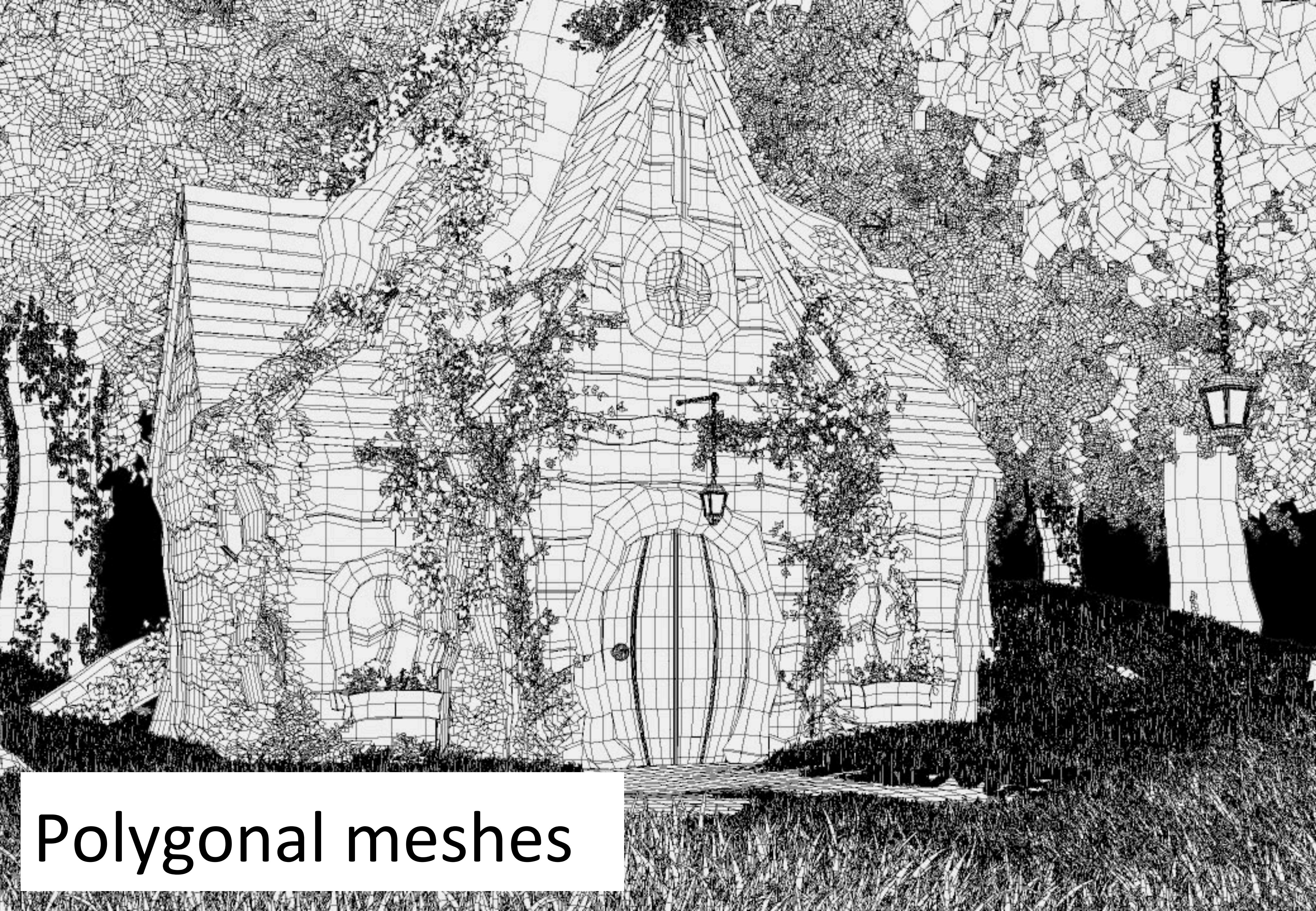
Planes, Polygons and Objects

Overview

- Polygons
- Planes
- Creating an object from polygons

No more spheres

- Most things in computer graphics are not described with spheres!
- **Polygonal meshes** are the most common representation
- Look at how polygons can be described and how they can used in ray-casting


Polygonal meshes

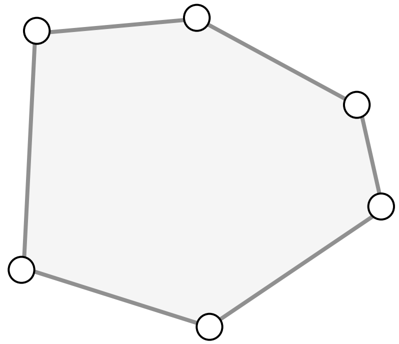
Polygons

- A polygon (face) P is defined by a series of points

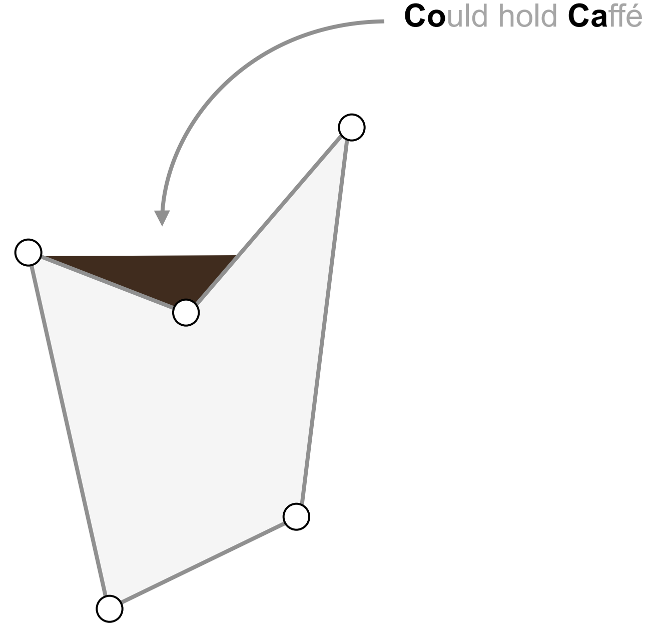
$$P = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_{n-1}, \mathbf{p}_n\}$$
$$p_i = \left(x_i, y_i, z_i\right)$$

- The points must be co-planar
- 3 points define a plane
- Further point need not lie on that plane

Convex vs. Concave



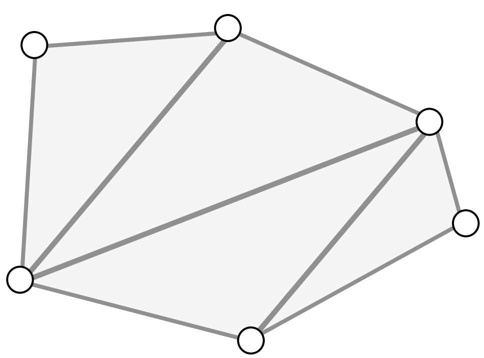
Convex

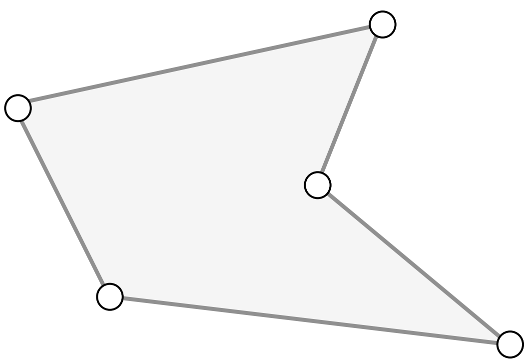


Concave

Convex, Concave

- CG people dislike concave polygons
- CG people would prefer triangles!!
  - Easy to break convex object into triangles, hard for concave



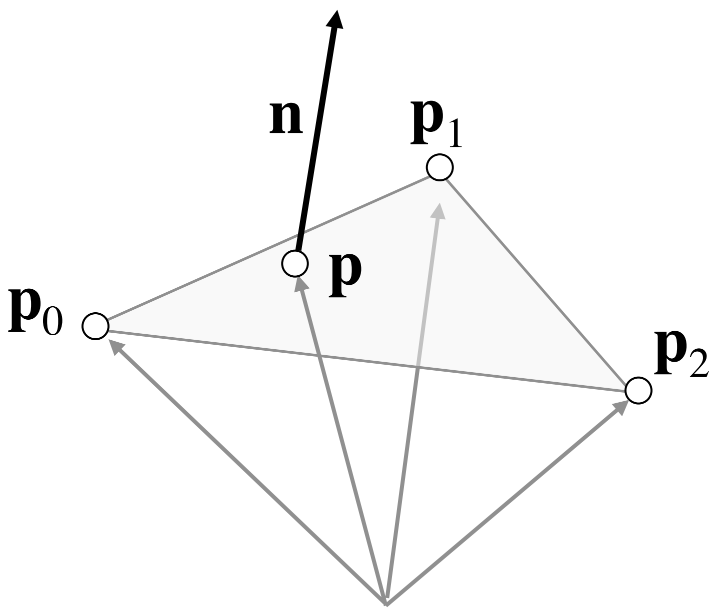


Equation of a plane

$$ax + by + cz = d$$

- a, b, c, d are constants that define a unique plane
- x, y, z form a vector $\mathbf{p}$

Deriving a, b, c, d (1)



- The cross product

$$\mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)$$

  defines a **normal** to the plane
- There are two normals (they are opposite)
- Vectors in the plane are **all** orthogonal to the plane normal vector

Deriving a, b, c, d (2)

- Every $\mathbf{p}\text{-}\mathbf{p}_0$ is normal to $\mathbf{n}$, therefore

$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$$

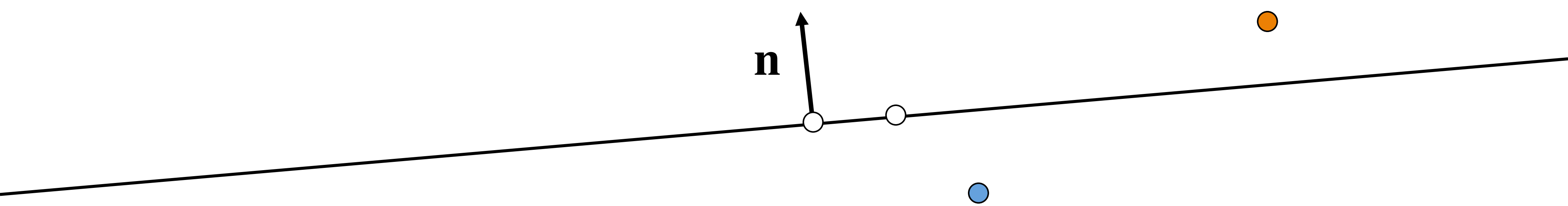
$$\mathbf{n} \cdot \mathbf{p} = \mathbf{n} \cdot \mathbf{p}_0$$

- If $\mathbf{n} = (a, b, c)$ and $\mathbf{p} = (x, y, z)$ and $d = \mathbf{n} \cdot \mathbf{p}_0 = n_1 x_0 + n_2 y_0 + n_3 z_0$

$$ax + by + cz = d$$

Half-space

- A plane cuts space into 2 **half-spaces**
- Define $$l(x, y, z) = ax + by + cz - d$$
- If $l(p) = 0$ point on plane
- If $l(p) > 0$ point in **positive** half-space
- If $l(p) < 0$ point in **negative** half-space



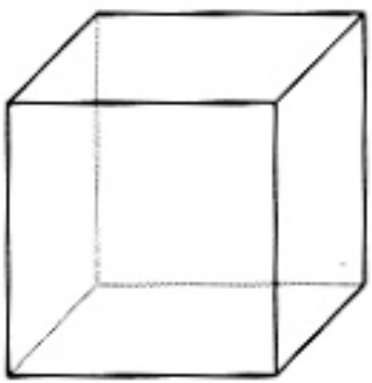
Polyhedra

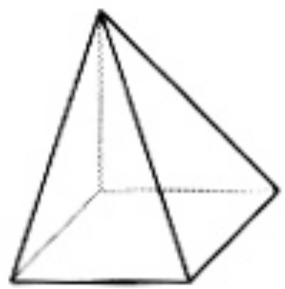
Polyhedra

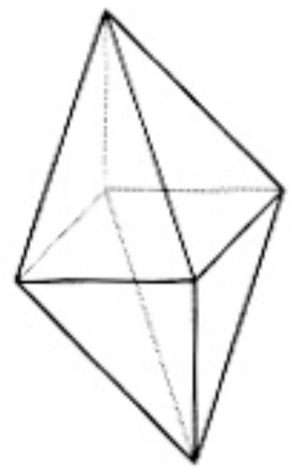
- Polygons are often grouped to form **polyhedra**
  - Each **edge** connects 2 vertices
  - Each **vertex** joins 3 (or more) edges
  - No faces intersect
  - Ideally: should be **manifold**
    - One vertex has one loop of polygons/edges
    - Each edge has one or two polygons

Polyhedra

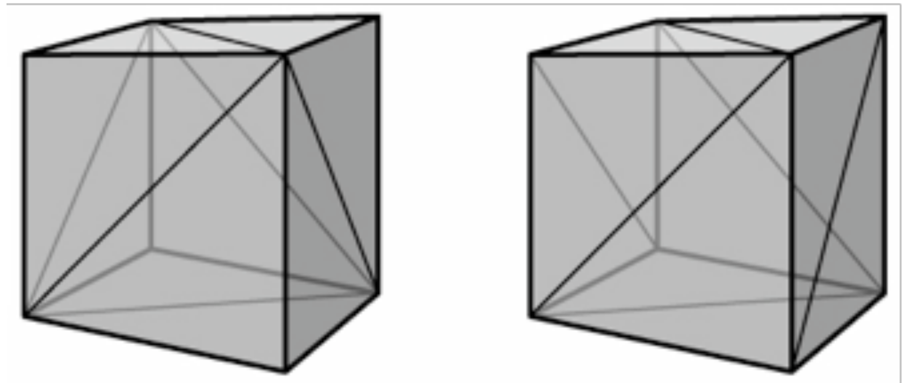
- $|V| - |E| + |F| = 2$
  - For cubes, tetrahedra, cows, etc...



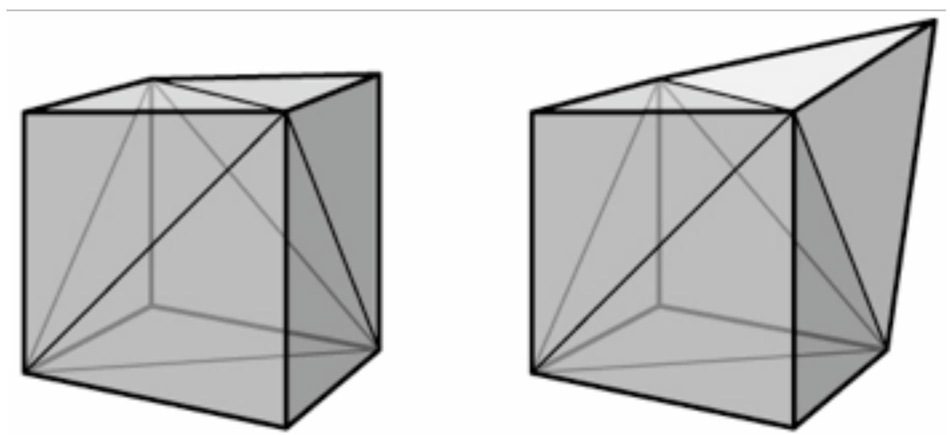




Topology / Geometry

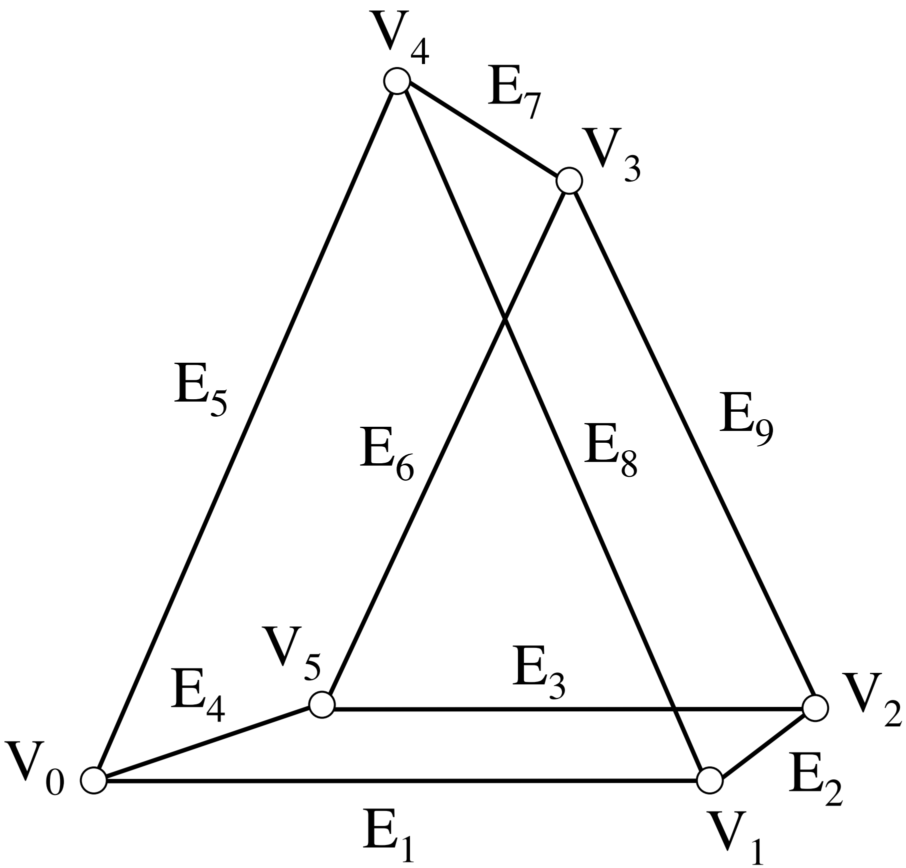


Same geometry, different mesh topology



Same topology, different geometry

Example polyhedron



$$F_0 = \{V_0, V_1, V_4\}$$
$$F_1 = \{V_5, V_3, V_2\}$$
$$F_2 = \{V_1, V_2, V_3, V_4\}$$
$$F_3 = \{V_0, V_4, V_3, V_5\}$$
$$F_4 = \{V_0, V_5, V_2, V_1\}$$

$|V|=6, |F|=5, |E|=9$

$|V| - |E| + |F| = 2$

Representing polyhedra

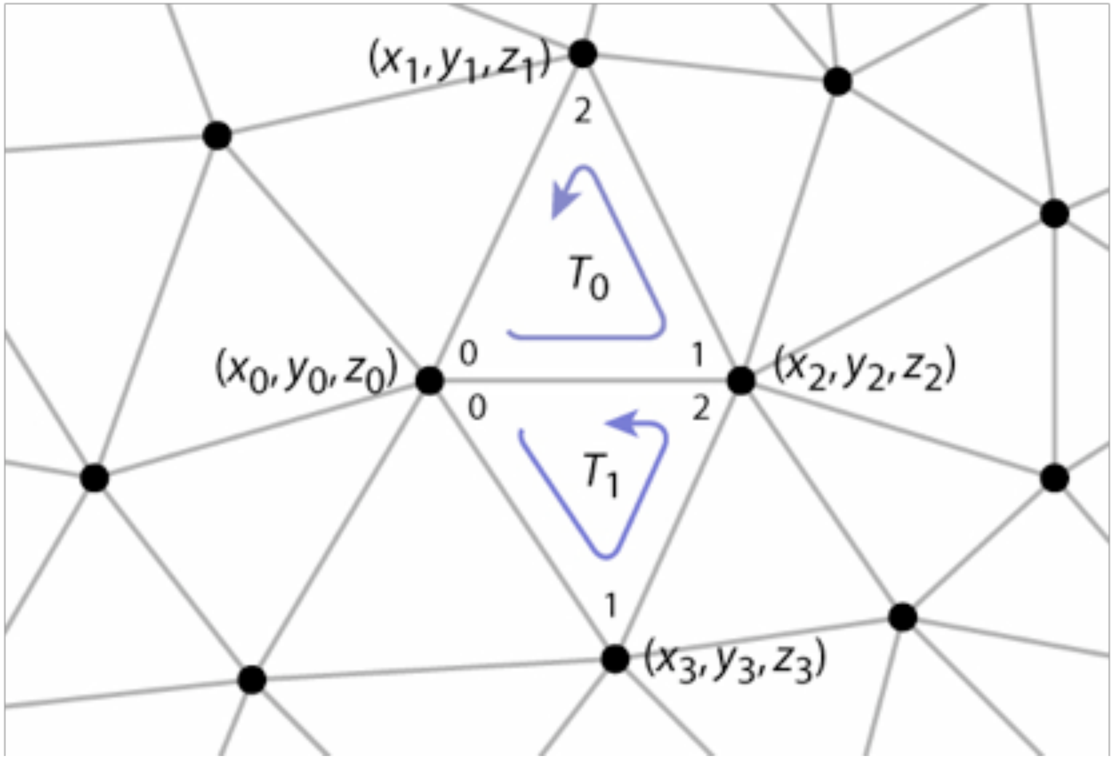
1. Separate polgyons
   – Replicate all coordinates

2. Index face set
   – Share vertices

3. Winged-edge data structure
   – General and space-efficient

Separate polygons

	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
	⋮	⋮	⋮



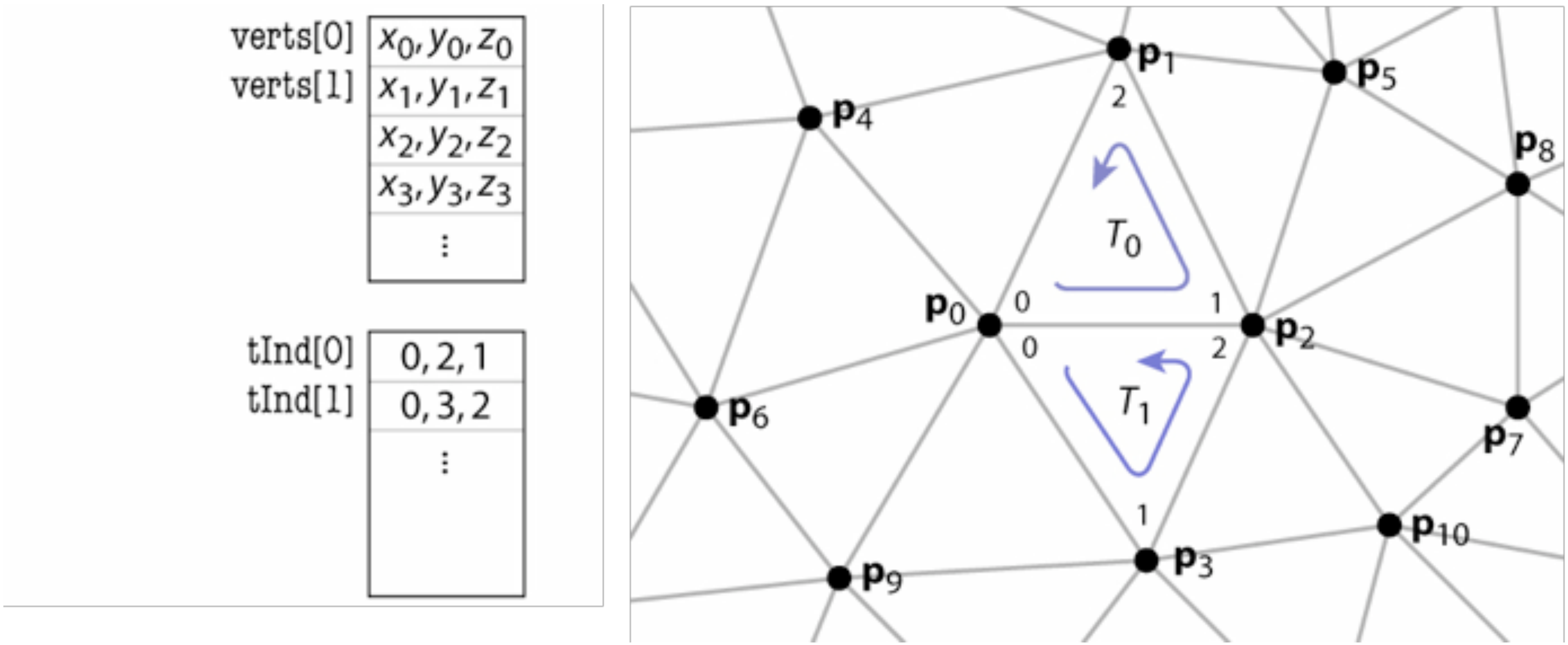
Separate polygons

- Exhaustive (array of vertex lists)

```
faces[0] = (x0,y0,z0),(x1,y1,z1),(x3,y3,z3);
faces[1] = (x2,y2,z2),(x0,y0,z0),(x3,y3,z3);
…
```

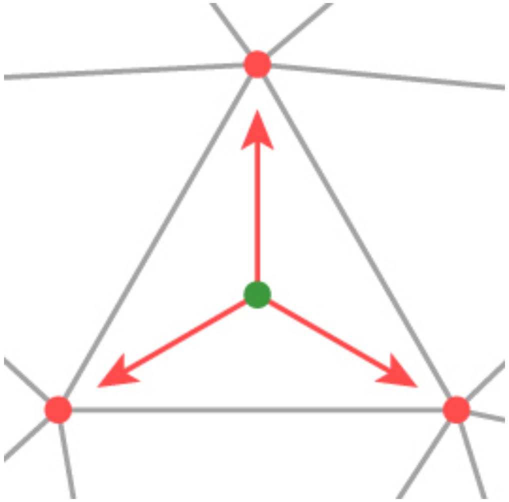
- Problems
  - Very wasteful
    - same vertex appears at 3 (or more) points in the list
  - Cracks due to rounding errors
  - Difficult to find neighboring polgyons

Indexed face set

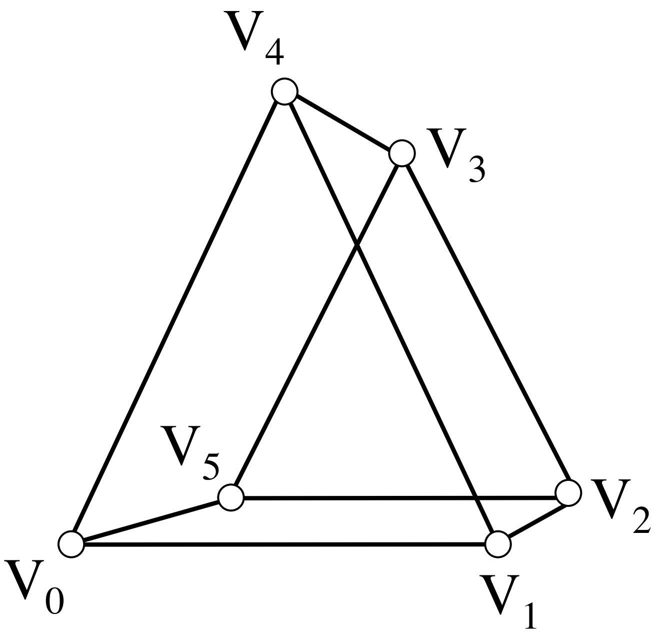


Indexed face set

- Store each vertex once
- Each polygon points to its vertices
  - Vertex array
    ```
    vertices[0] = (x0, y0, z0);
    vertices[1] = (x1, y1, z1);
    …
    ```
  - Face array (list of indices into vertex array)
    ```
    faces[0] = {0, 2, 1};
    faces[1] = {2, 3, 1};
    ...
    ```



Vertex order matters



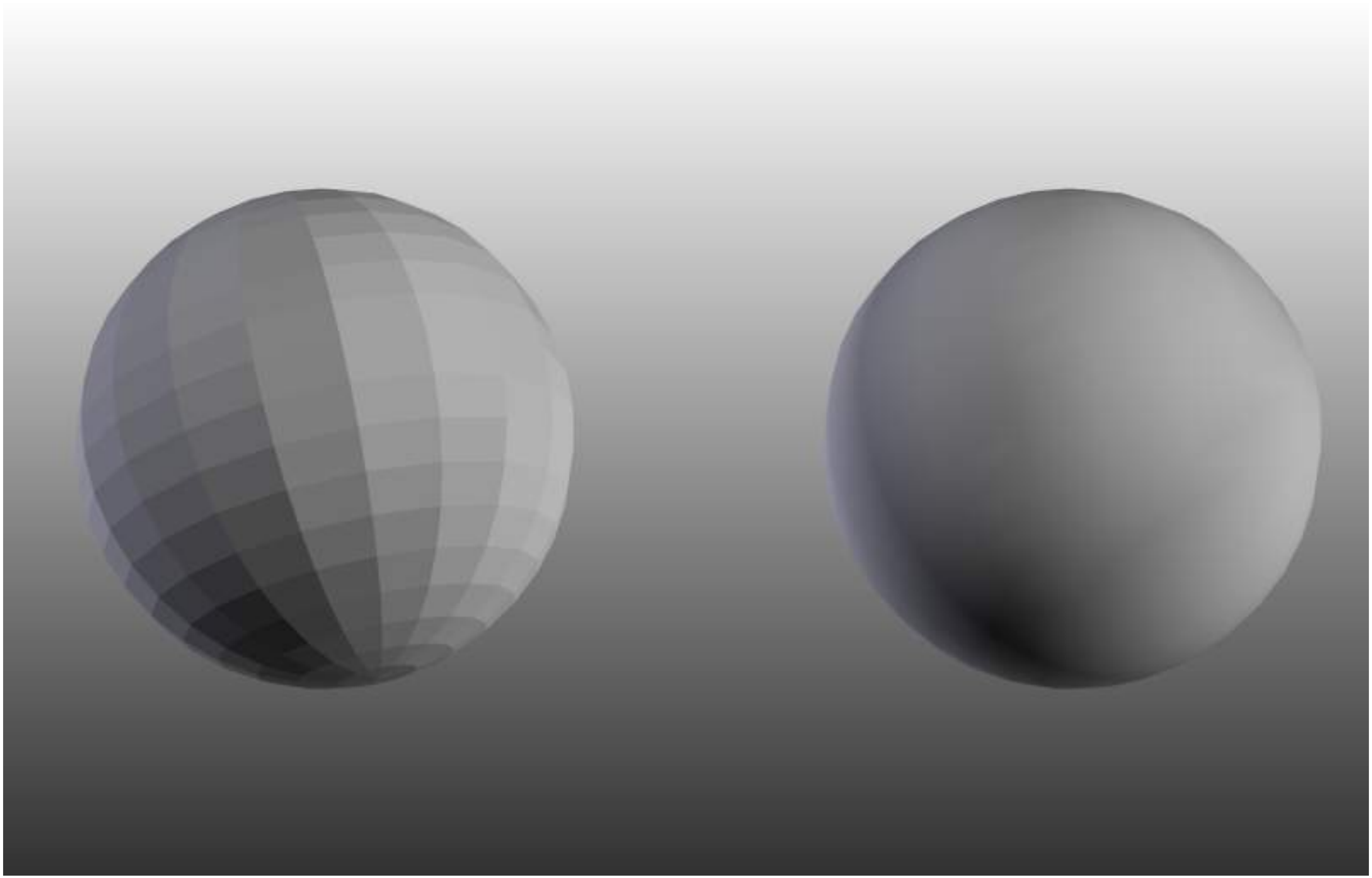
- Polygon V_0, V_1, V_4 is NOT equal to V_0, V_4, V_1
- Normal points in different directions
- Usually a polygon is only visible from points in its positive half-space
- Known as:

  back-face culling

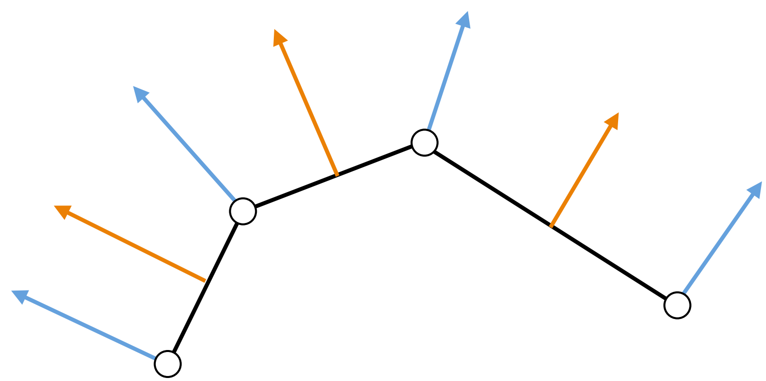
Indexed face set issues

- Even indexed face set wastes space!
  - Each face edge is represented twice
- Finding neighbors is expensive (search)

Vertex normals



Vertex normals



- Compute/store a normal at each vertex
- Improves shading
- Computed by averaging

Exercises

- Make some objects using index face set structure
- Verify that $V - E + F = 2$ for some polyhedra
- Think about testing for intersection between a ray and a polygon (or triangle)

Vertex normals

```
for all vertices i
 normals[i] = 0;

for all faces
 for all vertices in face[i]
  normals[faces[i][j]] += faces[i].normal;

for all vertices
 normals[i] = normalize(normals[i]);
```

Recap

- We have seen definition of planes and polygons and their use in approximating general shapes
- We have looked at data structures for shapes
  - Indexed face sets
- The former is easy to implement and fast for rendering
- It is possible, though we haven't shown how, to convert between the two