Computer Graphics (COMP0027) 2022/23

Planes and Polygons

Tobias Ritschel

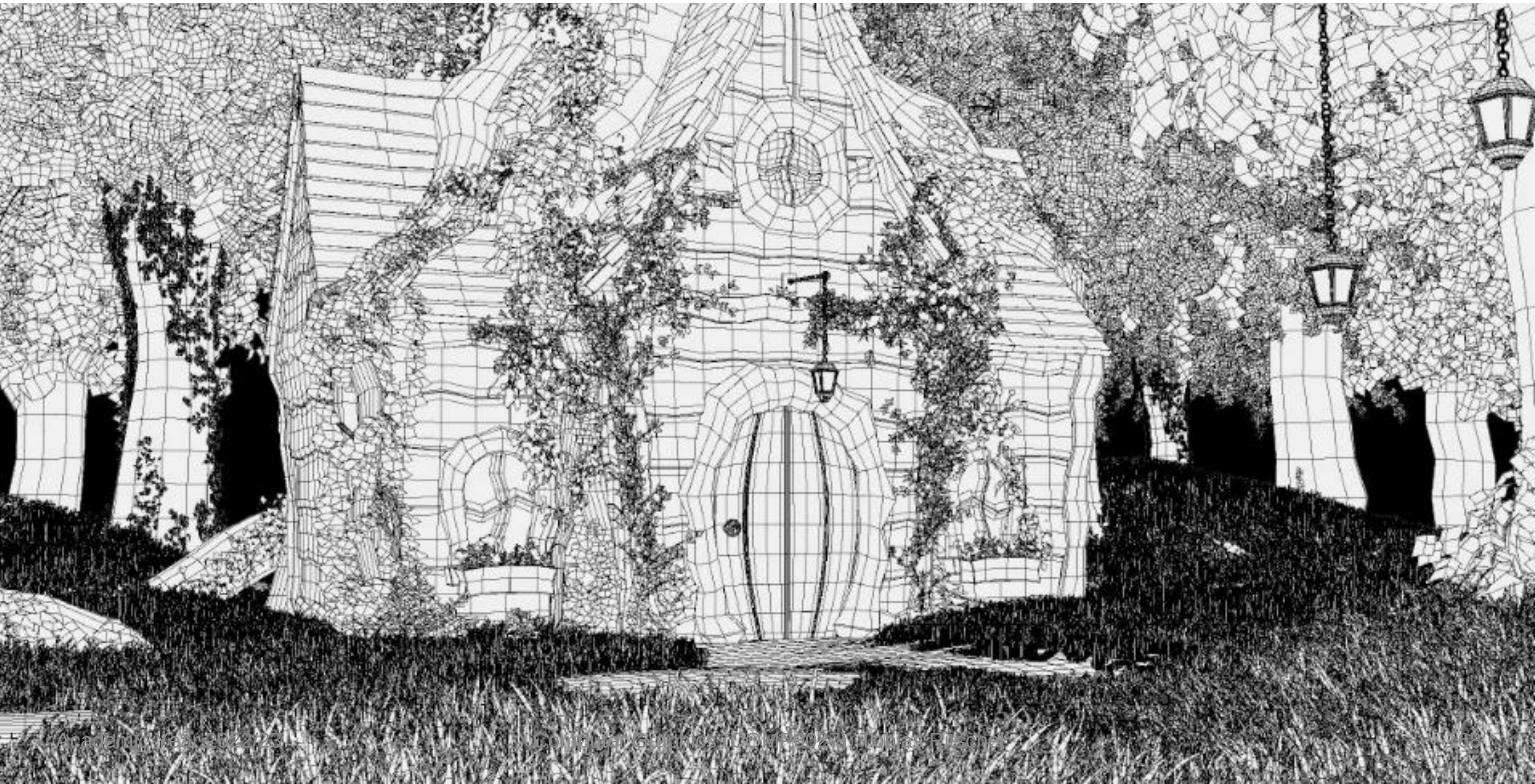
Overview

- 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

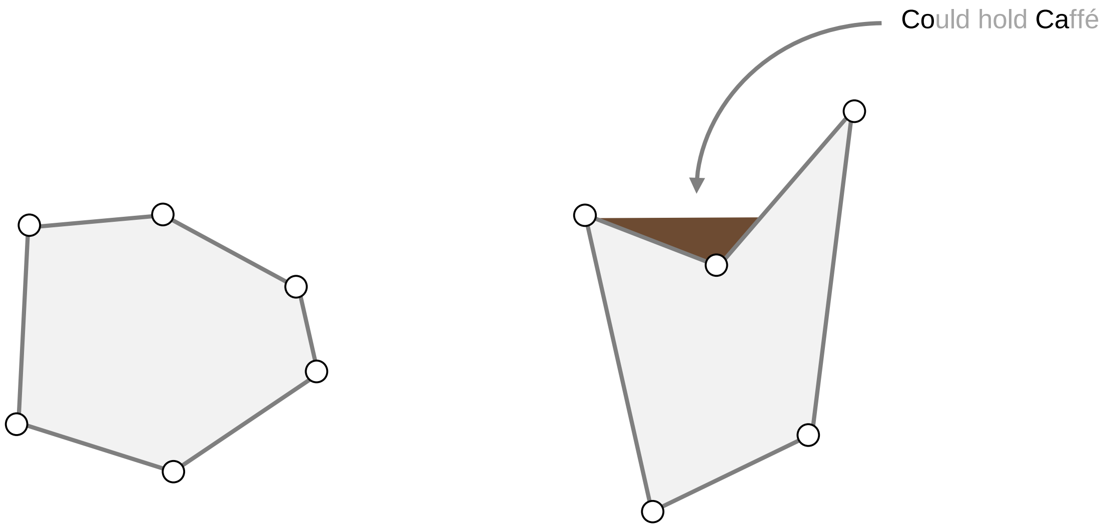
- A polygon (face) P is defined by a series of points

$$P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_{n-1}, \mathbf{p}_n\}$$

$$\mathbf{p}_i = (x_i, y_i, z_i)$$

- We ask the points to be **co-planar**
  - 3 points always a plane
  - Further point need not lie on that plane

Convex vs. Concave

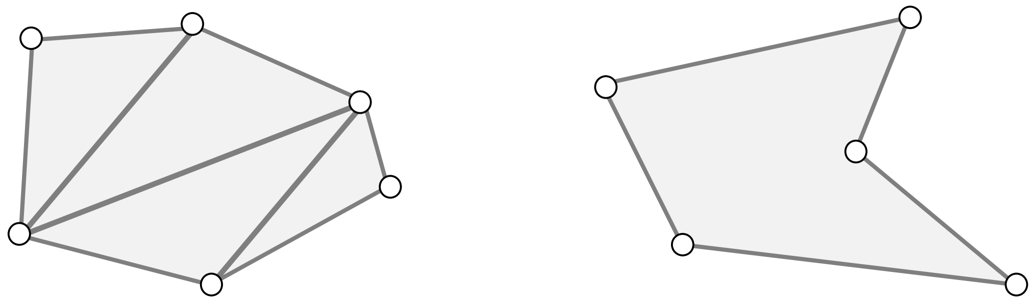


Convex

Concave

Convex, Concave

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



Recap: Equation of a sphere

$$\sqrt{x^2 + y^2 + z^2} = r$$

- All points x, y, z lie on a sphere of radius r
- r is radius
- Remember: sphere at the origin

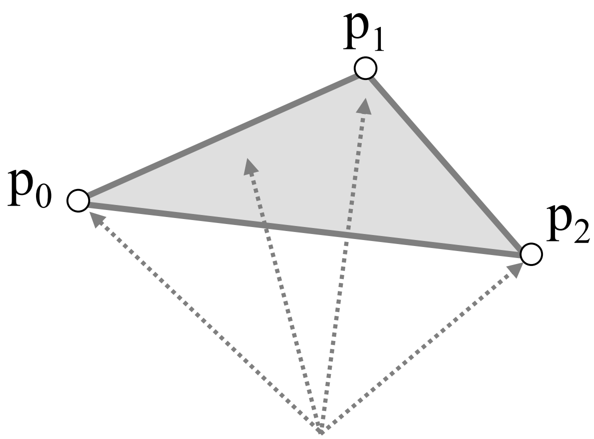
Equation of a plane

$$ax + bx + cz = d$$

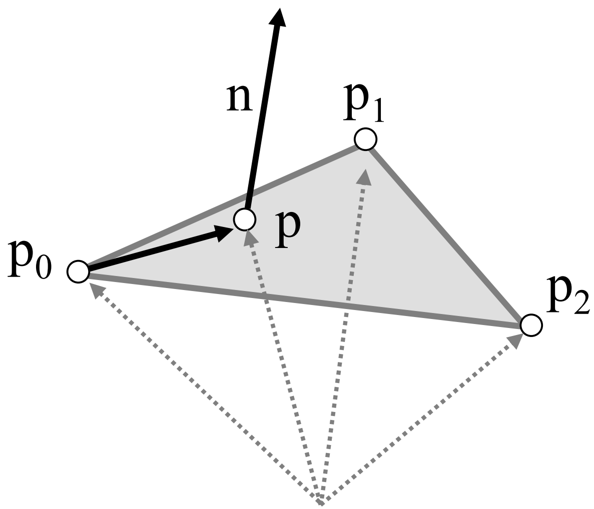
- All points x, y, z lie on a plane with minimal signed distance d
- Plane, other than sphere, does not have "position"
- We will derive a, b, c now

Deriving *a*, *b*, *c*, *d* (1)

- Given are three 3D points

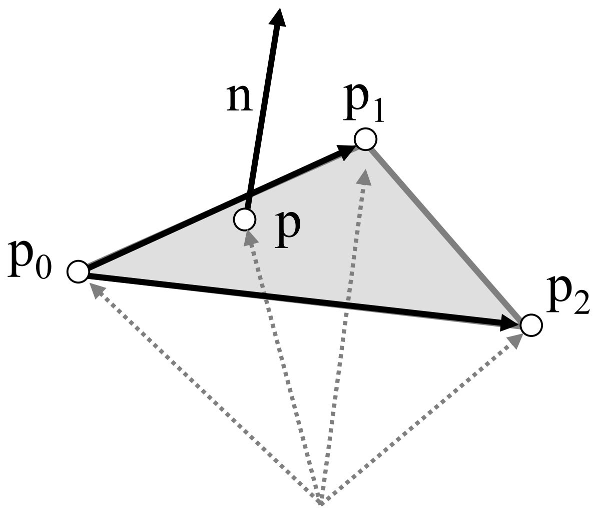


Deriving a, b, c, d (2)



- Vectors in the plane are **all** orthogonal to the plane normal vector

Deriving *a*, *b*, *c*, *d* (3)

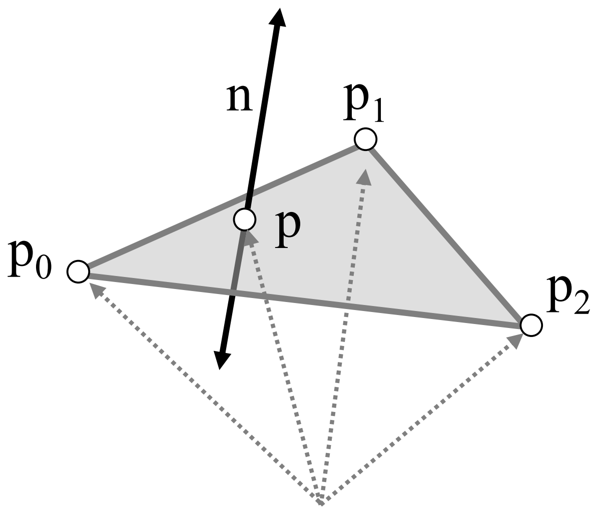


- The cross product

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

defines a **normal** to the plane

Deriving *a*, *b*, *c*, *d* (4)



- There are two normals (they are opposite)
- Depends on choice of cross product / left-hand vs right-hand

Deriving *a*, *b*, *c*, *d* (5)

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

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

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

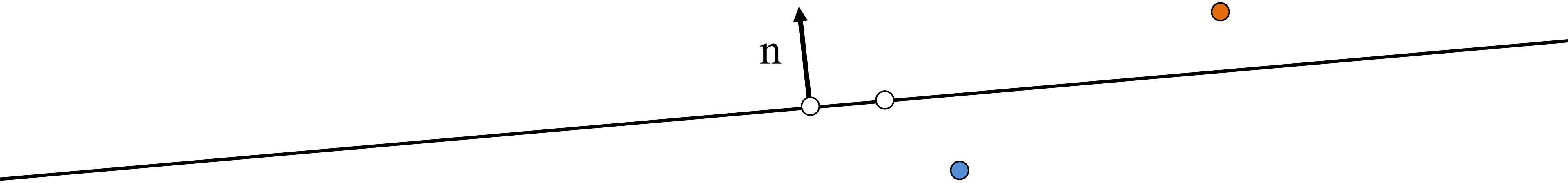
$$ax + bx + cz = d$$

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



Ray-plane intersection

- Coursework!

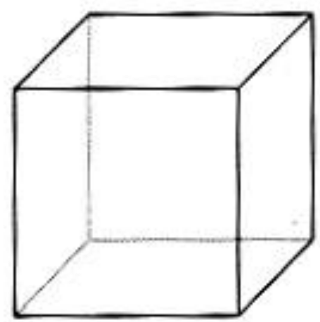
Polyhedra

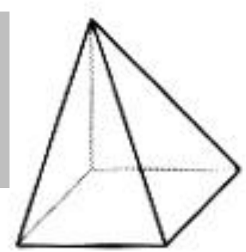
Polyhedra

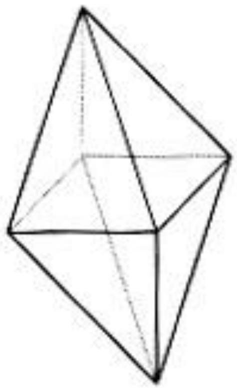
- Polygons are often grouped to form **polyhedra**
  - Each **edge** connects 2 vertices
  - Each **vertex** joins 3 (or more) edges
  - No faces intersect

Polyhedra

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

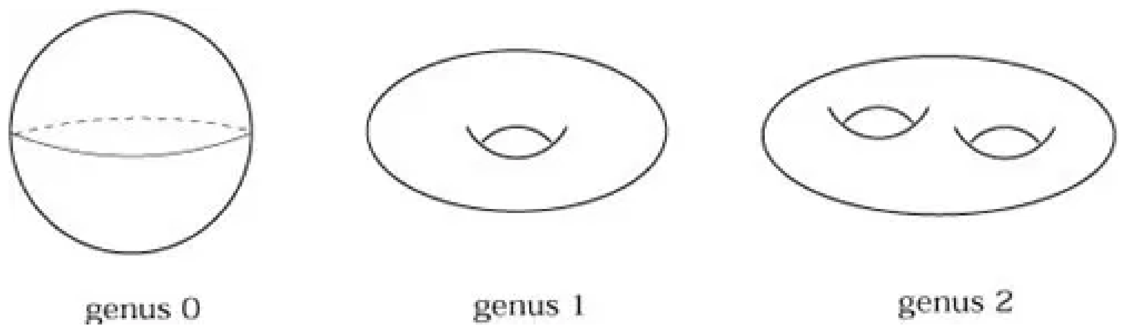






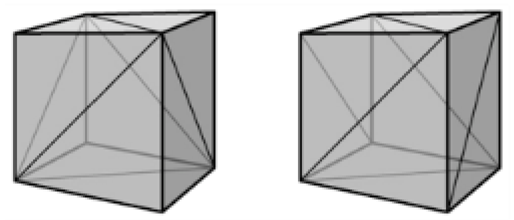
Genus g

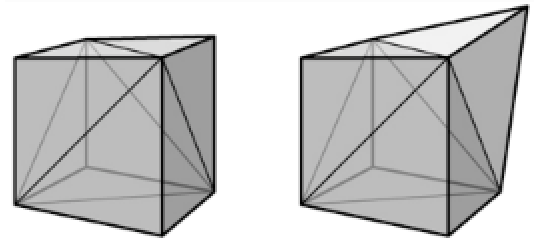
- "Number g of holes"



Topology / Geometry

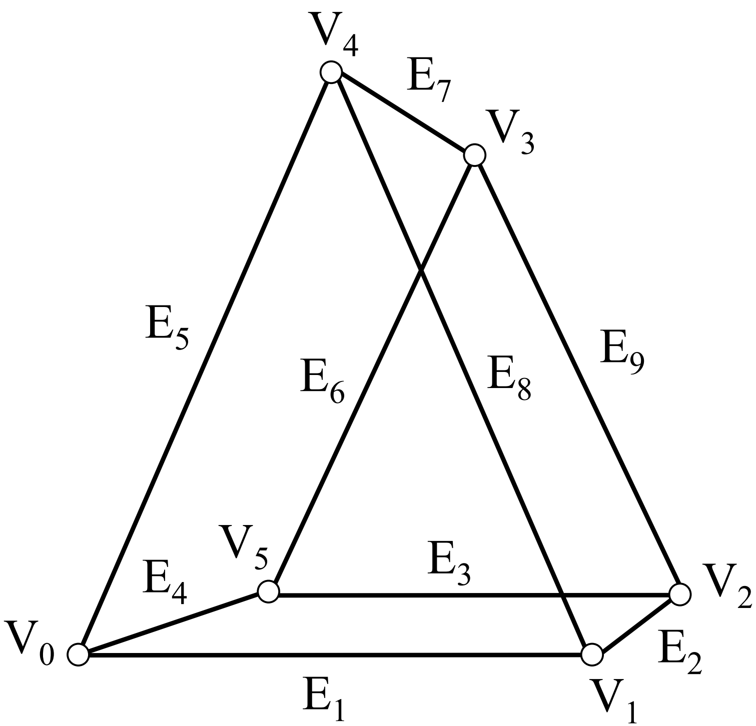
Same geometry, different mesh topology





Same topology, different geometry

Example polyhedron



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

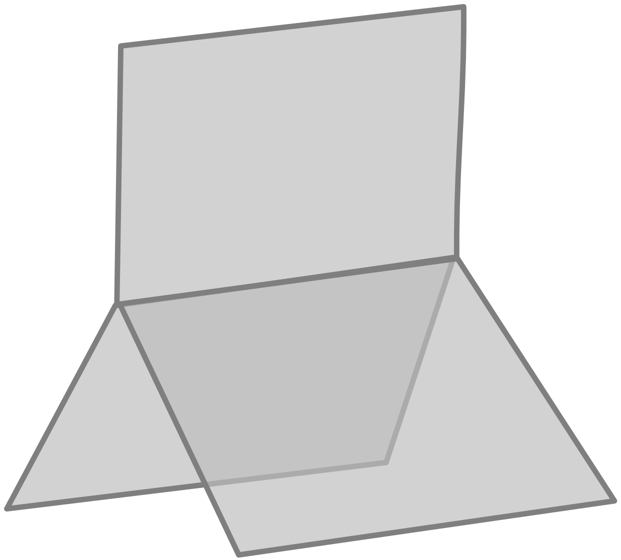
$|V|=6$, $|\mathrm{F}|=5$, $|\mathrm{E}|=9$

$|\mathrm{V}| - |\mathrm{E}| + |\mathrm{F}| = 2$

Manifold

- Ideally: should be **manifold**
  - One vertex has one loop of polygons/edges
  - Each edge has one or two polygons
- Quiz: Counter-examples?

Non-manifold of sadness



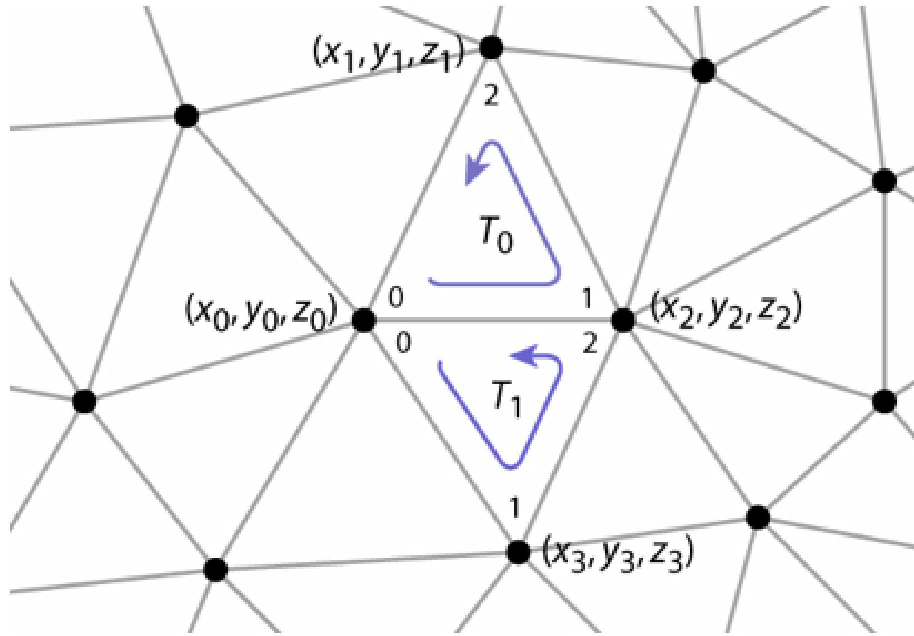
Representing polyhedra

Multiple options:

1. Separate polygons
   – Replicate all coordinates
2. Index face set
   – Share vertices
3. Winged-edge data structure
   – General and space-efficient

Separate polygons

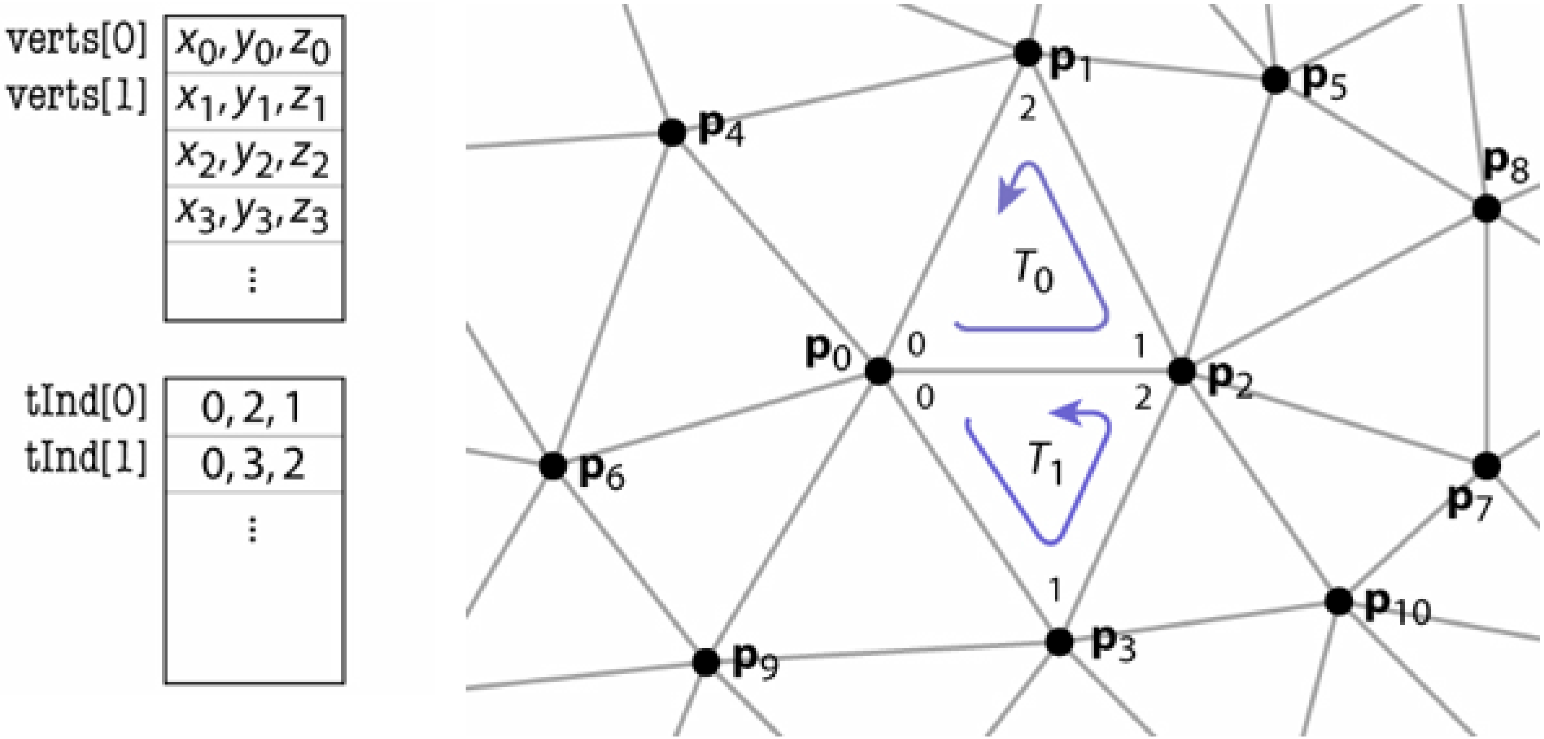
	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
	⋮	⋮	⋮



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 neighbouring polygons

Indexed face set



Indexed face set

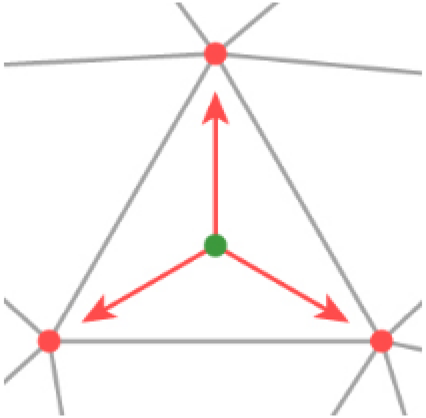
- Store each vertex once
- Each polygon points to its vertices

  - Vertex array

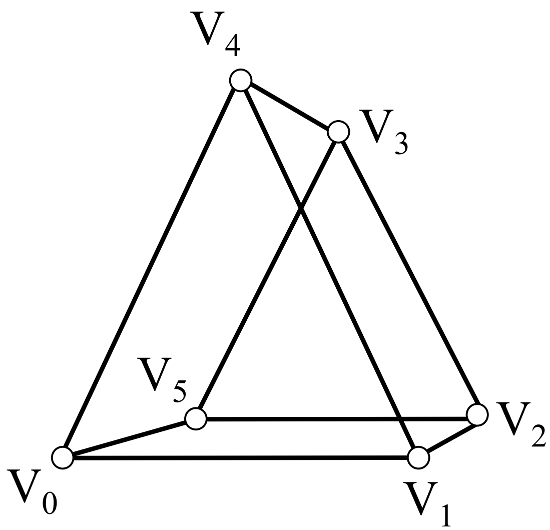
```
vertices[0] = (x0, y0, z0);
vertices[1] = (x1, y1, z1);
…
```

  - 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**

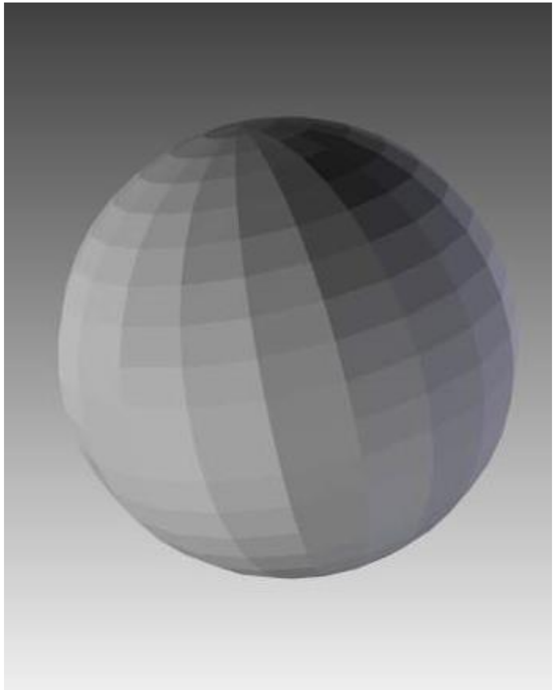
Indexed face set issues

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

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

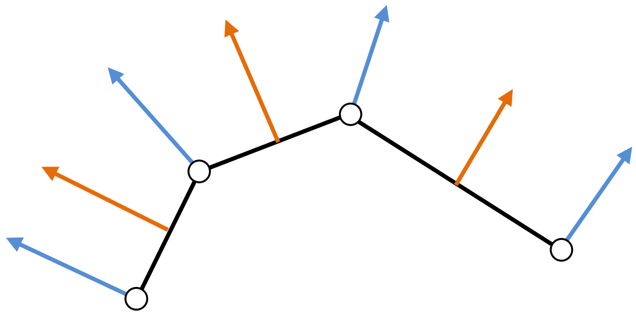


Face normals

Vertex normals

Vertex normals

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



Vertex normals (bad)

```
for all vertices i
 for all faces f
  if any(faces[f].index[] = i)
   normals[i] += faces[f].normal;

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

Vertex normals (good)

```
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]);
```

Complexity

- Bad complexity

$$O(\text{vertexCount} \times \text{faceCount})$$

- Good complexity

$$O(\text{vertexCount} + \text{faceCount})$$

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