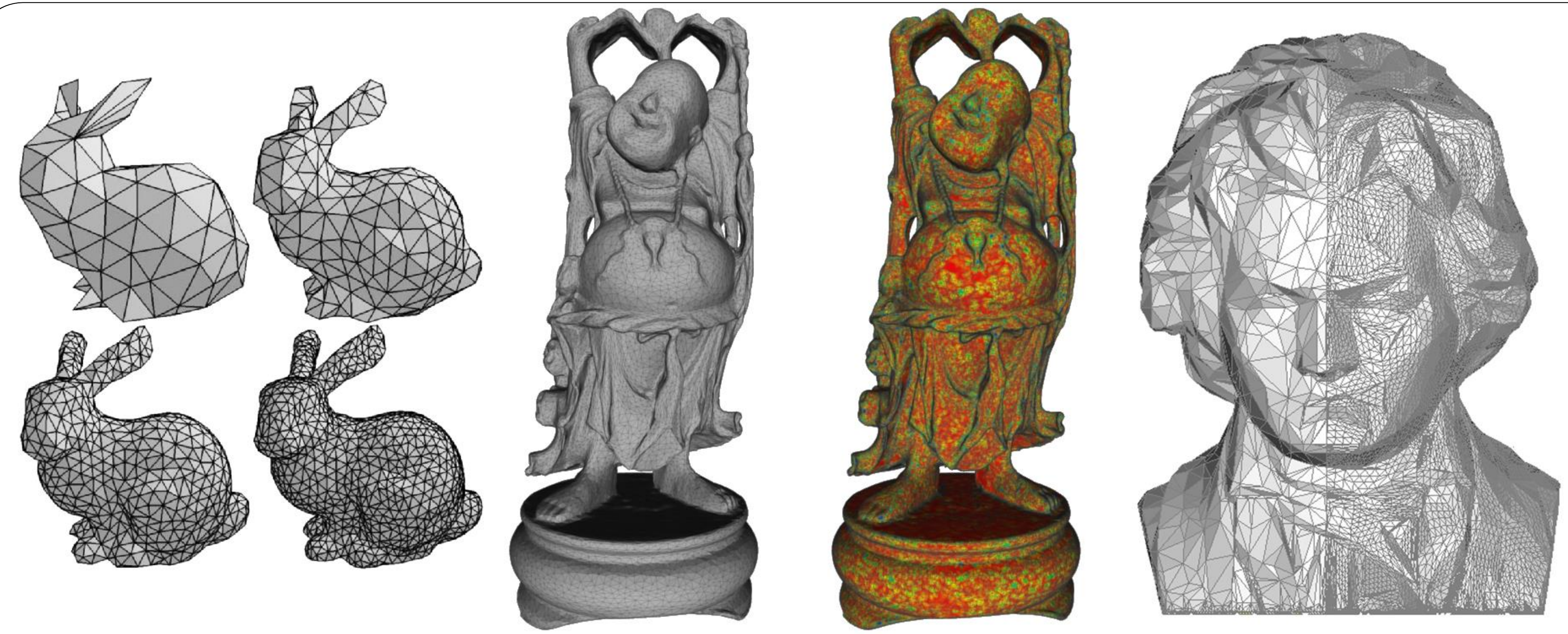


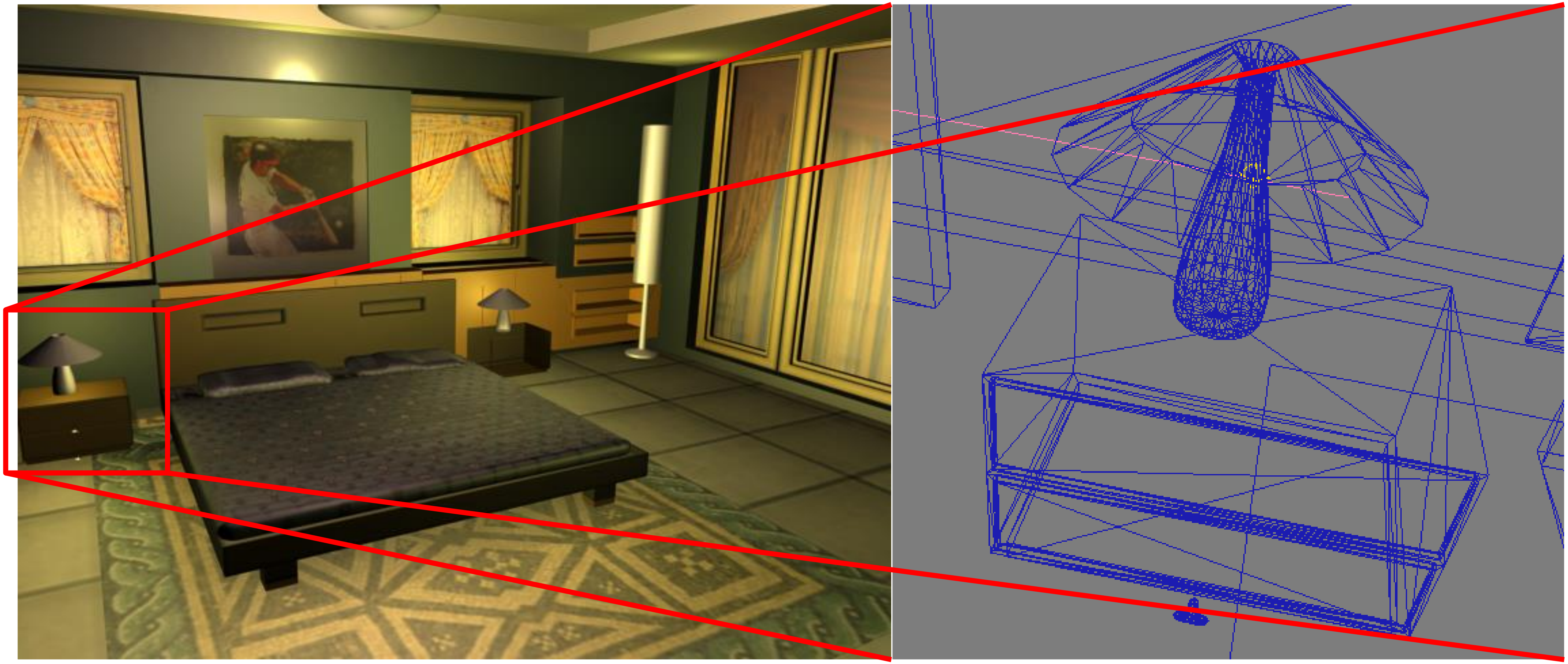
Computer Graphics

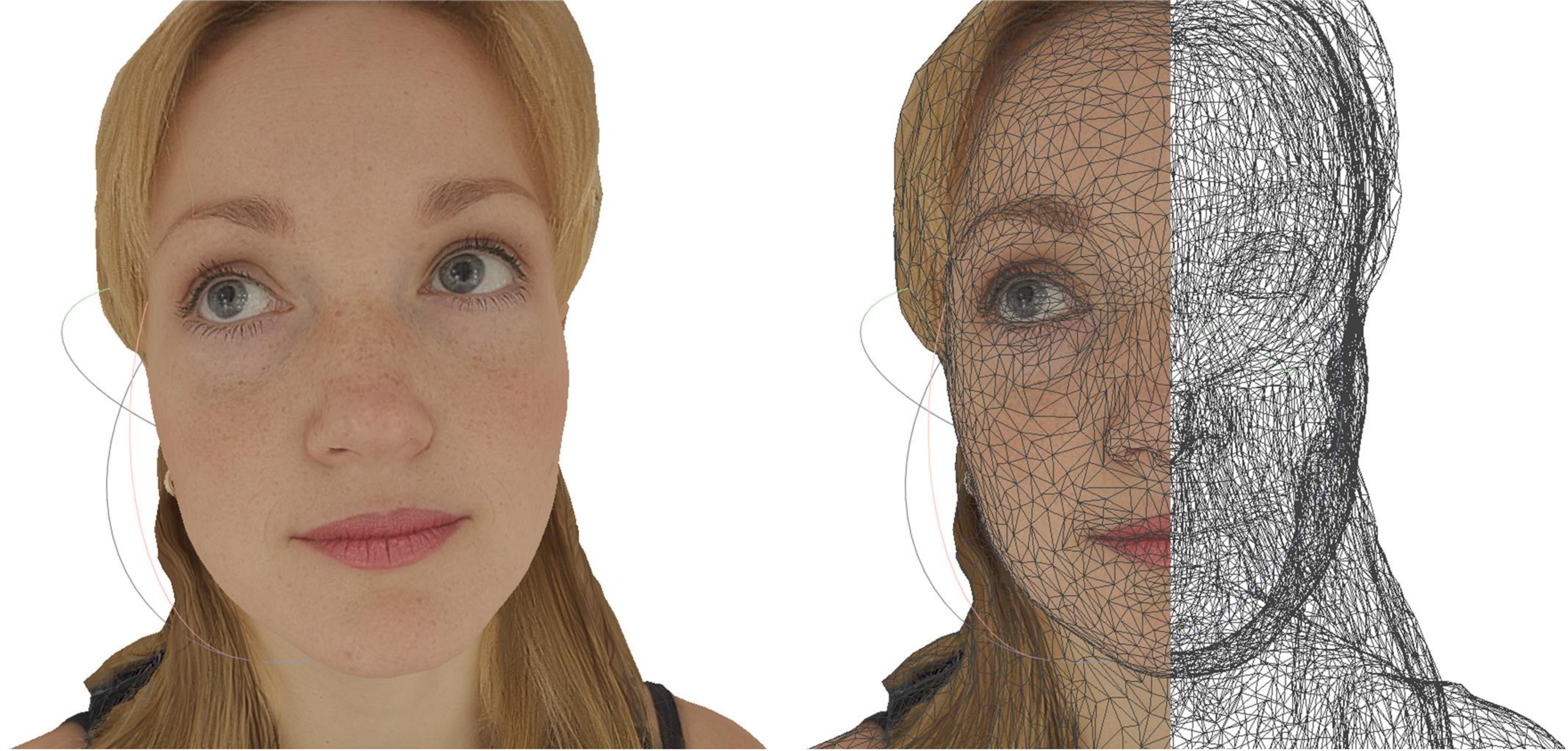
Polygonal Models

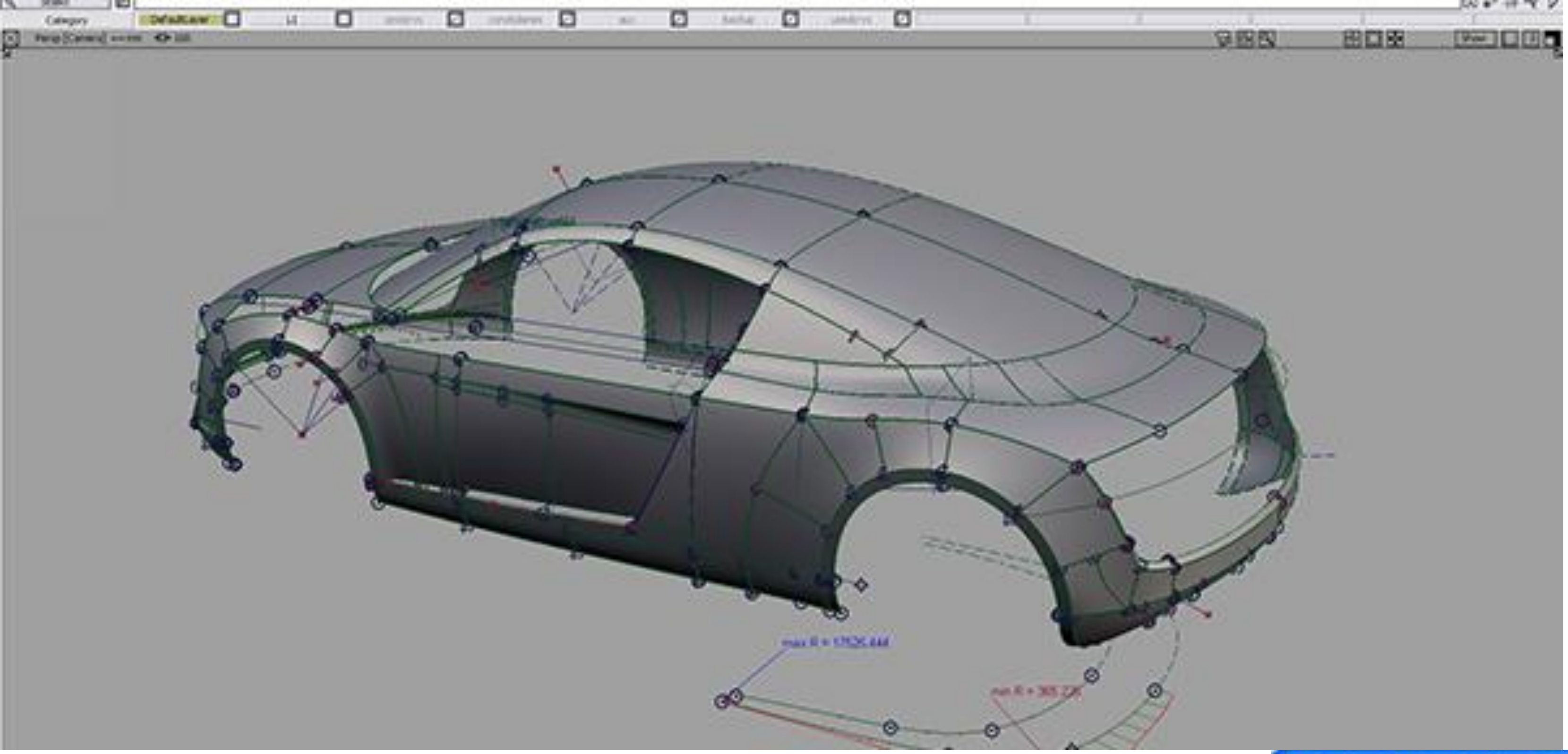
Andreas Aristidou
andarist@ucy.ac.cy
http://www.andreasaristidou.com

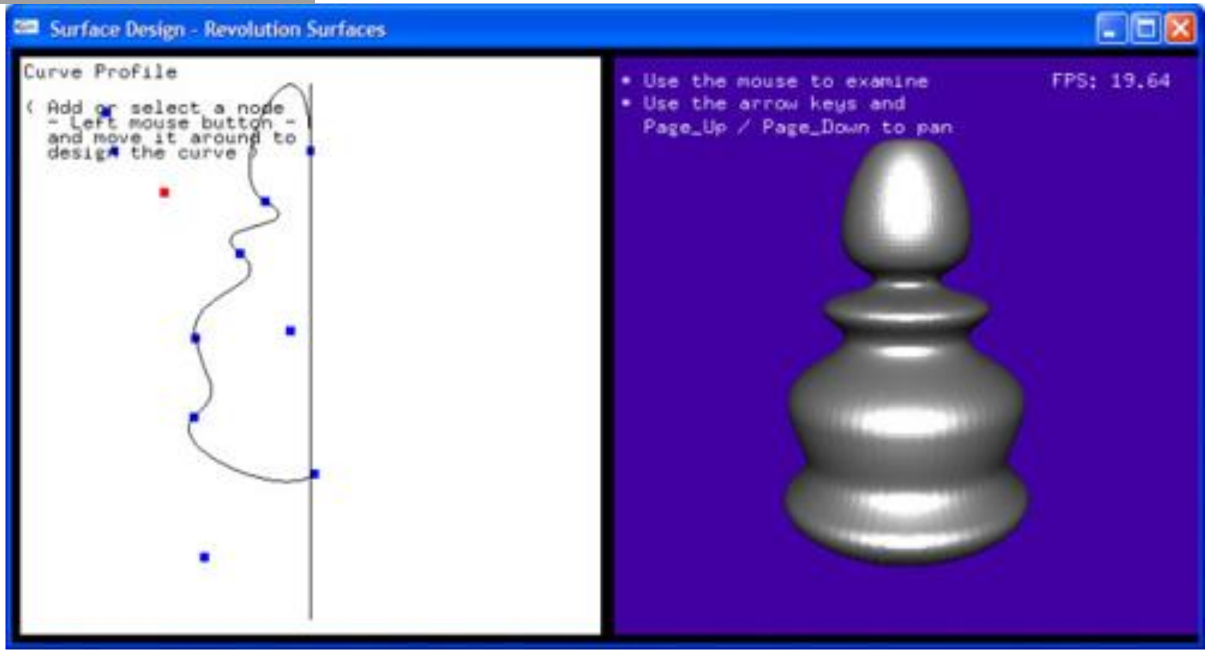
Contents

- We'll see how objects are shaped into 3D
- Graphic Modeling
  - Polygons
  - Polyhedra
  - Tetrahedral surfaces
  - Sweep Representations
  - Constructive Solid Geometry



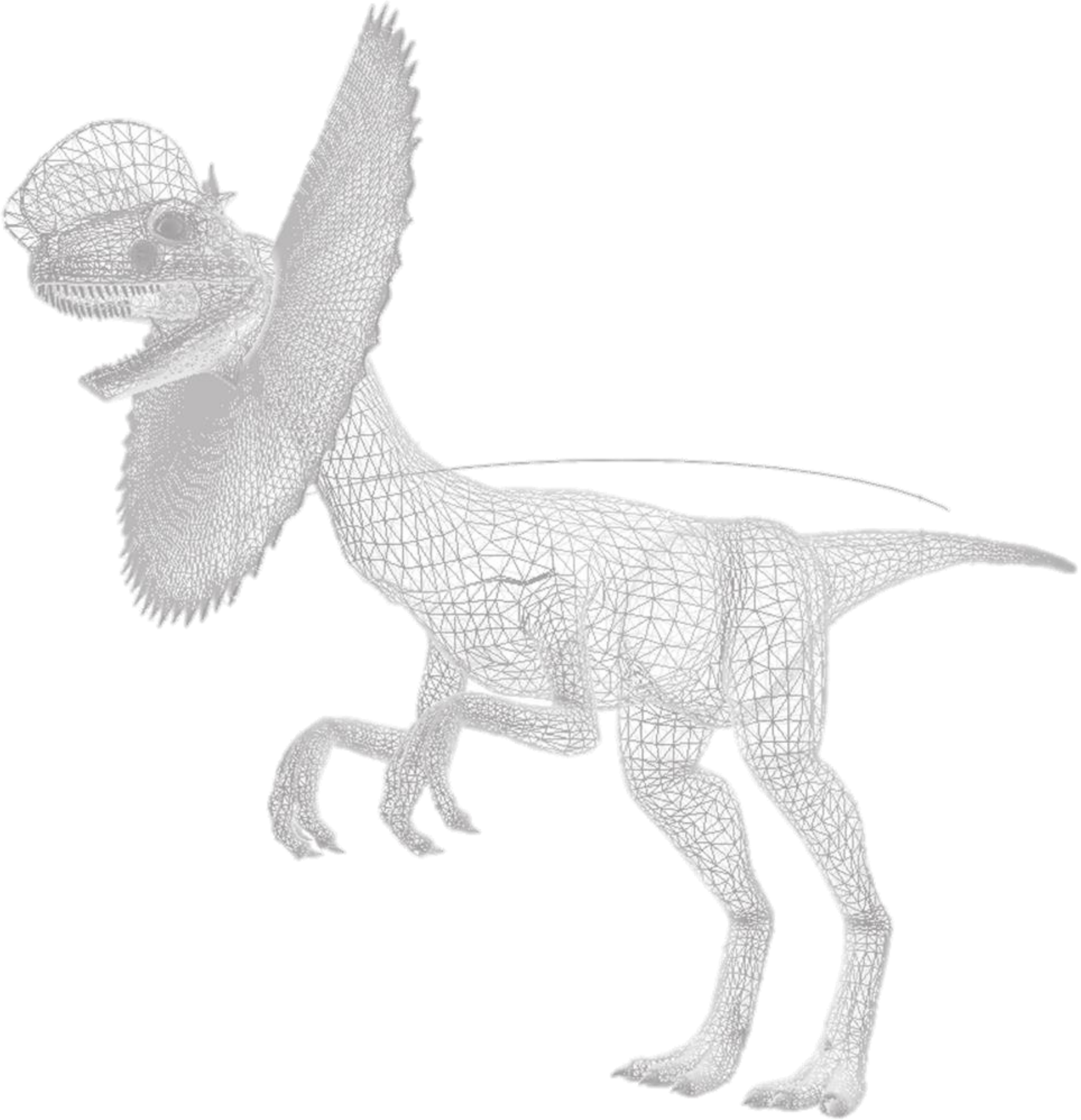





Surface Design - Revolution Surfaces
Curve Profile
(Add or select a node - Left mouse button - and move it around to design the curve)
• Use the mouse to examine
• Use the arrow keys and Page_Up / Page_Down to pan
FPS: 19.64

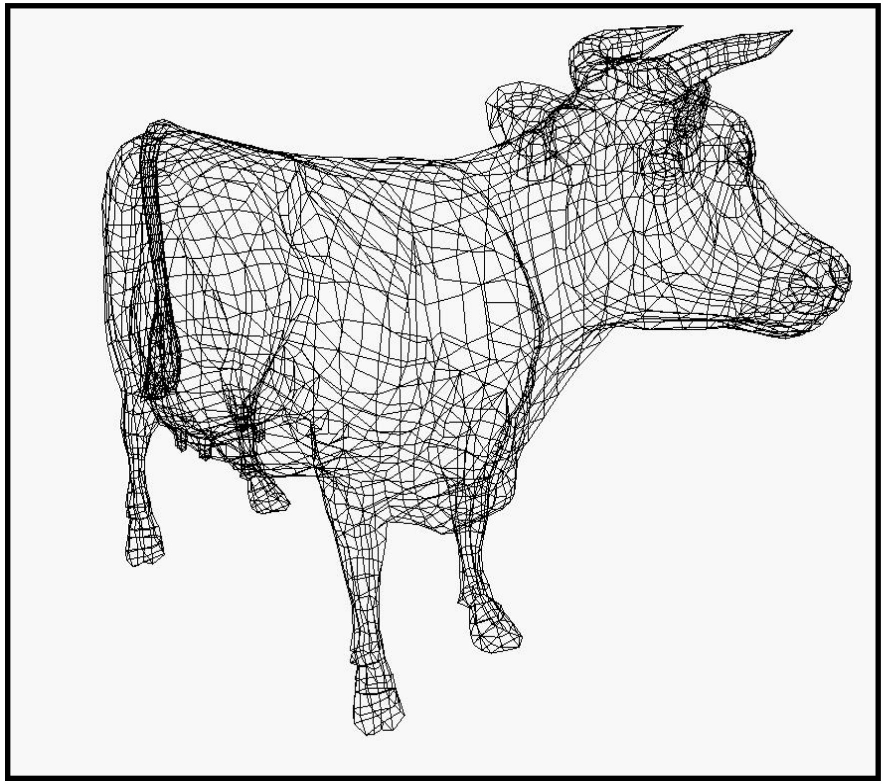
Objects

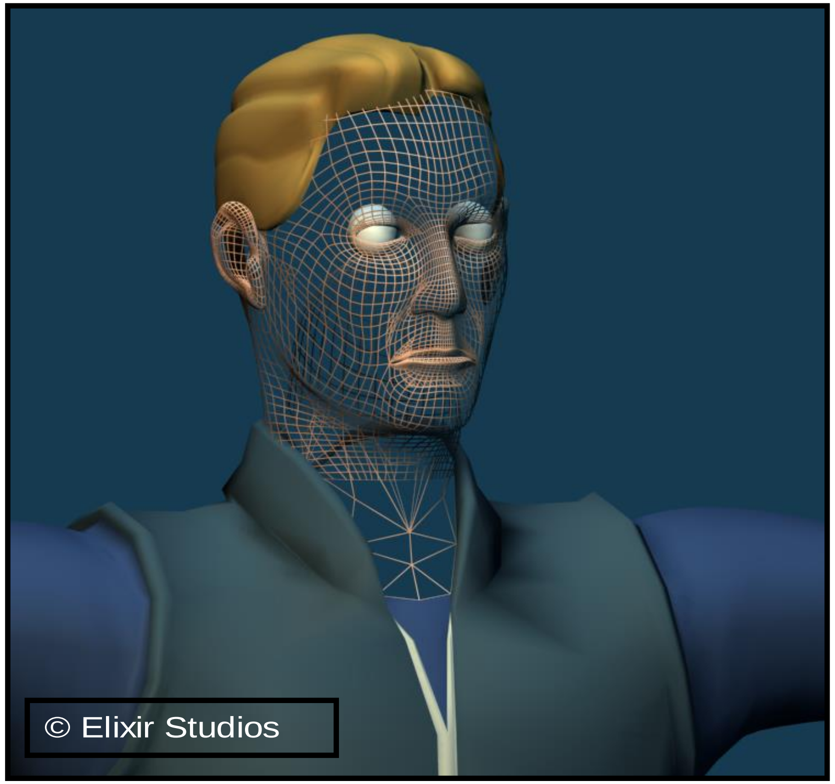
- Objects consist of geometry + materials
- **Geometry** – usually a 3D mesh
  - Approaches a continuous surface with a set of polygons (triangles + quadruple)
  - In rendering, we can also attribute mathematical models and volumes
- **Material** – describes how light interacts with the object



Geometric modeling

- With a combination of polygons we can describe any shape





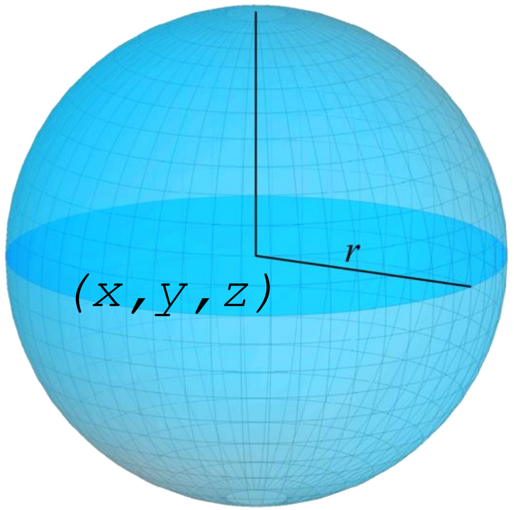
Of course the number of polygons can be very large

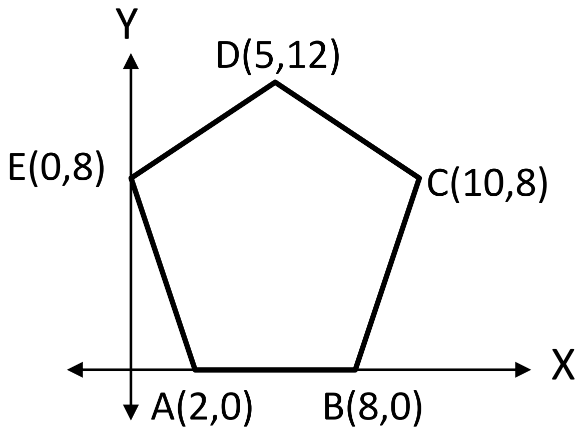
Geometric modeling

- Simple objects can be represented directly:

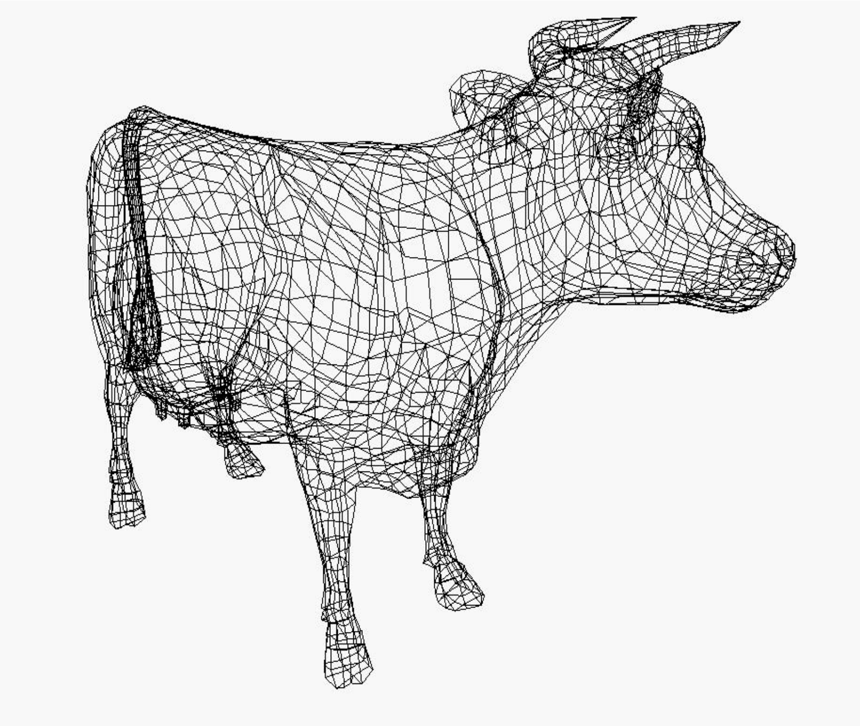
E.g.

  - a sphere is described only by 4 values (x, y, z, r)
  - a polygon is described by the coordinates of its vertices.





Polygons and polyhedrons



A. Polygons
   - Types of polygons
   - Polygon - ray intersection
   - Point within the polygon

B. Polyhedrons

Polygons

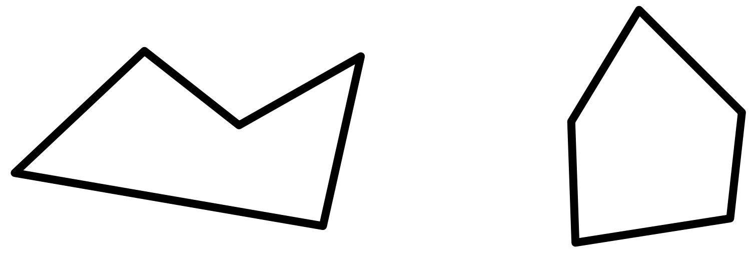
- A ***polygon*** (polygon or face = surface) is defined by a series of *n* points (vertices)

$$[p_0, p_1, p_2, ..., p_{n-1}]$$
$$p_i = (x_i, y_i, z_i)$$

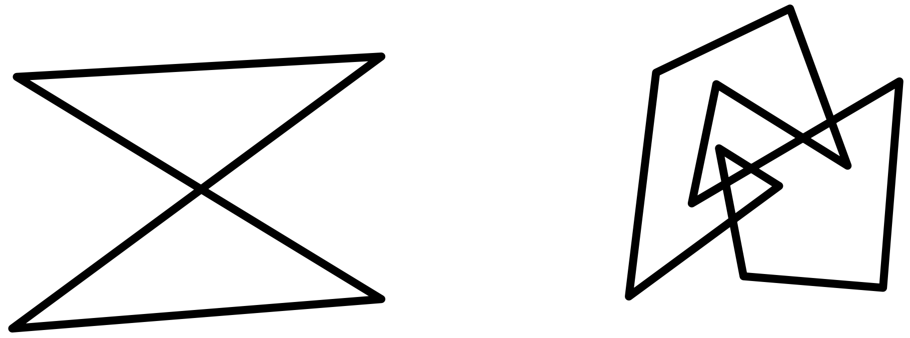
- **The points must be on the same plane.**
- 3 points define a plane. A fourth point is not necessarily at the same point as the other 3.

Types of polygons

- Simply - their edges do not intersect
  - concave, convex



- Complex

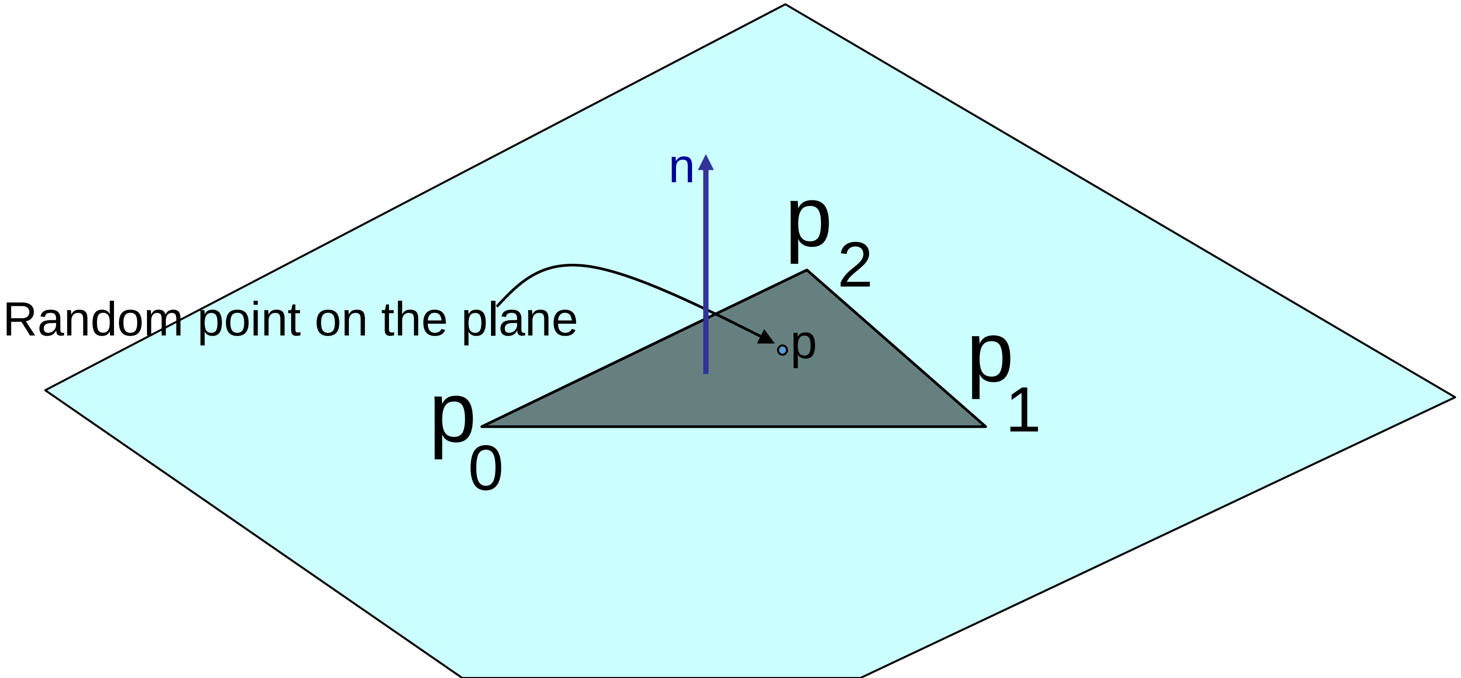


We are mainly interested in the **simple convex** polygons since the algorithms for them are simpler and faster. They can also be easily broken down into triangles.

Equation of the plane: *Revision*

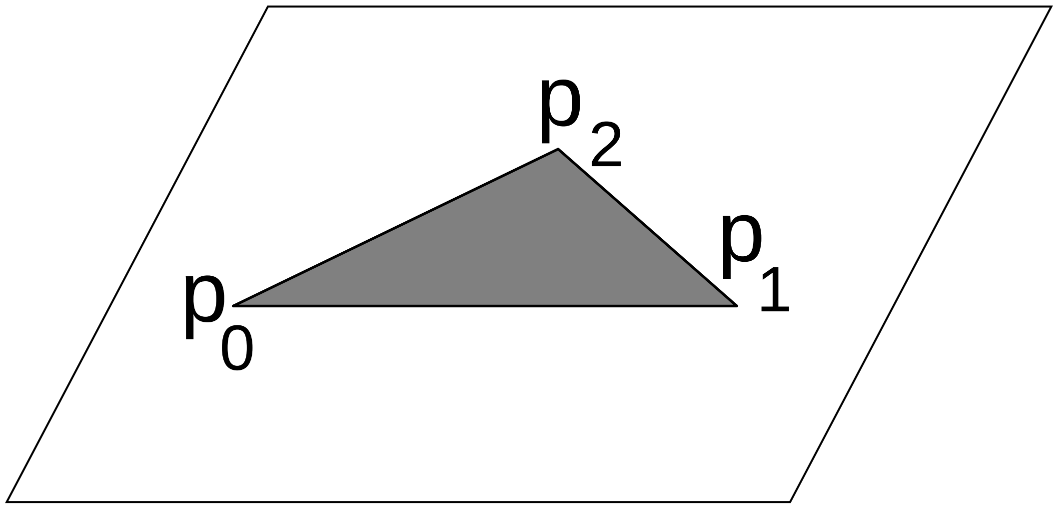
- a,b,c and d are **constants** that define a unique plane in space
- The (a, b, c) = n define the **perpendicular** on the plane
- Some point p (x, y, z) is on the plane, if and only if satisfies the equation:

$$ax + by + cz + d = 0$$

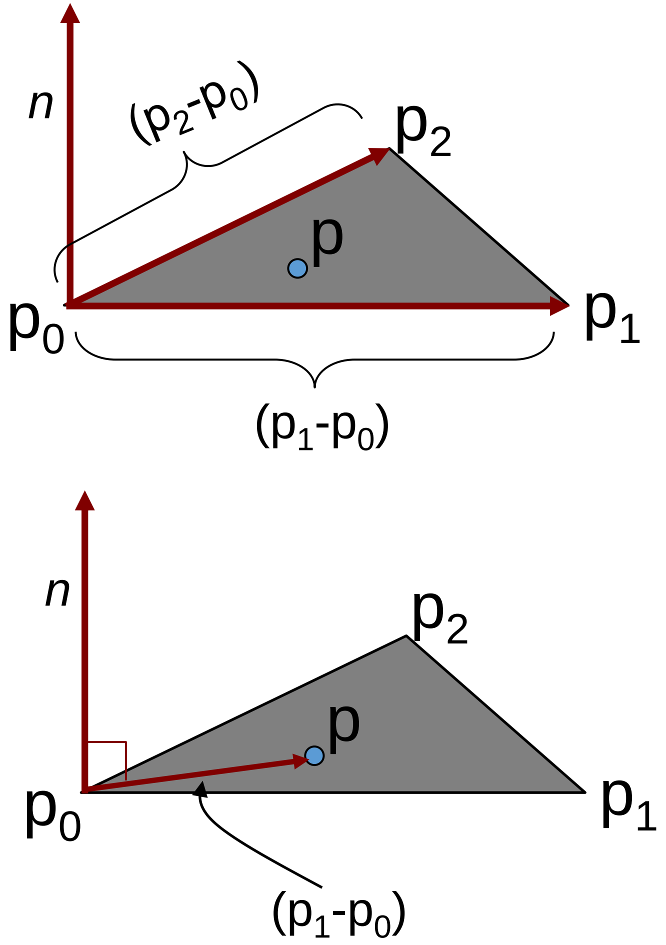


Equation of the plane: *Revision*

- If we have **3 points** we can calculate the equation of the plane:
  - We create 2 vectors and find the outer product, this gives us the (a, b, c)
  - We replace any of the 3 points in the equation and it gives us the d.



How to find the a,b,c & d: *Revision*



- The outer product

$$n = (p_1 - p_0) \times (p_2 - p_0)$$

  defines the perpendicular to the plane

  with n = (a, b, c)

- We have 2 perpendiculars (reverse directions)
- Vectors in the plane are perpendicular to the perpendicular
- From ax + by + cz + d = 0

  ⇨ d = - (ax + by + cz) if we replace the p_0

  ⇨ $d = n.p_0 = -(n_1 * x_0 + n_2 * y_0 + n_3 * z_0)$

Half-Space: *Revision*

- A plane splits a place into two half-spaces
- Let's define:

$$l(x, y, z) = ax + by + cz + d$$

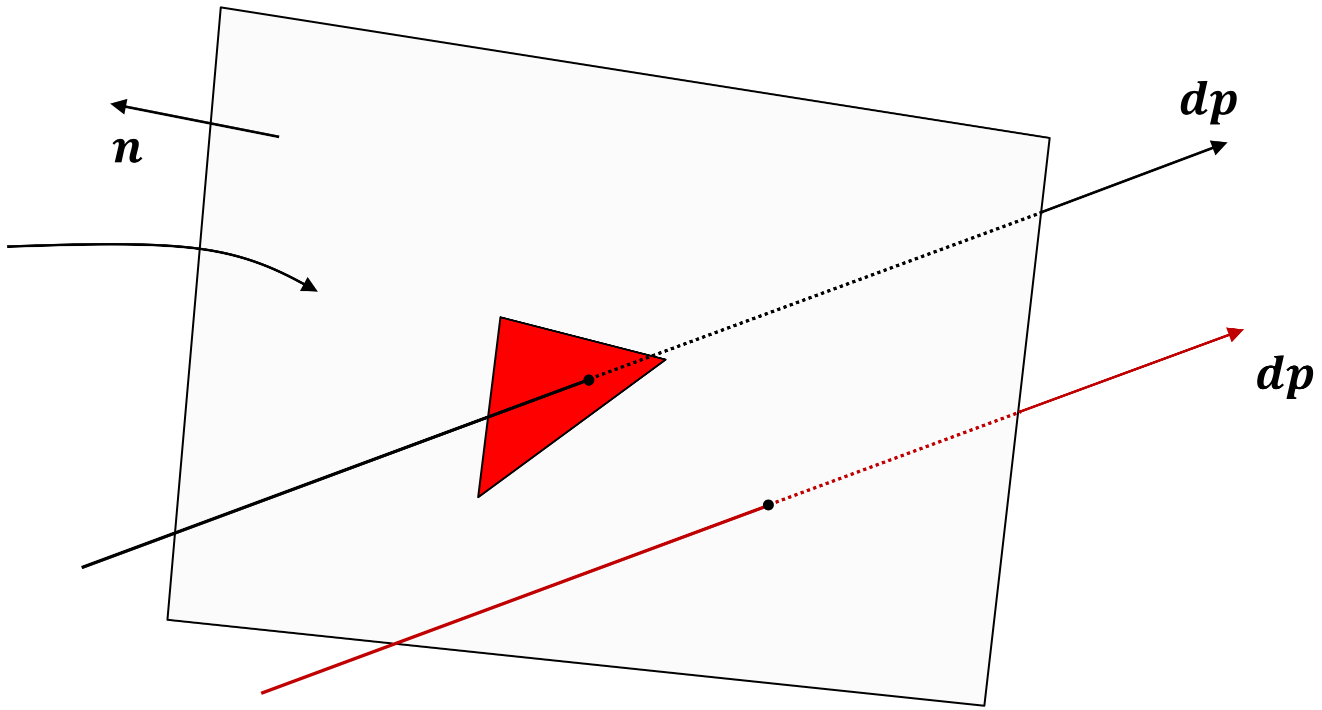
- If l(p) =0
  - The point p is in the plane.
- If l(p) > 0
  - The point p is in positive half-space.
- If l(p) <0
  - The point p is in the negative half-space.

Useful then when we talk about Visible Surface Detection etc

Description of the polygon-ray intersection

- Three steps
  - Does the ray (line) intersects the **plane** (of the polygon)?
    - To make sure that they are not in parallel.
  1. We find the **intersection point** (of ray & plane)
  2. We check if the point is **in** the polygon.

The infinite plane that defined by the polygon



Useful then when we talk about Visible Surface Detection etc

Step 1: Does the ray intersect the plane?

- The parametric equation of the ray is:
- $$\mathbf{p} = \mathbf{p}_0 + t.\mathbf{dp}$$
  - Where **dp** gives us the direction of the ray.
- The equation of the plane is:

$$ax + by + cz + d = 0 \quad \text{ή}$$

$$\mathbf{n}.(x,y,z) + d = 0$$

  - where **n** is the perpendicular
- Then you just need to check if

$$\mathbf{n.dp} \; != 0$$

  - Which means they are not perpendicular

Step 2: Where does it intersect?

- We replace the equation of the line in the equation of the plane

$$n.\left(x_0 + t.dx \quad y_0 + t.dy \quad z_0 + t.dz\right) = -d$$

- We solve as to t

$$t = \frac{-d - (n \cdot p_0)}{n \cdot dp}$$

- And we find the p_i from

$$p_i = p_0 + t \cdot dp$$

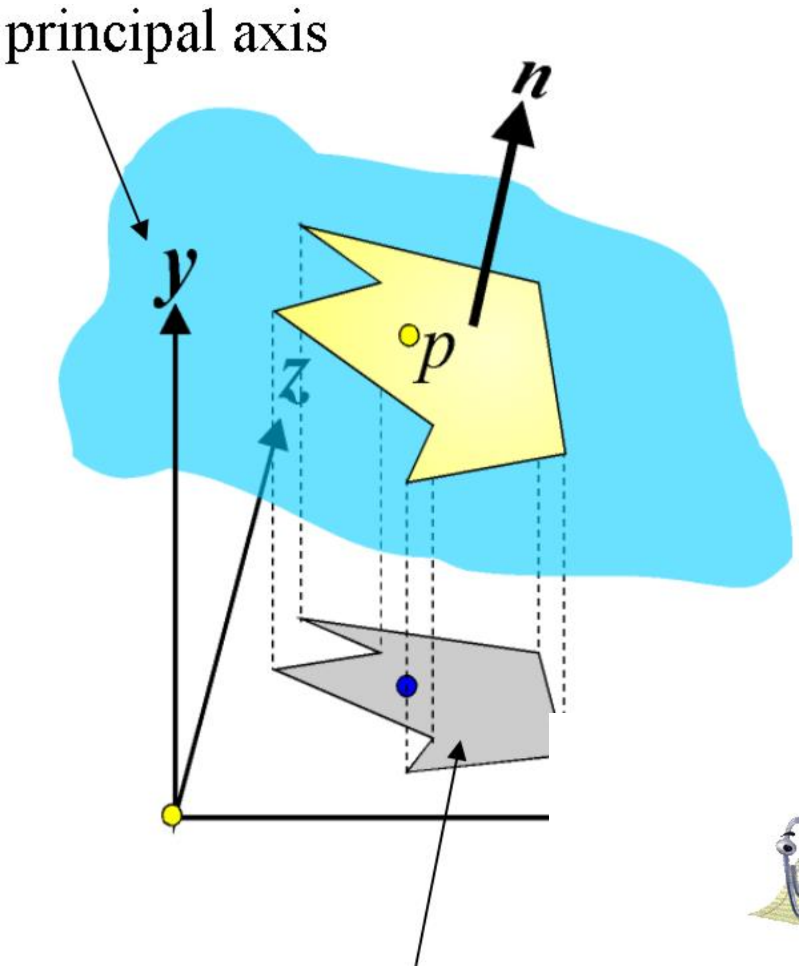
Useful then when we talk about Visible Surface Detection etc

Step 3: Is it the intersection within the polygon?

- It is easier to do the test in 2 dimensions than in 3
- First we "project" the polygon and the point in 2D
- Then we have an several tests:
  - Sum of angles
  - Half-space test
  - Sum of edges

Project on 2D

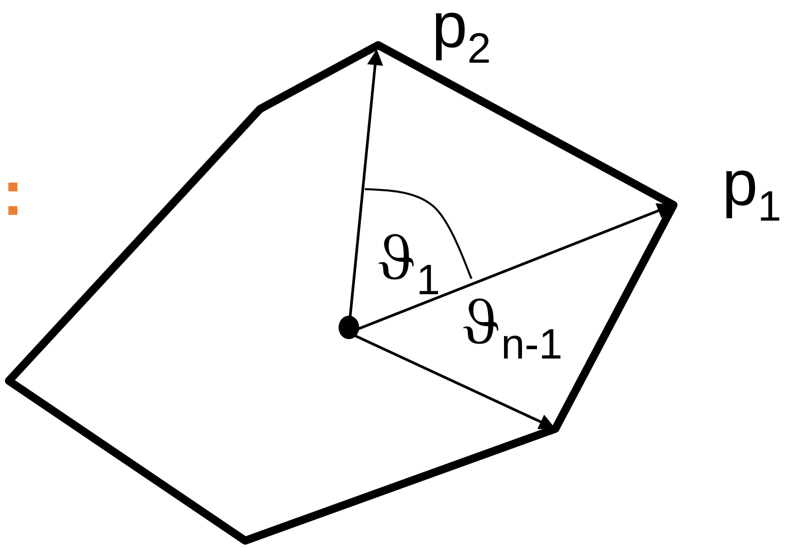
- We choose one of the primary planes
- If the perpendicular of the polygon is $n=(n_x, n_y, n_z)$ then we choose the biggest of the three values



Sum of angles (Convex polygons)

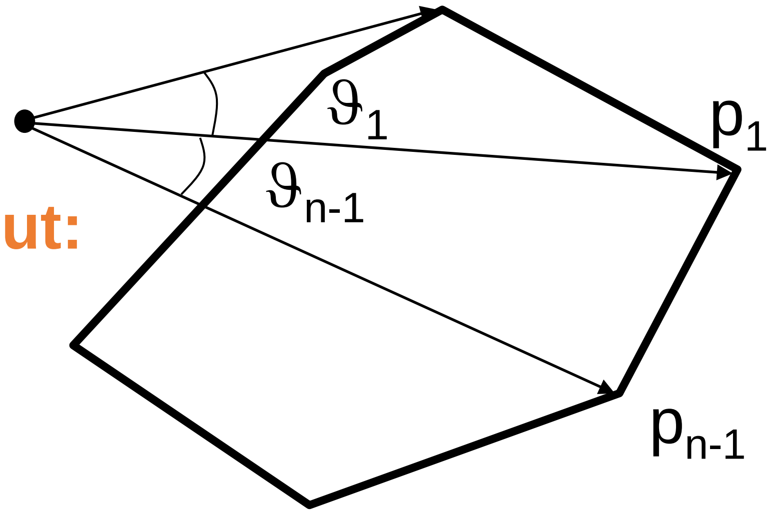
We sum the angles that have as their vertex the point p

In:



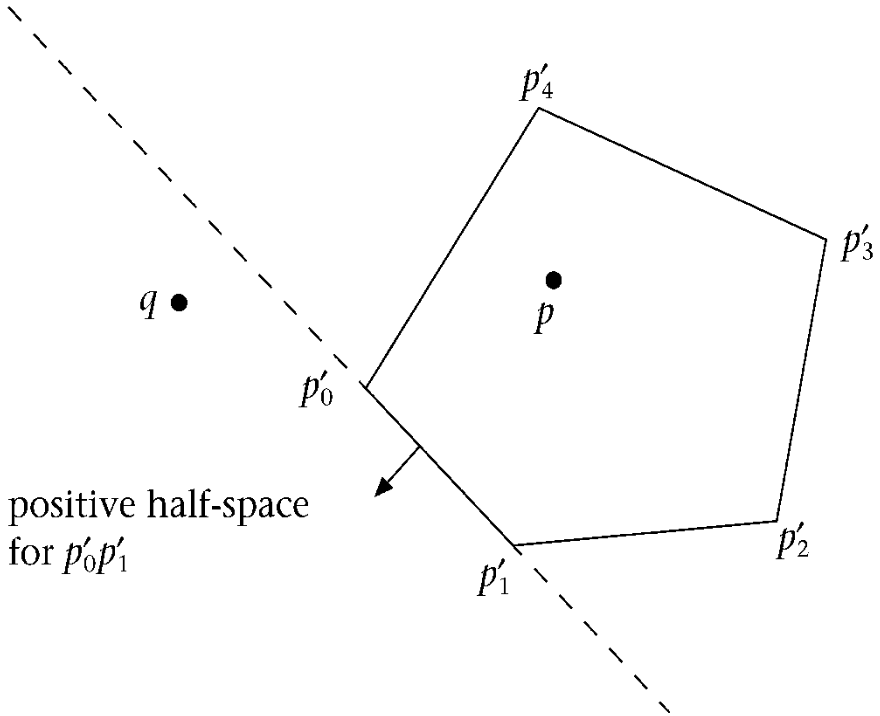
$$\sum_{i=1}^{n} \vartheta_i = 2\Pi$$

Out:



$$\sum_{i=1}^{n} \vartheta_i \neq 2\Pi$$

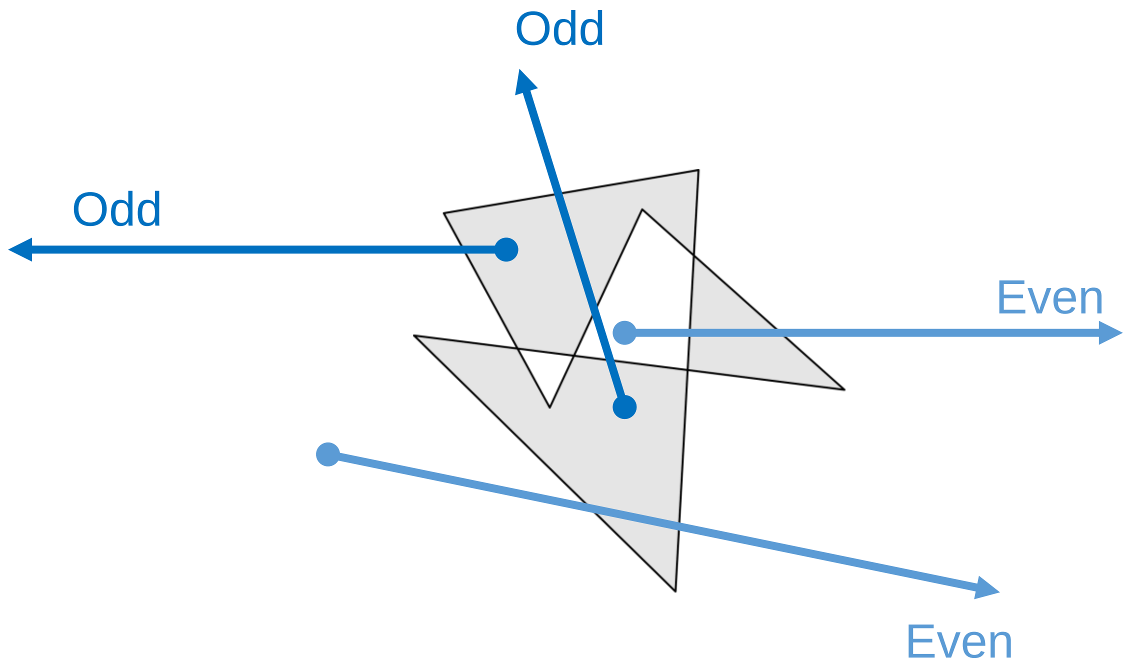
Half-Space Test (Convex polygons)



- the p-point is inside the polygon only if it is in the negative half-space of all edges (sides)

Sum of edges (general polygons)-Odd-even rule

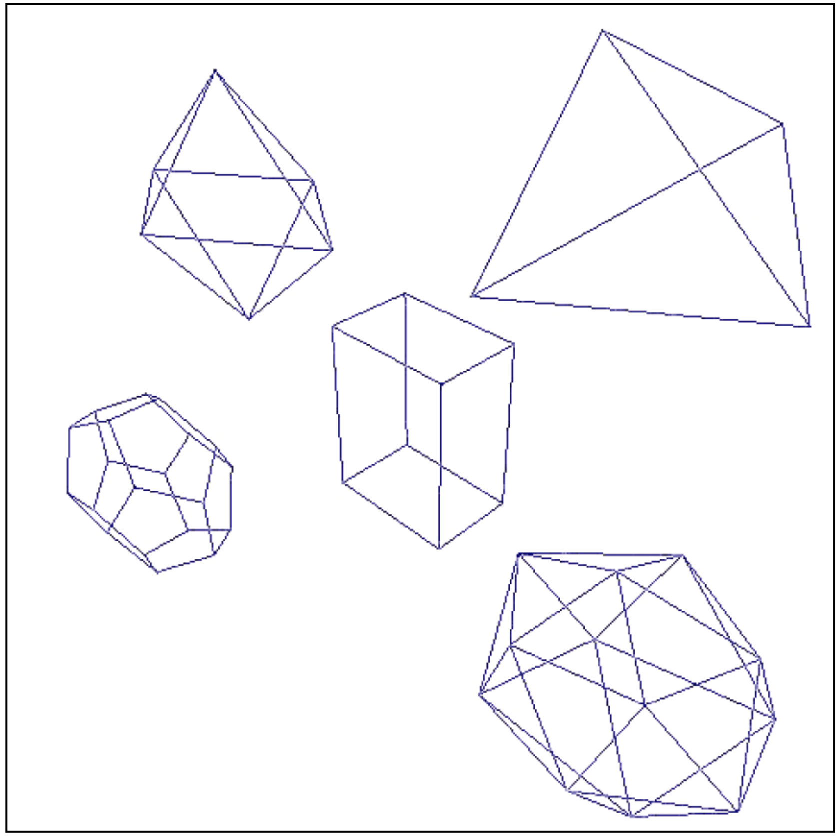
- We send a ray to infinity in any direction
- Whether the point is inside or outside, depends on whether the number of intersecting sides is odd (in) ή even (out)

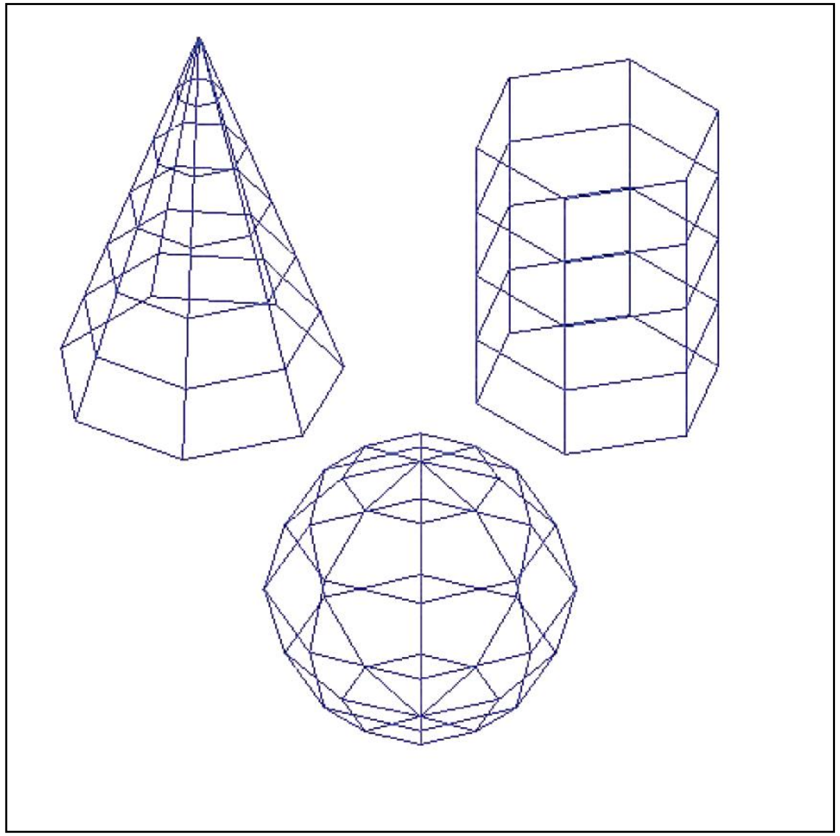


Polyhedrons (or Polyhedra)

- Objects are just a series of polygonal surfaces
- It is the simplest and fastest way to render objects
- Often referred to as ***standard graphics objects***
- In many cases we are allowed to define objects such as curved surfaces, but in reality they are converted to polygon meshes for their appearance
- To define polyhedrons, we simply define the vertices of the polygons required

Polyhedrons



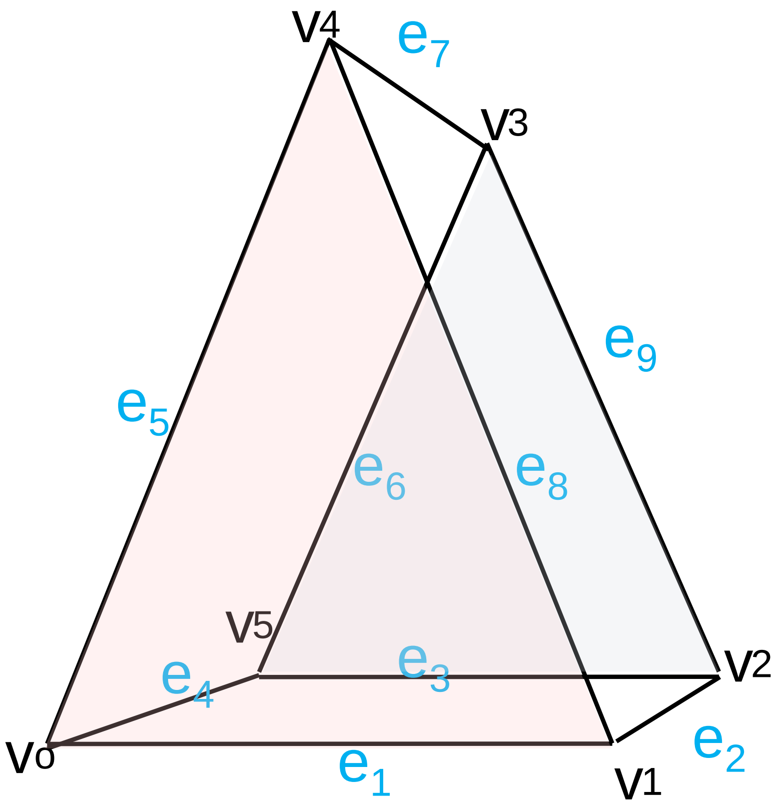


Polyhedrons

- Each object can be a group of **Polygons or Faces** : one polyhedron. Or, it can be even expressed as a series of polyhedrons.
  - Each ***Edge*** consists of 2 ***Vertices*** and joins two polygons
  - Usually each vertex is common for 3 sides
  - Polygons do not intersect

- For closed objects:
  - V-E+F=2
  - #Vertices - #Edges + #Faces = 2
  - For cubes, tetrahedrons, cows, etc...

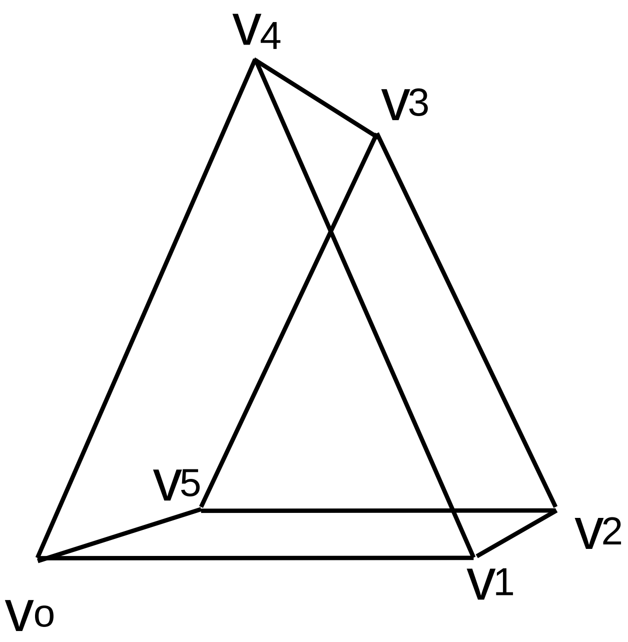
Example of a Polyhedron



- $F0=v_0v_1v_4$
- $F1=v_5v_3v_2$
- $F2=v_1v_2v_3v_4$
- $F3=v_0v_4v_3v_5$
- $F4=v_0v_5v_2v_1$

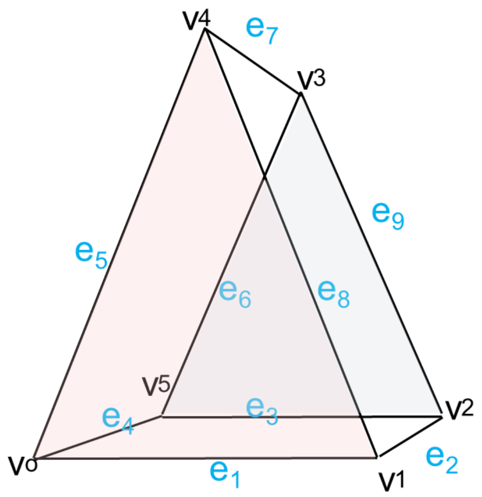
- $V=6, F=5, E=9$
- $V-E+F=2$

The order of the vertices is important



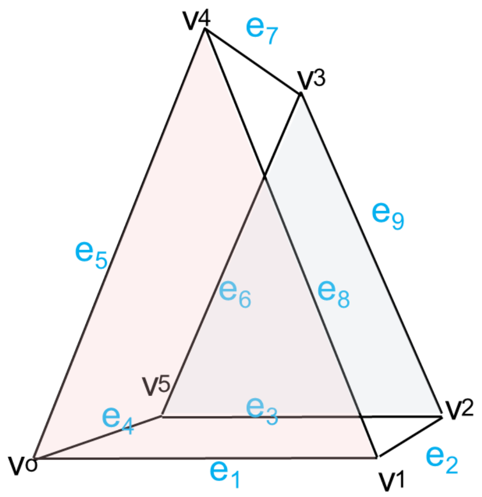
- The polyhedron $\{v_0,v_1,v_4\}$ is not the same as the $\{v_0,v_4,v_1\}$
- Their perpendicular shows to the opposite direction.
- Usually each polygon is visible only from its positive half-space
- This is well known, as we will see in later slides, as **back-face culling**.

Representation of polyhedrons



- Exhaustively (array from series of vertices)
  - faces[0] = (x_0,y_0,z_0), (x_1,y_1,z_1), (x_4,y_4,z_4)
  - faces[1] = (x_5,y_5,z_5), (x_3,y_3,z_3), (x_2,y_2,z_2)
  - etc ….
- Not efficient, since each vertex is presented (at least) 3 times in the list.
- However, it is used quite a lot!

Representation of polyhedrons



Indexed Face set

1. **Vertex array**
   - vertices[0] = (x_0, y_0, z_0)
   - vertices[1] = (x_1, y_1, z_1)
   - etc …

2. **Face array** – a list of pointers on the vertex array
   - faces[0] = 0, 1, 4
   - faces[1] = 5, 3, 2
   - etc ...

Quadric Surfaces

- A category of objects that is often used is quadric surfaces
- These are 3D surfaces that are described using quadratic equations
- Some quadric surfaces objects are:
  - Spheres
  - Ellipsoid
  - Tori
  - Paraboloids
  - Hyperboloids

$$x^2 + y^2 + z^2 = r^2$$

$$x = r\cos\phi\cos\theta \qquad -\pi/2 \le \phi \le \pi/2$$

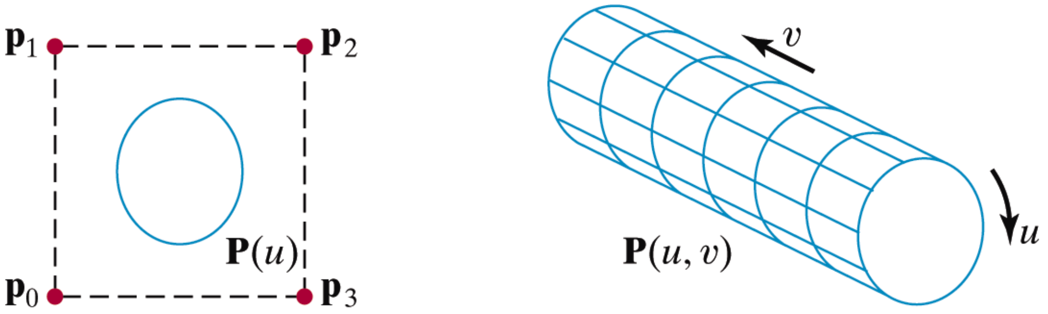
$$y = r\cos\phi\sin\theta$$

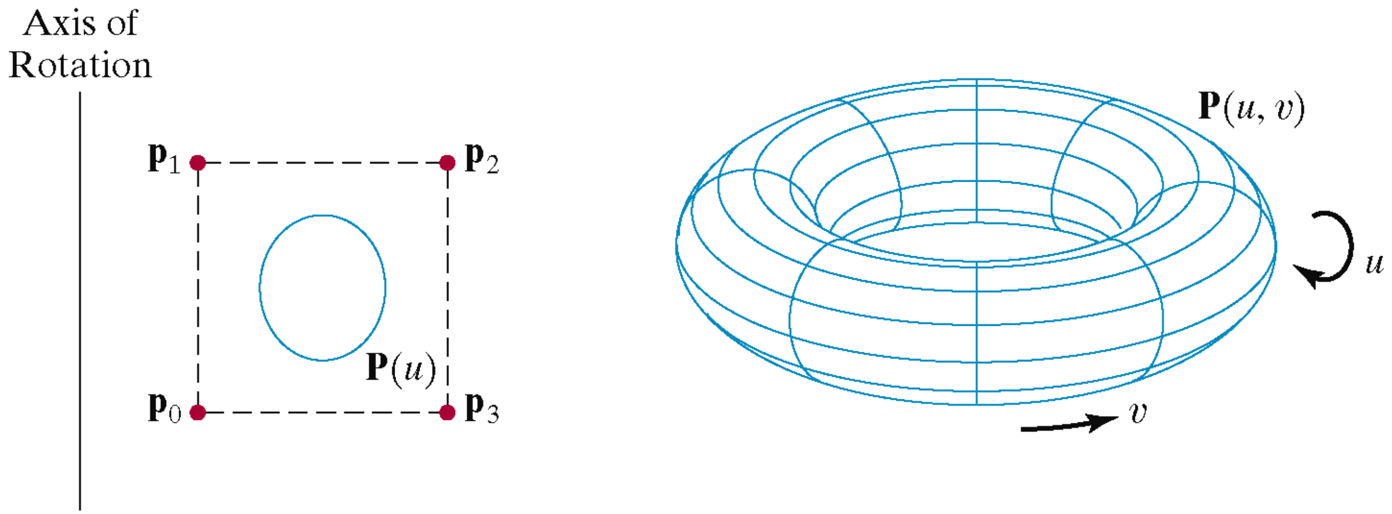
$$z = r\sin\phi \qquad -\pi \le \theta \le \pi$$

Sphere (equation & parametric equation)

Sweep Representations

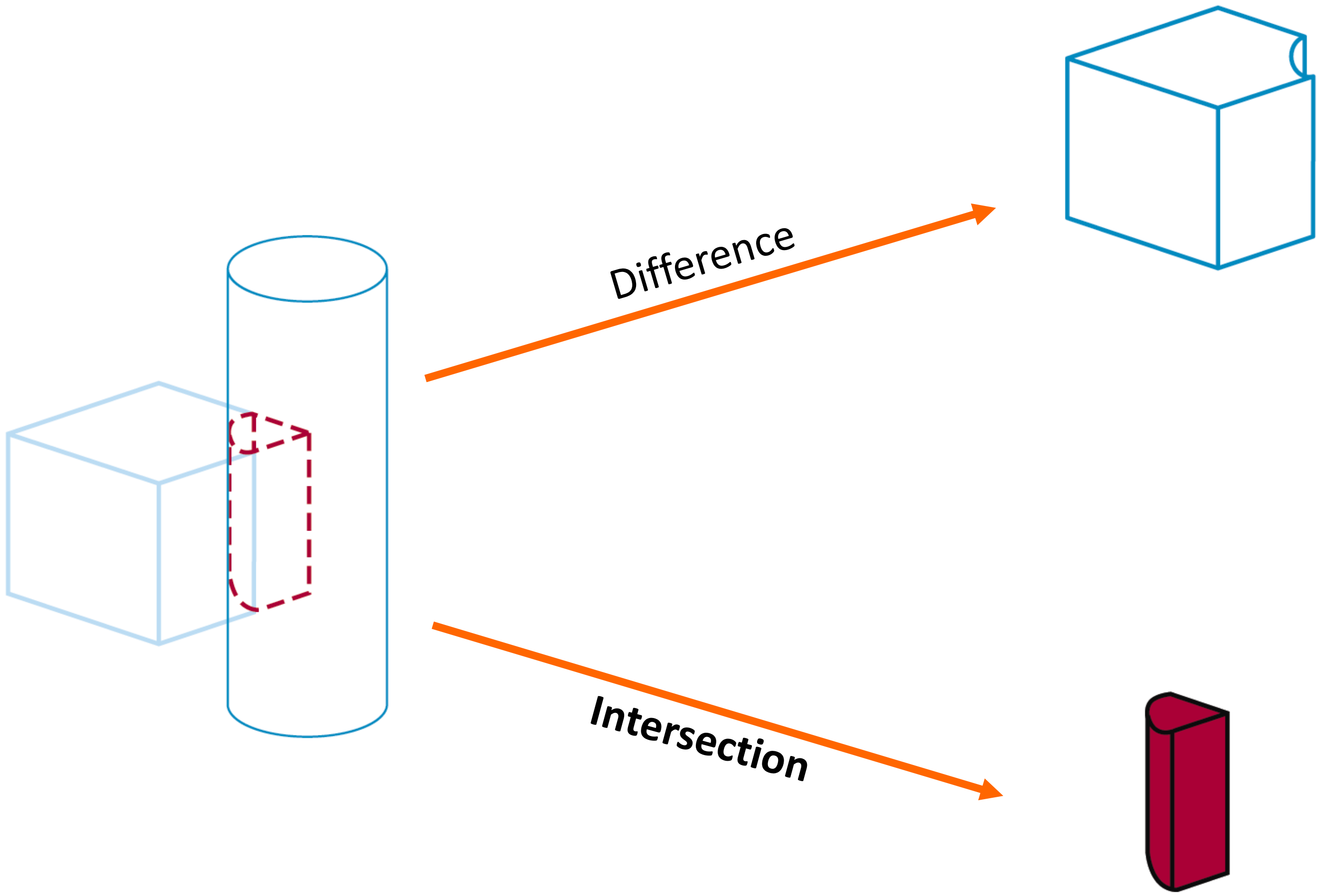
- Sweep Representations are useful for constructing 3 dimensional objects using translation, rotation, or other symmetries





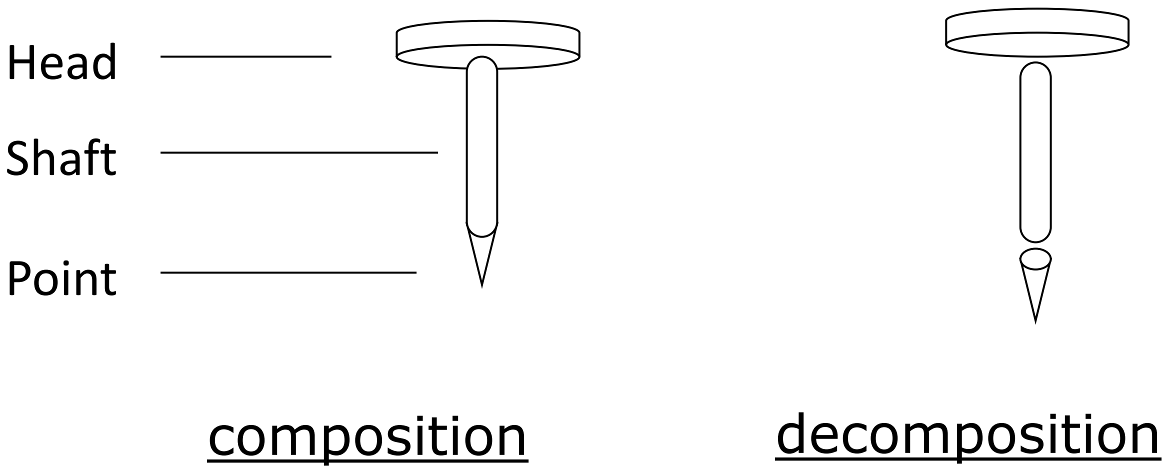
Constructive Solid Geometry

- Constructive Solid Geometry (CSG) performs compact modeling by creating a new object from other objects using a specific function such as:
  - Union
  - Intersection
  - Difference



Decomposition of a geometric model: *Revision*

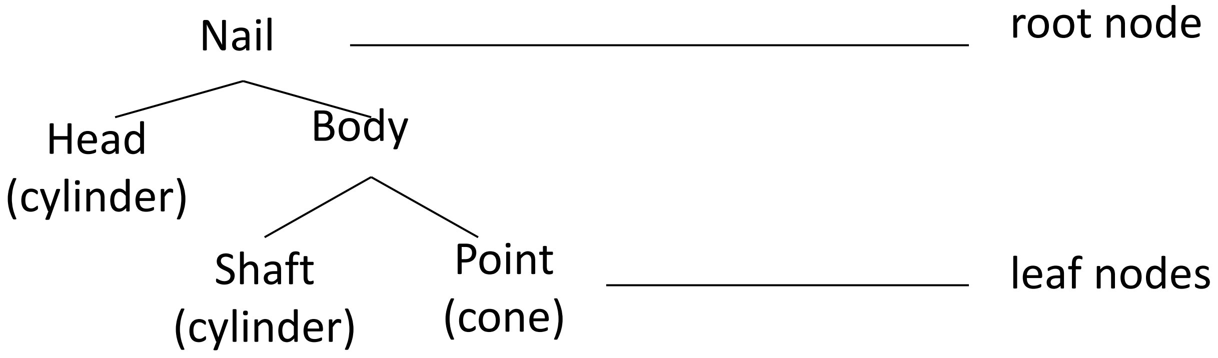
- Divide and Conquer
- Hierarchy of geometrical components
- Reduction to primitives (e.g., spheres, cubes, etc.)
- Simple vs. not-so-simple elements (nail vs. screw)



Scene graph: *Revision*

Hierarchical (Tree) Diagram of Nail

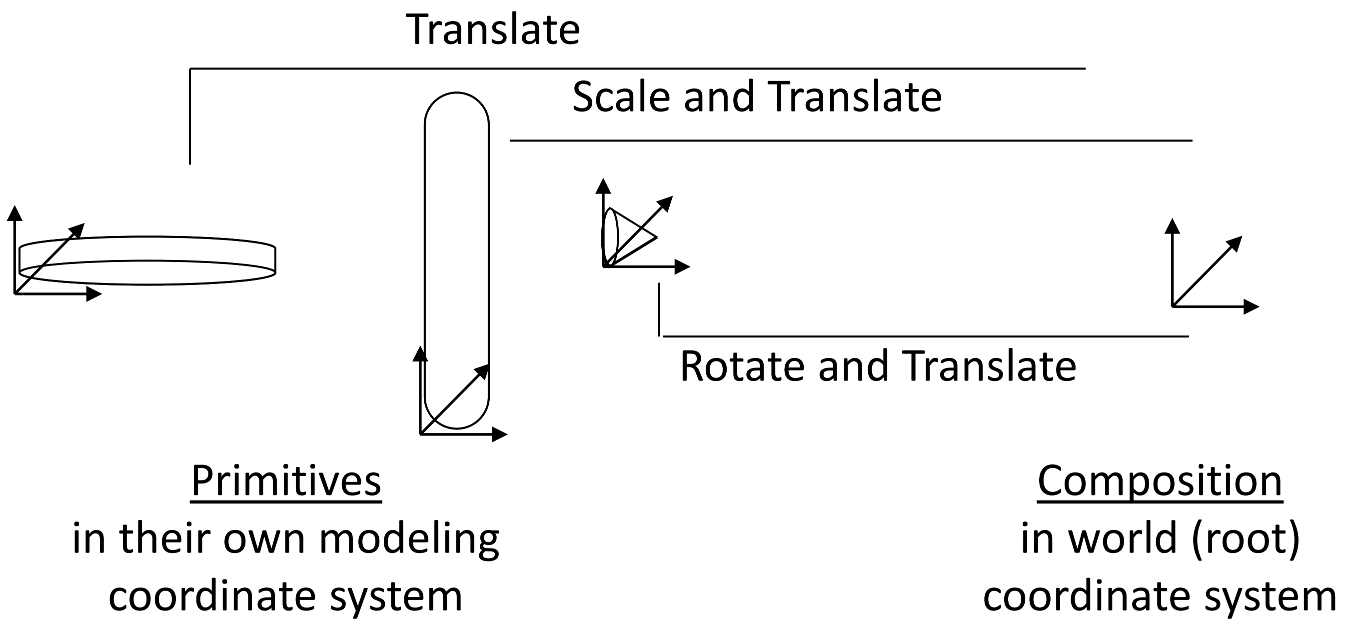
- Object to be modeled is (visually) analyzed, and then decomposed into collections of primitive shapes.
- Tree diagram provides visual method of expressing “composed of” relationships of model



tree diagram

- Such diagrams are part of 3D program interfaces (e.g., 3D Studio MAX, Maya)

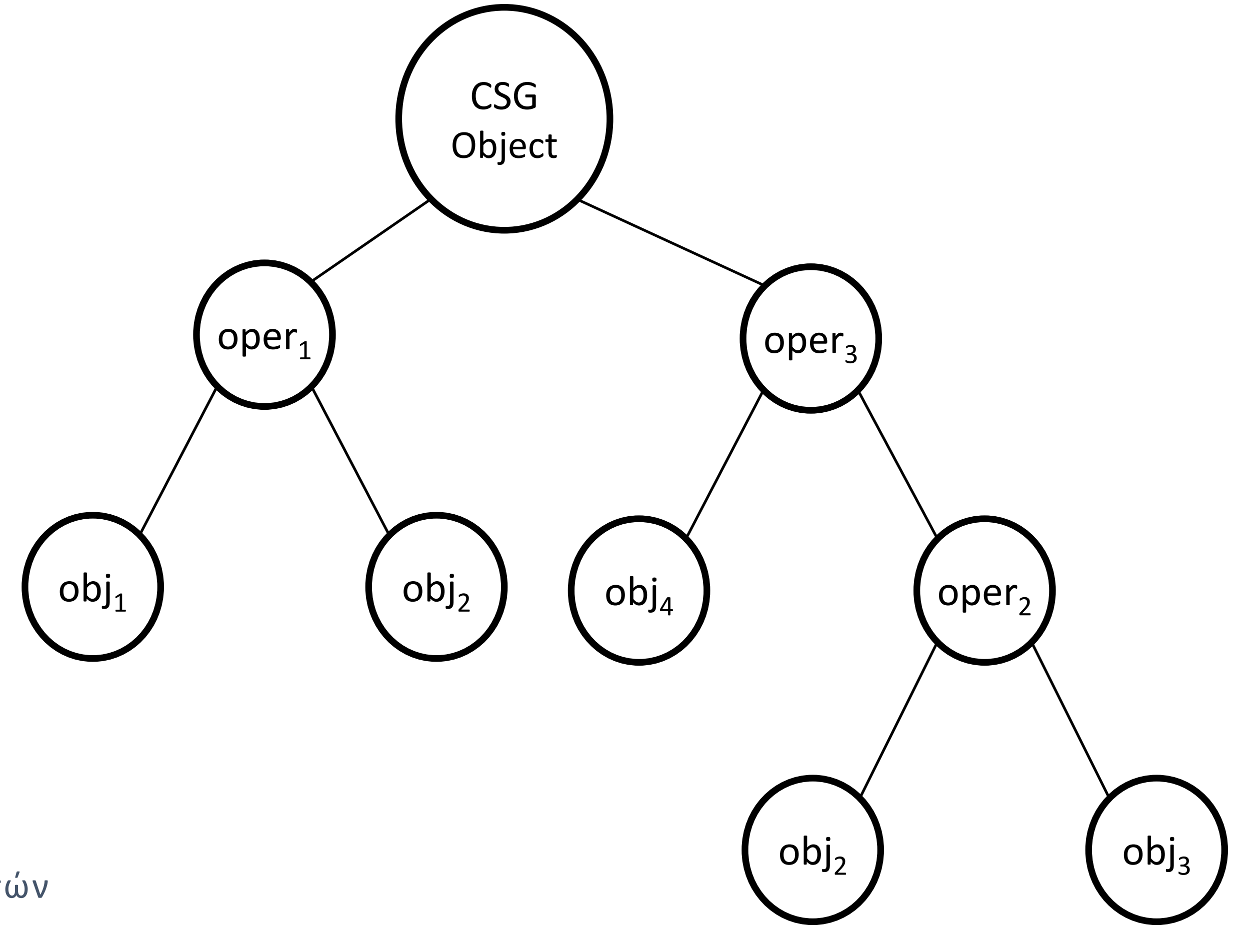
Composition of a geometric model: *Revision*



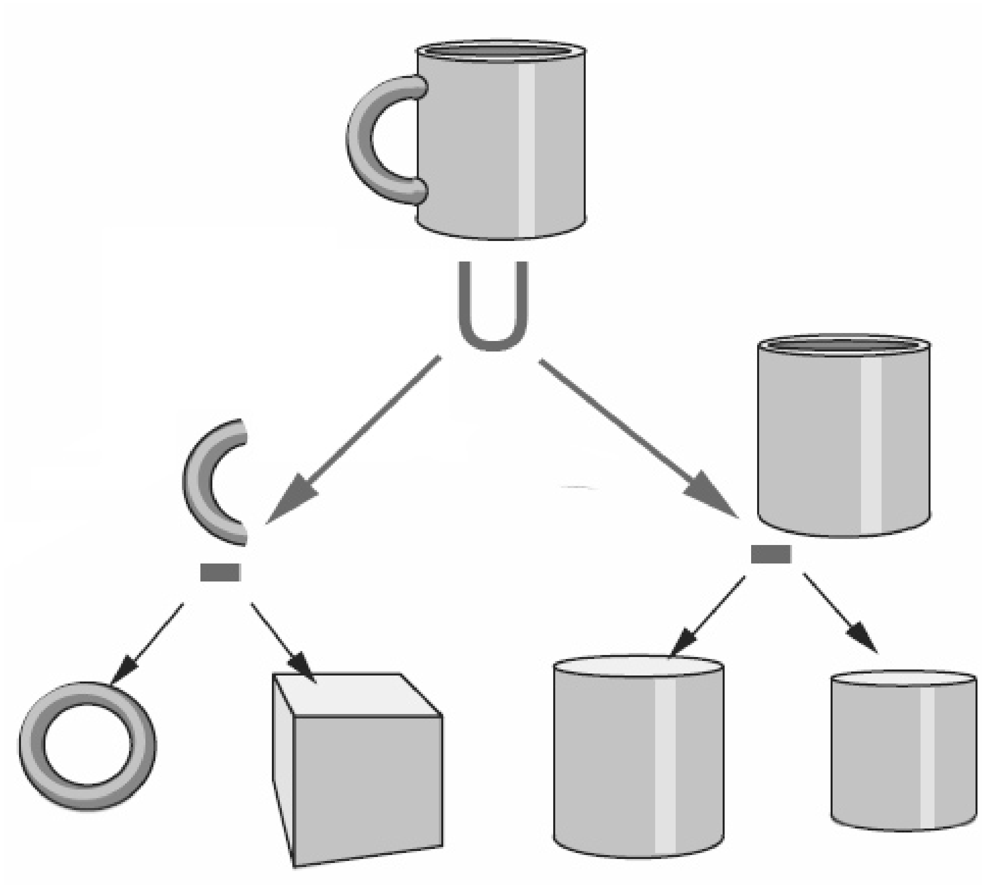
- Primitives created in decomposition process must be assembled to create final object. Done with **affine transformations**, T, R, S (as in above example). Order matters – these are not commutative!

Constructive Solid Geometry

- Στα CSG συνήθως ξεκινά με ένα μικρό σύνολο πρωτόγονων αντικειμένων, όπως κύβους, πυραμίδες, σφαίρες, και κώνους
  - Τα μοντέλα CSG αντιπροσωπεύονται συνήθως ως δέντρα



Constructive Solid Geometry



Next lesson

- Transfer from one coordinate system to another.
- Convert the scene so that it appears in front of the camera (viewing)