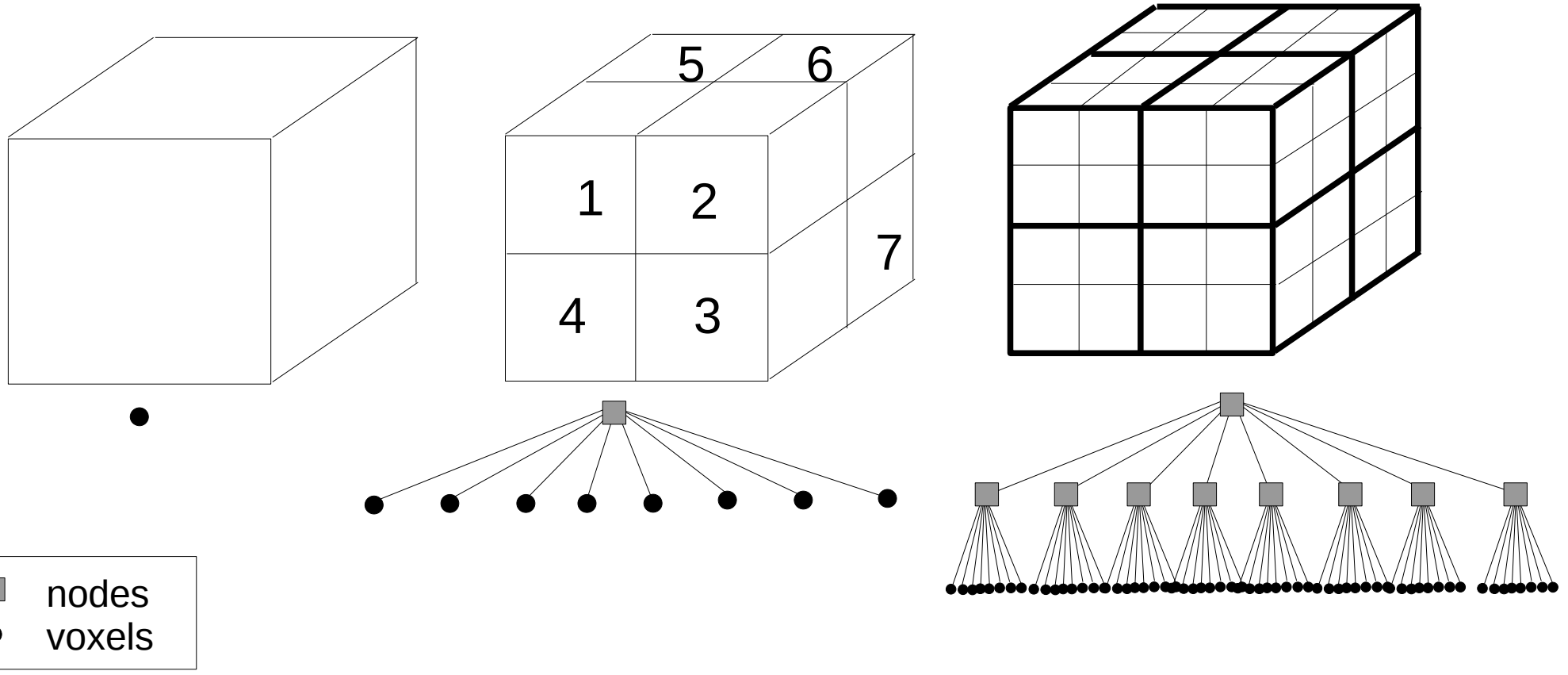
Octrees and meshes

Outline

- Octree representation
  - Regular octree
  - Surface representation / volume representation of a 3D object
- B-Rep models
- Brief history
- Meshes
  - Definition and properties
    - duality
    - closed meshes and 2-Manifold meshes
    - Euler formula
  - Internal representations
    - Representations optimised for memory space
    - Representations optimised for mesh exploration
  - OpenGL visualization

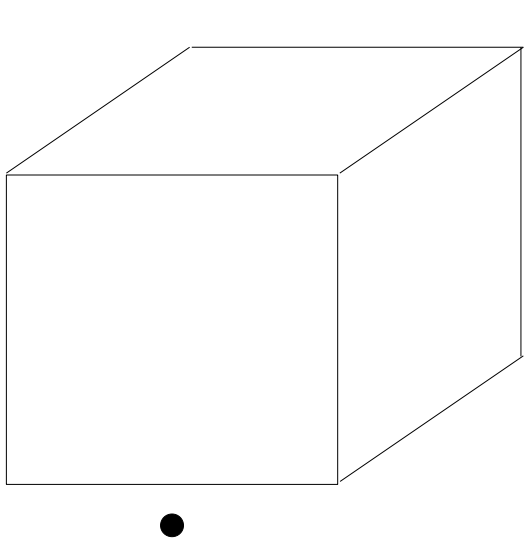
Regular octree

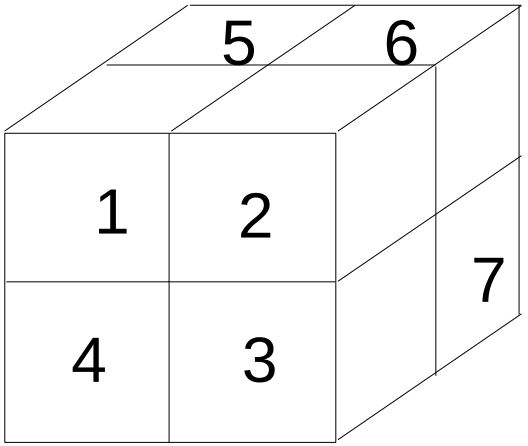
- An octree is a tree data structure in which each internal node has up to eight children. A regular octree recursively subdivides a cube in eight cubes of equal size. The leaves of an octree are called « voxels ».

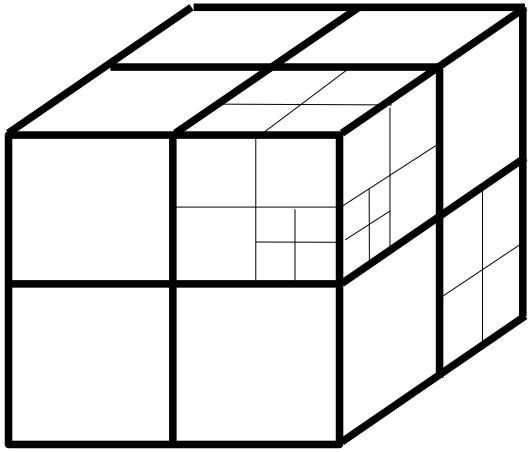


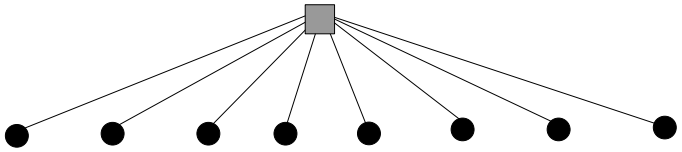
Adaptative octree

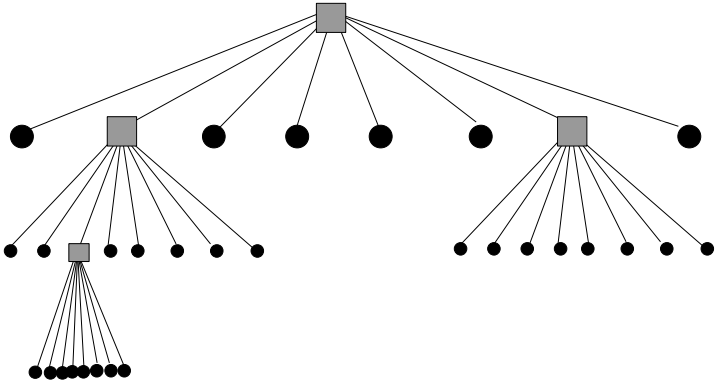
- In an adaptative octree, the nodes of the tree can have different lengths so as to sample space in an heterogeneous way.

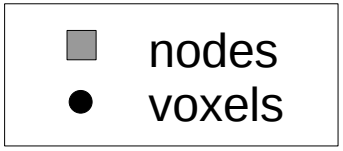






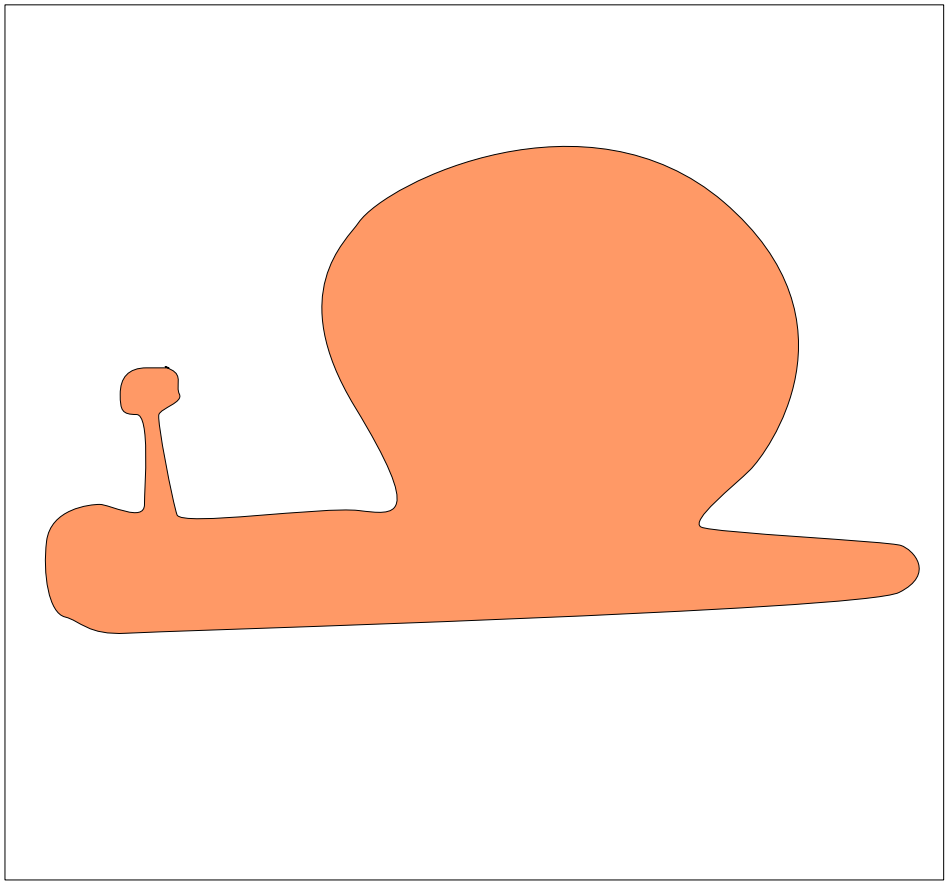


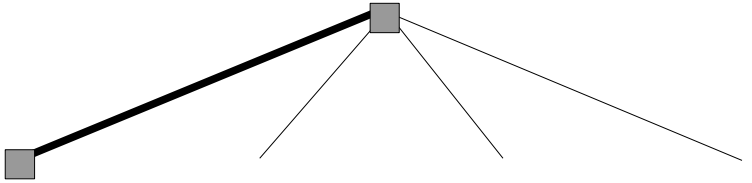




Example on a quadtree

- A quadtree is a tree in which each node has 4 children. It is the 2D equivalent of an octree
  - Draw the leaves of the adaptative quadtree representing the object below.
  - Write the « tree form » of the quadtree (develop the first node only).





Surface representation using an octree

- **Regular Octree :** we subdivide until sufficient precision is attained and for each voxel:
  - either the voxel does not intersect the surface: the corresponding leaf is empty (value 0),
  - or it does intersect the surface: the corresponding leaf is full(value1).

- **Adaptative octree:**
  - either the node does not intersect the surface : it is an empty leaf,
  - or it does intersect the surface : if we have enough precision, it becomes a full leaf. Otherwise, it is going to get subdivided

Volume representation using an octree

- **Regular octree :** we subdivide until sufficient precision is attained and for each voxel:
  - either it is inside the object (value 1 for instance)
  - either it is outside the object (value -1 for instance),
  - or it is on the boundary of the object (value 0).

- **Adaptative octree :**
  - either the node is on the boundary of the object : if we have sufficient precision it becomes a leaf (with value 0). Otherwise it gets subdivided.
  - Or it is inside or outside the object: it becomes either an “inside leaf” or an “outside leaf”.

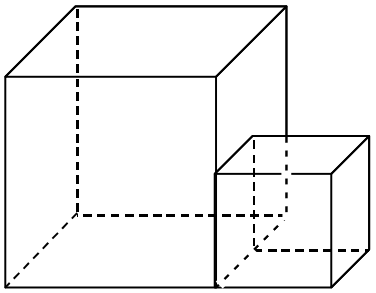
Octree : +/-

- Pros:
  - Hierarchical representation of an objet : it can be displayed at different resolutions
  - Possibility of a volume representation.
  - Easy to test the intersection with another object.
  - Simple construction and exploration (done recursively)

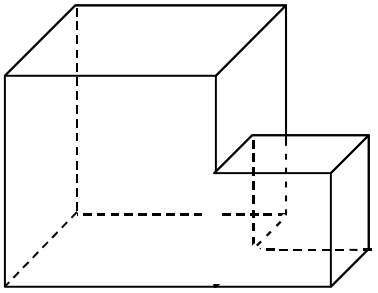
- Cons:
  - Surface visualisation of voxels is problematic.
  - Rendering can be costly for complex scenes.
  - Expensive to store.

B-Rep models

- Boundary-Representation
  - A model is represented by its boundary
  - No notion of volume
  - Used to represent solids



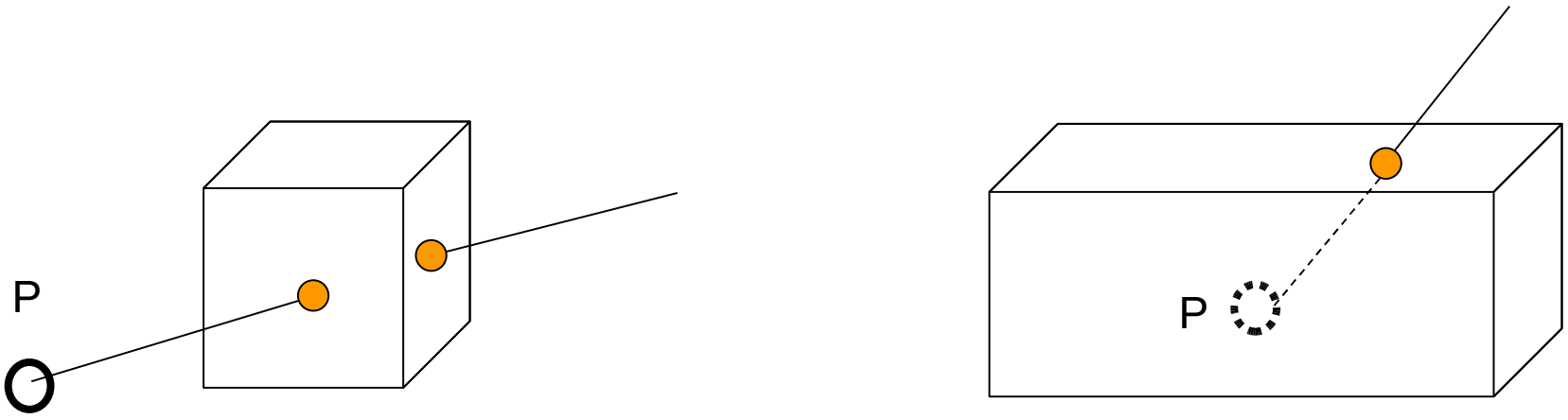
Ordinary B-Rep



Solid B-Rep

Solid B-Rep

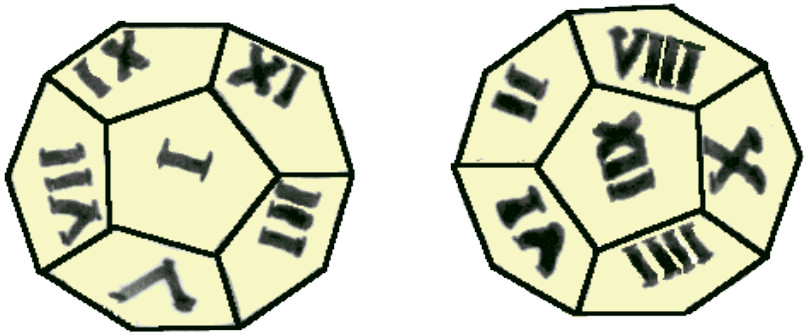
- Inside / Outside test on an object :



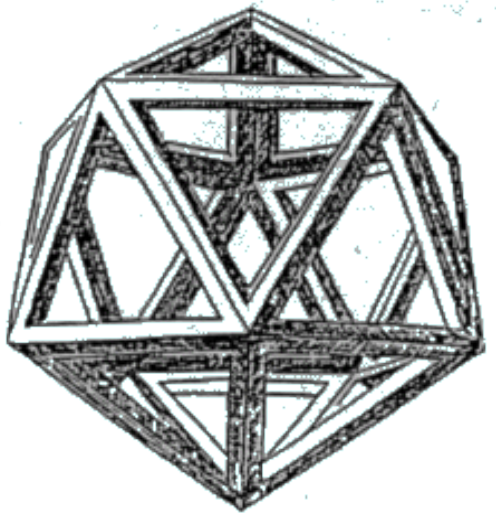
  - Even number of intersections : point P is outside the solid
  - Odd number of intersections : point P is inside the solid

Brief history

- 800 years Before Christ Era, Etruscans used to play games using regular dodecahedrons (12 faces).



- The pythagorians, 450 years BCE, already knew the cube and the tetrahedron.
- 400 years BCE, Platon discovered the 5 regular polyhedra: the tetrahedron, the cube, the octahedron, the dodecahedron and the icosahedron (20 faces).
  - In platon's philosophy, each one was associated with an element (the Fire, the Earth, the Air, the Water and the Universe).

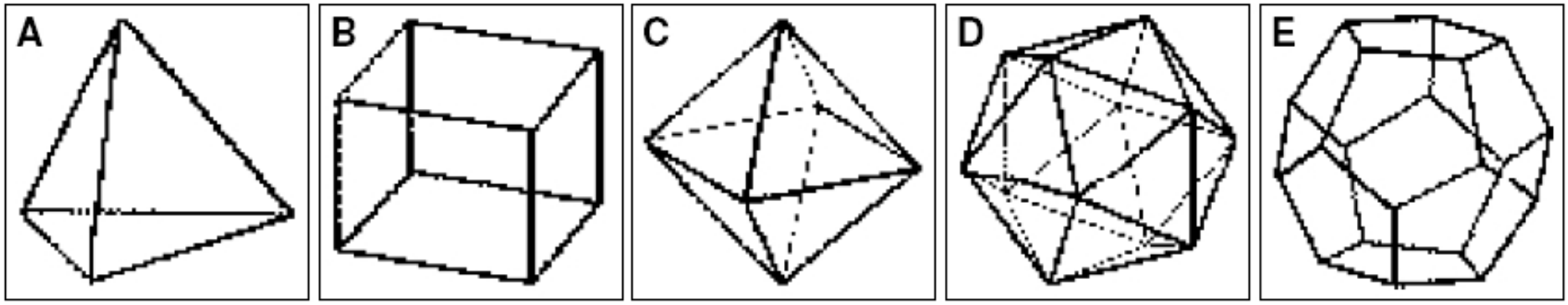


Icosaèdre dessiné, par Léonard de Vinci, pour le *De Divina Proportione* de Fra Luca Pacioli.

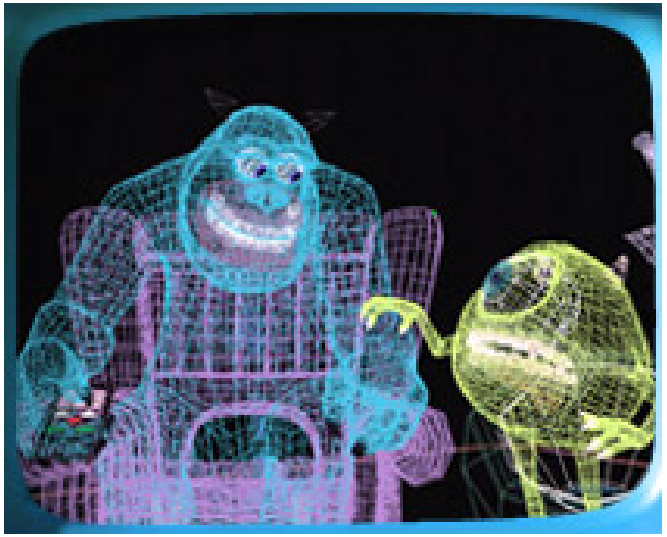
Platonic solids

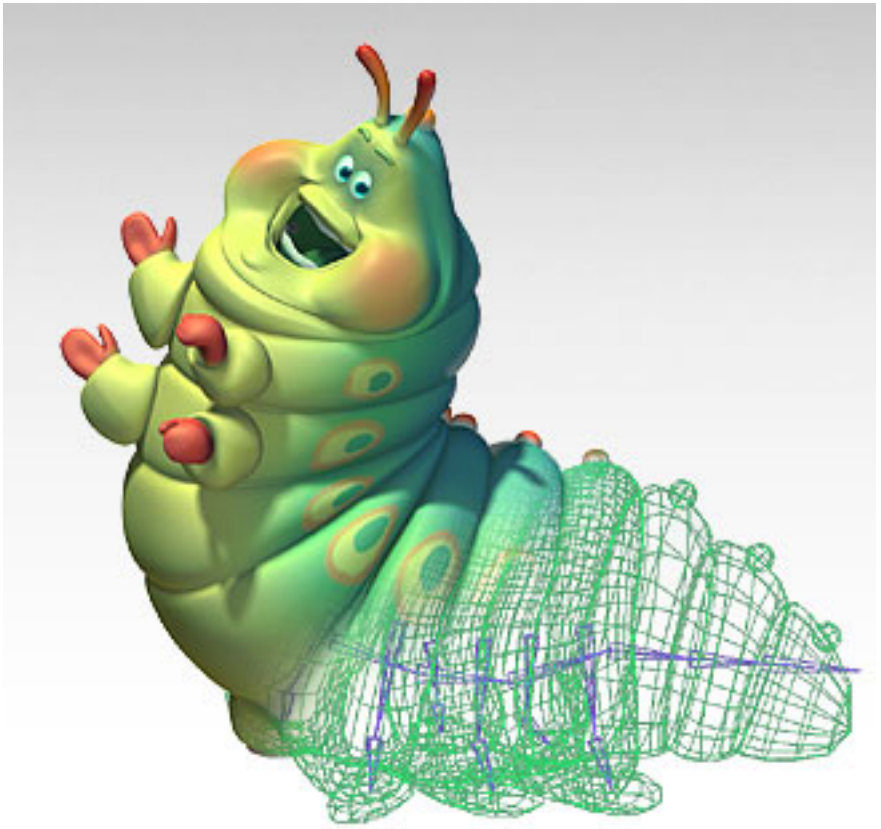
Regular polyhedron =

- Convex polyhedron having a circumsphere, whose faces are identical
- All its faces are regular convex polygons
- All its vertices have the same valency (same number of incident edges)



Meshes

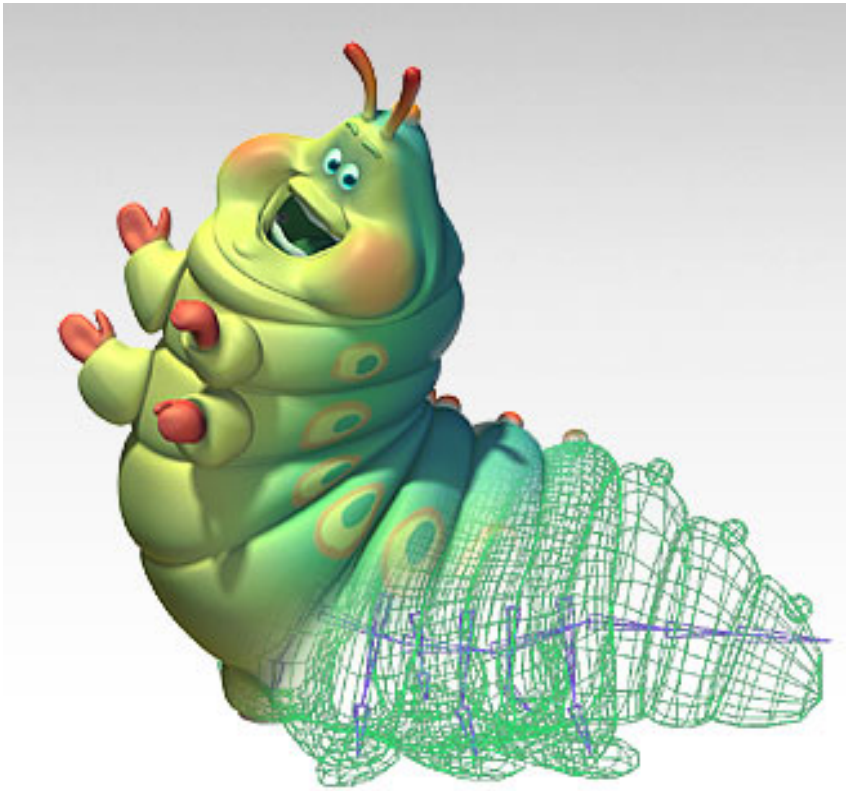




Meshes

- Surfaces are represented by polygons.
- Global continuity class is C^0 (normals are discontinuous along the edges).
- They define the geometry and the topology of a surface

- It is a standard structure used in the rendering of 3D scenes
- Their manipulation and visualization is optimised by today's graphic cards

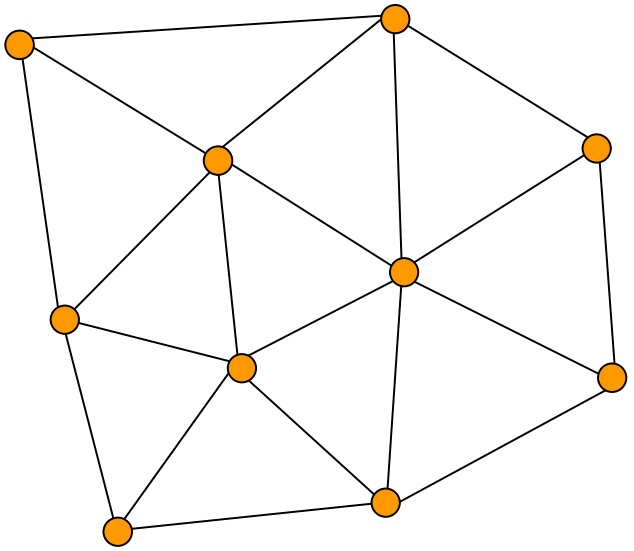


Topological properties of meshes

- Duality
- Closed mesh and 2-manifold mesh
- Euler formula

Duality

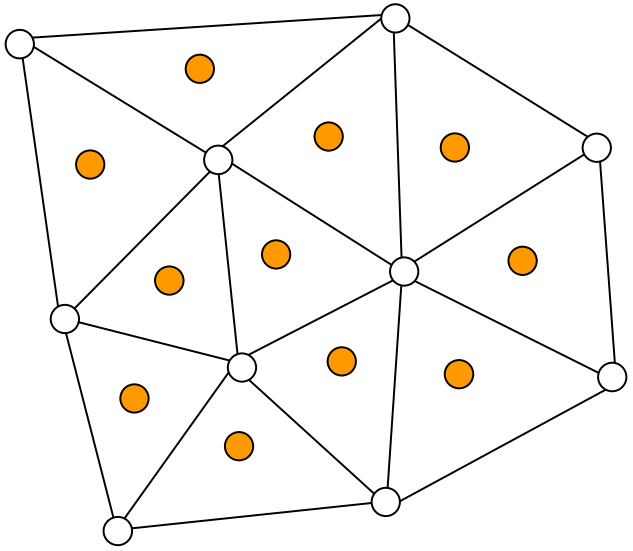
- Dual mesh: each entity of dimension k in the primal mesh is replaced by an entity of dimension (2-k) in the dual mesh:
  - Each face (dim 2) is replaced by a point (dim 0). Edges keep their dimension and points are replaced by faces.
  - To compute the geometry of the dual mesh, we place a vertex at the center of each face and connect two vertices with an edge if the faces they represent in the primal mesh share an edge.



Duality

- Dual mesh: each entity of dimension k in the primal mesh is replaced by an entity of dimension (2-k) in the dual mesh:
  - Each face (dim 2) is replaced by a point (dim 0). Edges keep their dimension and points are replaced by faces.
  - To compute the geometry of the dual mesh, we place a vertex at the center of each face and connect two vertices with an edge if the faces they represent in the primal mesh share an edge.

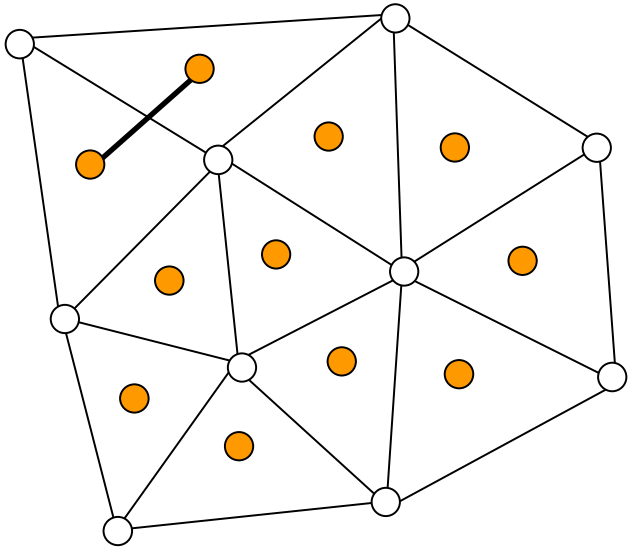
Faces are replaced by their center



Duality

- Dual mesh: each entity of dimension k in the primal mesh is replaced by an entity of dimension (2-k) in the dual mesh:
  - Each face (dim 2) is replaced by a point (dim 0). Edges keep their dimension and points are replaced by faces.
  - To compute the geometry of the dual mesh, we place a vertex at the center of each face and connect two vertices with an edge if the faces they represent in the primal mesh share an edge.

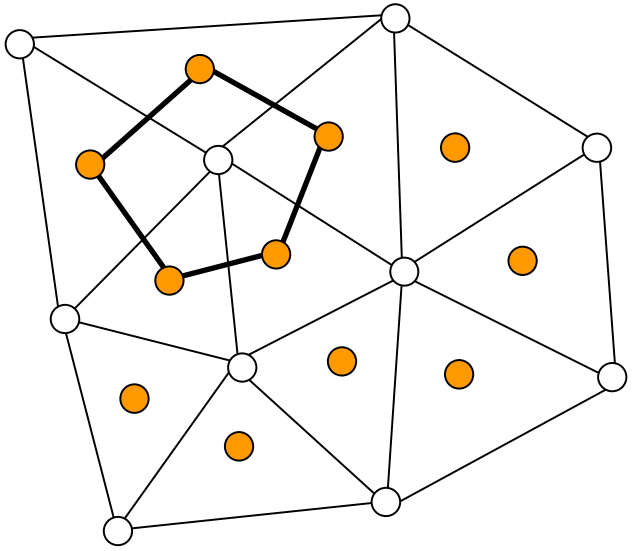
Edges are “flipped”



Duality

- Dual mesh: each entity of dimension k in the primal mesh is replaced by an entity of dimension (2-k) in the dual mesh:
  - Each face (dim 2) is replaced by a point (dim 0). Edges keep their dimension and points are replaced by faces.
  - To compute the geometry of the dual mesh, we place a vertex at the center of each face and connect two vertices with an edge if the faces they represent in the primal mesh share an edge.

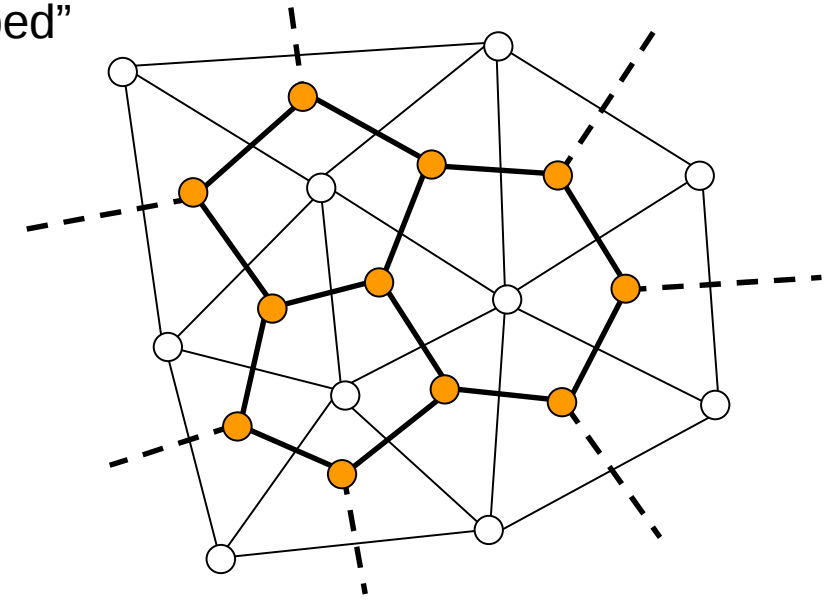
Edges are “flipped”



Duality

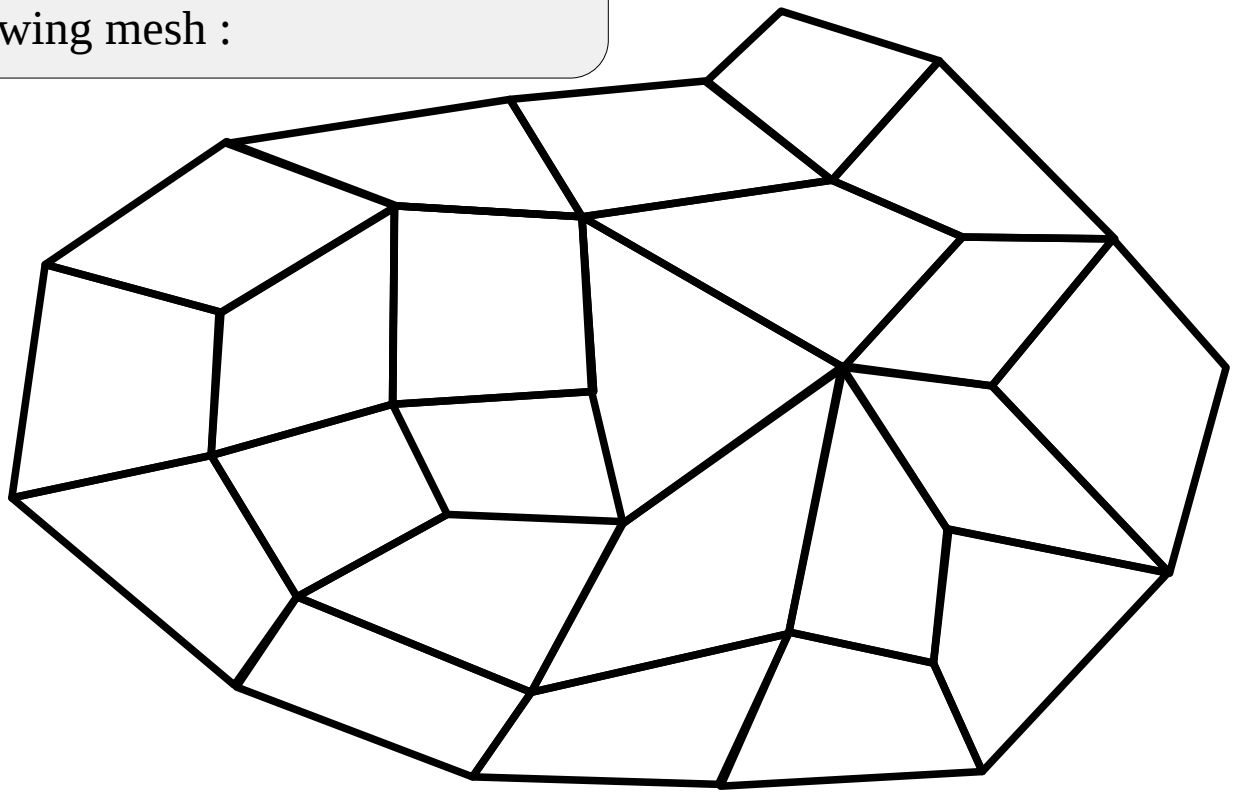
- Dual mesh: each entity of dimension k in the primal mesh is replaced by an entity of dimension (2-k) in the dual mesh:
  - Each face (dim 2) is replaced by a point (dim 0). Edges keep their dimension and points are replaced by faces.
  - To compute the geometry of the dual mesh, we place a vertex at the center of each face and connect two vertices with an edge if the faces they represent in the primal mesh share an edge.

Edges are “flipped”



Duality: example

- Draw the dual mesh of the following mesh :



- What do you notice at the faces ?
- What about the dual mesh of the dual mesh ?

Closed mesh

A mesh is a closed mesh if it divides space into two sets of points:

A set inside the mesh

A set outside the mesh

In other words, a closed mesh does not have any border

2-manifold

- A closed mesh is 2-manifold (or manifold) if the object it represents is « manufacturable ».

- By definition, a mesh is 2-manifold if the condition below is satisfied:

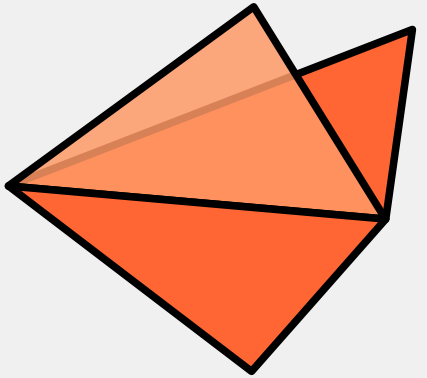
  - Local disc property: at each point of the mesh, we can find a sphere of radius $\varepsilon > 0$ such that the intersection of that sphere and the mesh is homothetic to a circle.

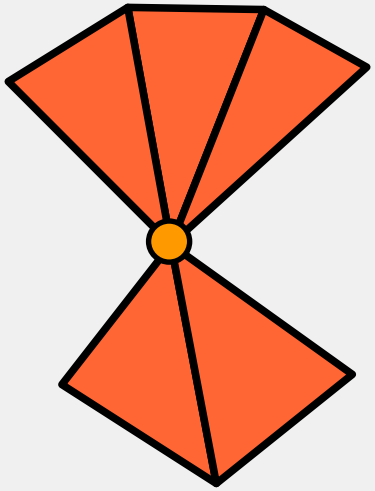
- An open mesh can be 2-manifold. At the border, the intersection between the sphere and the mesh must be homothetic to a half-circle.

Example : manifold / non-manifold ?

On triangular meshes:

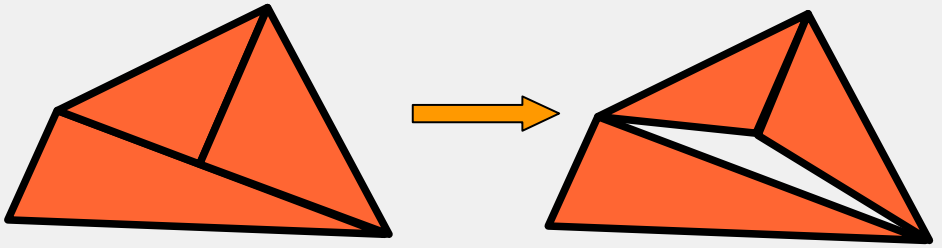
- An edge shared by more than 2 triangles ?



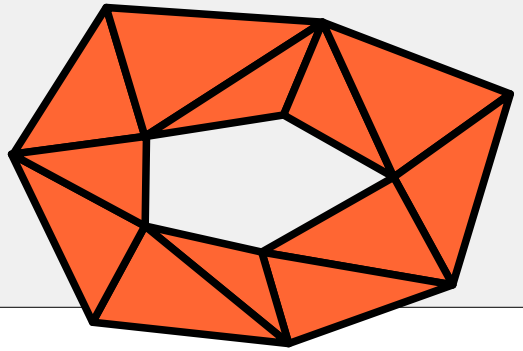


- A vertex shared by two disjoint sets of triangles ?

- A T-junction (« crack » problem) ?



- A hole in the mesh?



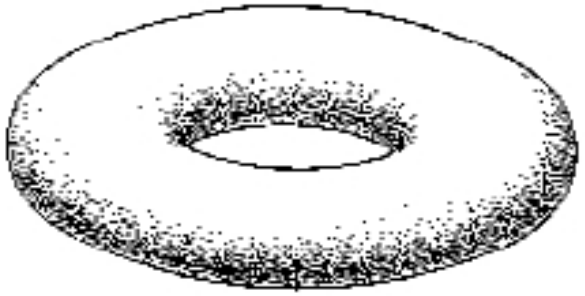
Euler formula (1752)

- It is a relation between every entity of the mesh (faces, edges, vertices):

$$V - E + F = 2\,(1-g)$$

Where V is the number of vertices, E is the number of edges and F is the number of faces. g is the genus of the object = the number of holes (handles) in a closed mesh.

genus 0



genus 1

genus 2

Triangular meshes

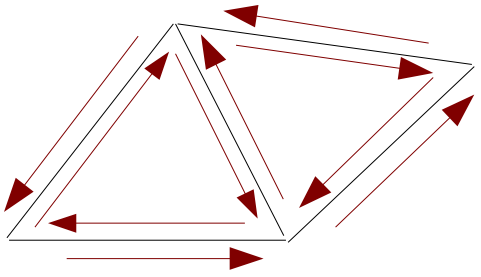
Specific properties from Euler's formula:

$$V - E + F = 2\,(1-g) = c$$

where c is a constant usually neglectible in front of V, E or F. Thus we can write :

$$V - E + F = 0$$

It is difficult to establish a relationship between those entities because an edge is shared by two triangles. We can introduce the notion of an half-edge. A face is composed of 3 half-edges and an edge is composed of 2 half-edges:



$He = 3\,F$

$He = 2\,E$

$F = 2V$

$He = 6V$

Exemples: Euler's formula

Compute the number of faces **F** from the number of vertices **V**, then the average number of outgoing Half-edges **He** for each vertex. Then deduce the shape of a regular mesh composed of :

- quads

- hexagons

Data structures for meshes

Choose a representation adapted to the operations we would like to perform on the mesh :

- Either a compact representation (save memory space)
- Or a representation optimized for mesh exploration

Triangles list

- For each triangle, we store the coordinates of its 3 vertices
  - Naive approach
  - 4 bytes per coordinate (a float)
  - 9 coordoninates per face
  - Number of faces is about twice the number of vertices
- For a mesh composed of V vertices, we thus need $4 \times 9 \times 2 \times V = 72 \times V$ bytes.

$x_0,y_0,z_0 \quad x_1,y_1,z_1 \quad x_2,y_2,z_2$

$x_5,y_5,z_5 \quad x_1,y_1,z_1 \quad x_{12},y_{12},z_{12}$

$x_9,y_9,z_9 \quad x_0,y_0,z_0 \quad x_{1025},y_{1025},z_{1025}$

...

Shared vertices

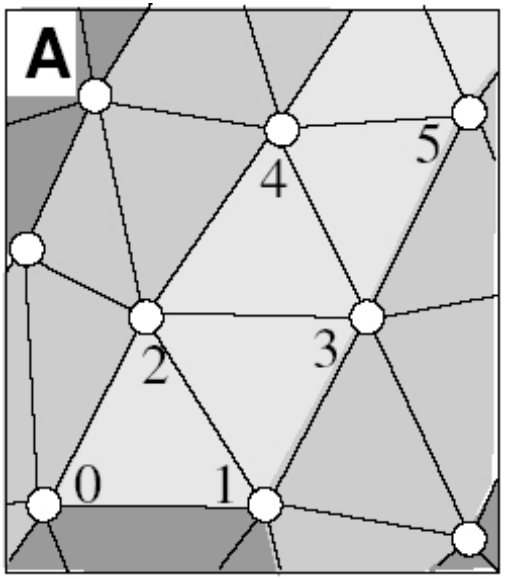
- First we store the vertices list
- Then a triangle is given by the indices of its 3 vertices
- Usually, the file storing the mesh begins with the number of vertices and the number

Vertices list	Triangles list
x_0,y_0,z_0	0 1 2
x_1,y_1,z_1	5 1 12
x_2,y_2,z_2	9 0 1025
…	…

How many bytes stored per vertex? (36)

Triangle strips

- Topology is implicitely stored by the order the vertices appear in the file
- Each vertex is explored 2 times (one vertex belongs to 2 strips or appears twice in a strip)
- If the mesh is defined by a single strip, what is the memory space occupancy ?



Actually, a mesh is usually composed of many strips and an additionnal 20 bytes is needed for each new strip (storage of the 2 first vertices)

Per face structure

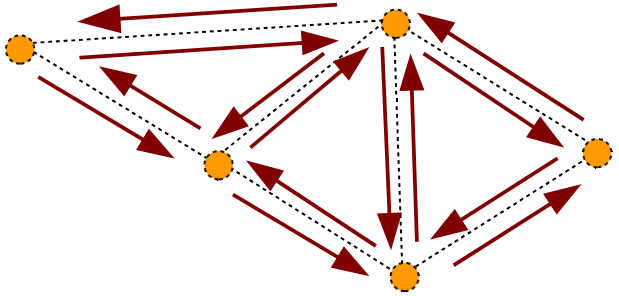
- Each face has pointers to its vertices and to adjacent faces
- Each vertex has a pointer to one of its faces

- No direct access to the edges

- Bytes per vertex? (64)

Per half-edge structure

- Each edge is sliced into two half-edges of opposite directions



- For each half-edge, we store a pointer to the opposite half-edge, a pointer to the vertex it points to, a pointer to the face it belongs to, and a pointer to the next half-edge.
  - Each vertex has a pointer to one of its outgoing half-edges. Each face has a pointer to one of its half-edges.
  - Very useful to explore a mesh in various ways
  - Bytes per vertex ? (120)

Visualization of a mesh using OpenGL

- Object construction :
  - The list of its vertices

    GLfloat vertices[] = {x_0, y_0, z_0,

    ...

    x_n, y_n, z_n};
  - The list of its attributes (normals for instance)

    GLfloat normals[] = {n_{x0}, n_{y0}, n_{z0},

    ...

    n_{xn}, n_{yn}, n_{zn}};
  - The list of its indices

    GLuint index[] = {0, 1, 2,

    5, 0, 4,

    . . . };

Visualization of a mesh using openGl

- Different kinds of primitives:
  - GL_TRIANGLES : the indices in the list are considered by groups of 3 elements. Each triplet forms a triangle .
  - GL_QUADS : the indices in the list are considered by groups of 4 elements. Each group forms a quad.
  - GL_TRIANGLE_STRIP : the indices in the list are considered in their given order so as to form a triangle strip. In the indices list, it is possible to use a special value to separate different strips,
  - GL_QUAD_STRIP : same, but with quads instead of triangles,
  - …

OpenGL rendering

// Activating rendering mode using vertex arrays

glEnableClientState (GL_VERTEX_ARRAY);

glEnableClientState (GL_NORMAL_ARRAY); *// if we want to set the normal for each vertex*

glEnableClientState (...); *//if we want to set other attributes of the vertices (color...)*

// Setting pointers to the different arrays

glVertexPointer (3, GL_FLOAT, 0, sommets);

glNormalPointer (GL_FLOAT, 0, normales); // if we use normals

// … + other possible arrays (color,…)

// Drawing triangles:

glDrawElements (GL_TRIANGLES, nb_index, GL_UNSIGNED_INT, index);

// Deactivating the rendering mode

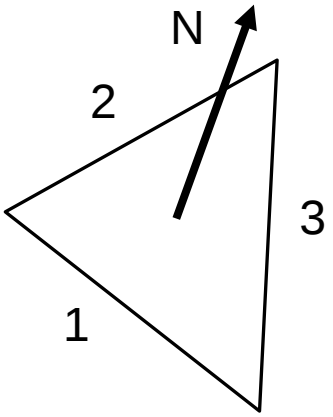
glDisableClientState (……..);

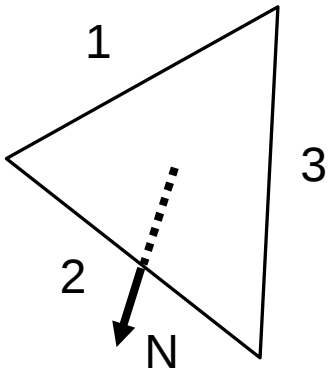
Further optimisation

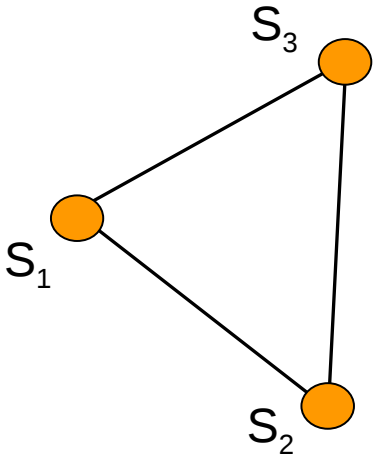
- Vertex arrays are a simple way to visualize meshes in OpenGL.
- A most common and efficient way to visualize meshes is to use OpenGL's Vertex Buffer Objects (VBO). VBOs are vertex arrays stored in the memory of the GPU. Thus their access from the GPU is faster, hence a faster rendering.
- VBOs will be studied in a next lesson...

Normal to a face

- The normal to a face can be obtained by computing the cross product of two edges from that face







$$\vec{N} = \frac{\vec{S_1S_2} \times \vec{S_1S_3}}{\|\vec{S_1S_2} \times \vec{S_1S_3}\|}$$

- When storing a mesh, one has to be careful about normals orientation. The vertices composing each face must be stored in the same order...
- Normals point to the outside of a closed object. They are needed when computing the lighting of the object.

Normal at a vertex

- Naive approach: average the normals to the faces that contain the vertex.

$$\vec{N}^s = \frac{\sum \vec{N}_i^f}{\|\vec{N}_i^f\|}$$

- Advanced approach : the sum is ponderated

$$\vec{N}^s = \frac{\sum a_i \times \vec{N}_i^f}{\|\sum a_i \times \vec{N}_i^f\|}$$

  - For instance, we can set a_i as the area of triangle i

- Either way, we obtain an approximation of the normal to the surface at the vertex

Tutorial: load a mesh from a file

- Write an algorithm to load a mesh from a file. The mesh is stored using the shared vertices technique.
  - The first two values in the file are the number of vertices followed by the number of faces. The next values are the coordinates of the vertices then the list of the indices.
  - The mesh must be loaded so that the data structures used can be used by OpenGL to display the mesh (Vertex Array).
  - We would like to compute the normals at each vertex so that the object can be correctly lightened