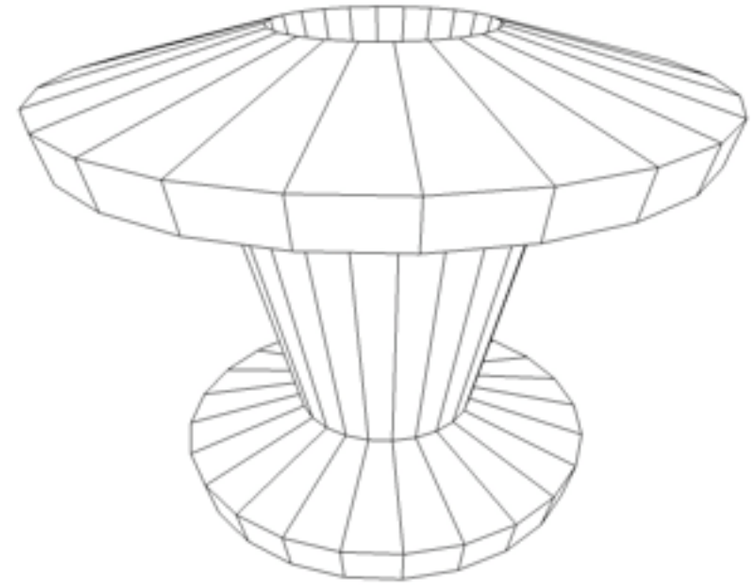


Monash University • Clayton's School of Software Engineering

CSE3313 Computer Graphics

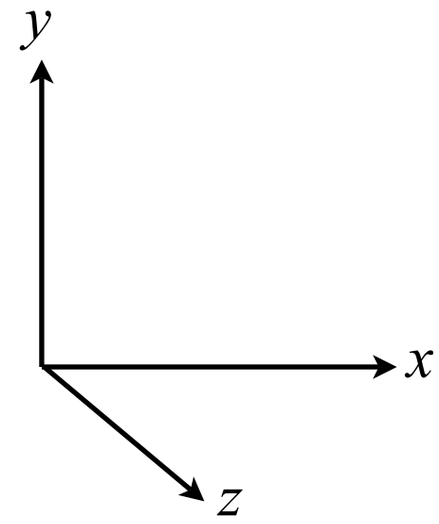
Lecture 14: Introduction to 3D graphics

Three-Dimensional Graphics

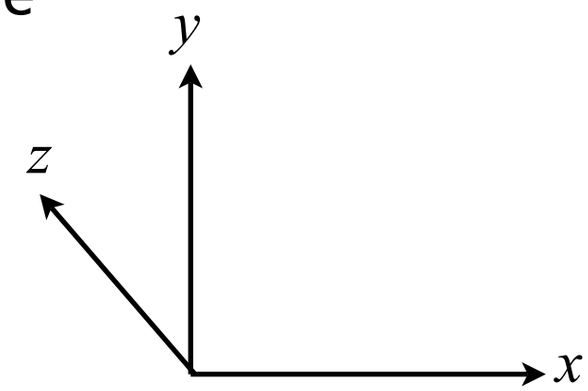
- In many ways three dimensional graphics is an extension of two dimensional graphics.
- Main differences:
  - Objects and the viewer are in a 3D world but the image produced on the display device is 2D.
  - In producing the final image an additional step called **projection** is necessary.
  - The extra dimension in the data means an increase in the amount of work — efficiency becomes more important.
- OpenGL works natively in three-dimensions — 2D graphics are just a restricted case of 3D.

Extension of 2D primitives to 3D

- We need to add an extra axis which is at right angles to the normal x and y axes.



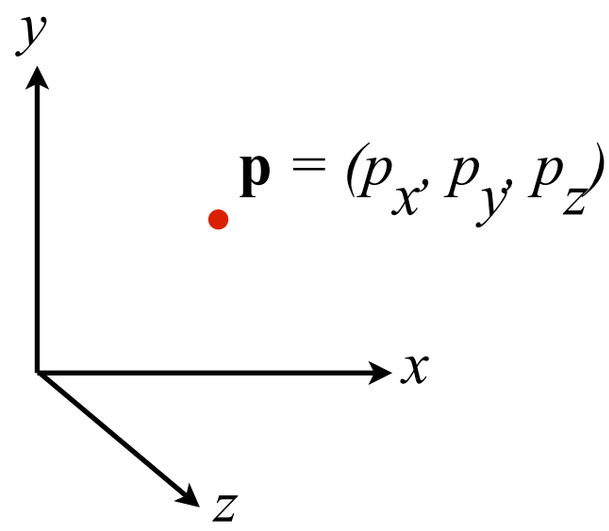
- Two options:
  - Left-handed coordinate system;
  - Right-handed coordinate system.
- By convention, we will use a right-handed coordinate system. Some packages may use a left-handed system, or a combination of both.
- A further convention will be necessary to determine the directions of positive rotation around the coordinate axes.
- A positive rotation will be counter-clockwise when looking at the origin from the positive side of the axis.



Points and Lines in 3D

Points in 3D space can be represented:

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

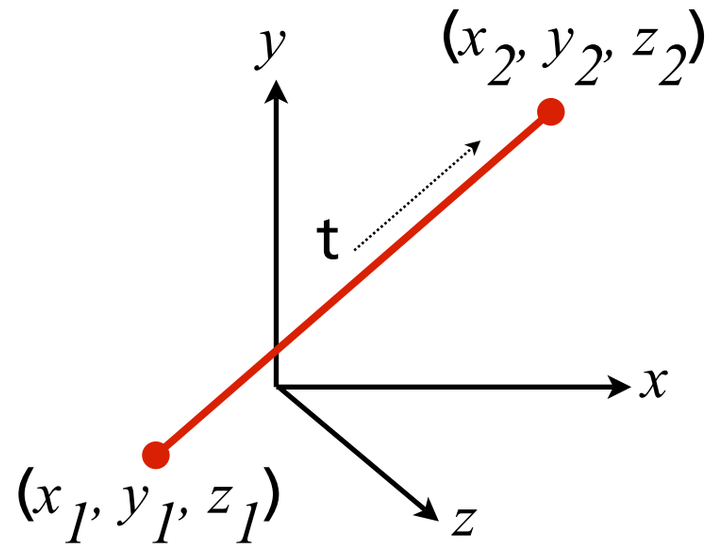


A line connecting (x_1, y_1, z_1) and (x_2, y_2, z_2) can be represented parametrically by:

$$x(t) = (1-t)x_1 + tx_2$$
$$y(t) = (1-t)y_1 + ty_2$$
$$z(t) = (1-t)z_1 + tz_2$$

or :

$$l(t) = (1-t)p_1 + tp_2$$



Lines and Planes

- A line in 2D space can be represented by the general equation:

$$ax + by + D = 0$$

- In 3D space this can be generalized to give

$$ax + by + cz + D = 0 \qquad (1)$$

- As long as $D \neq 0$ we could scale D so that (1) only has three distinct parameters.

- Any 3 non-colinear points define a plane.

- We can normalize (1) by choosing

$$a^2 + b^2 + c^2 = 1$$

- In which case D gives the distance from the origin to the plane.

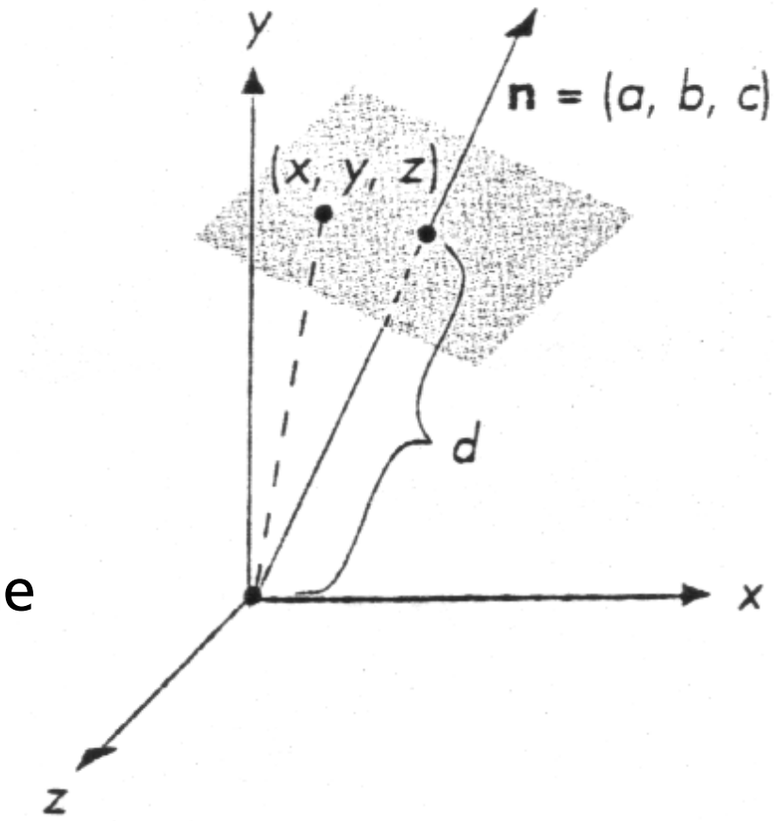
- Planes with the same a, b and c are parallel to each other.

- If $D = 0$ then the origin is in the plane.

Point-Normal form of a plane

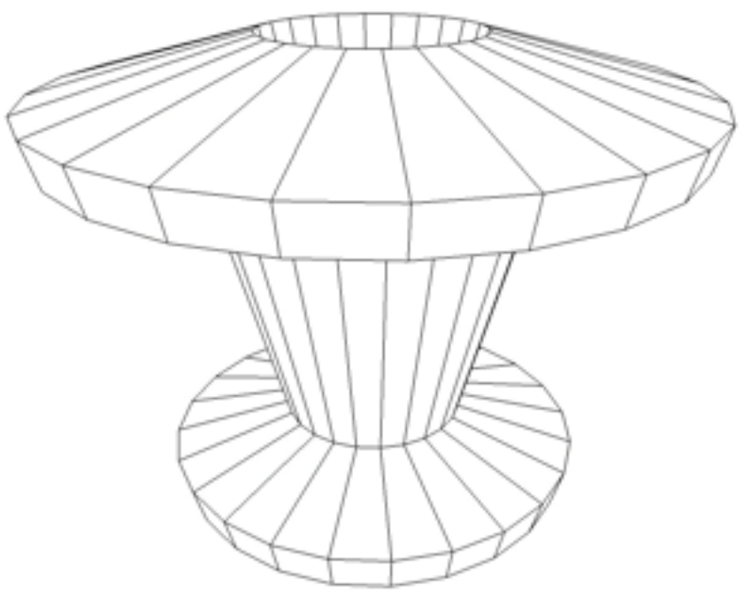
Let $p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ and $n = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ $p_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$

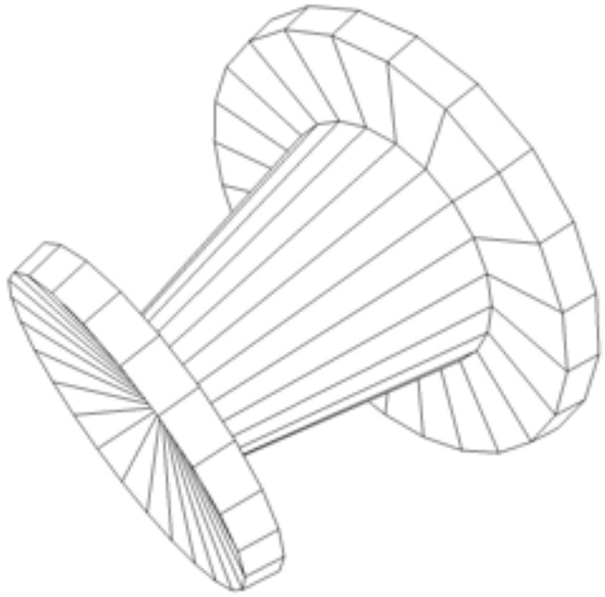
- Then from (1) we have $(\mathbf{p} - \mathbf{p_0}).\mathbf{n} = 0$, $((\mathbf{p}-\mathbf{p_0})^T\mathbf{n} = 0)$.
- $\mathbf{n}$ is *orthogonal* (perpendicular, at right angles) to all vectors in the plane.
- $\mathbf{n}$ is the **unit normal** to the plane and is useful in describing the orientation of the plane.
- $\mathbf{n}$ can also be found by taking the cross product of the two vectors formed by the three non-colinear points that define the plane.

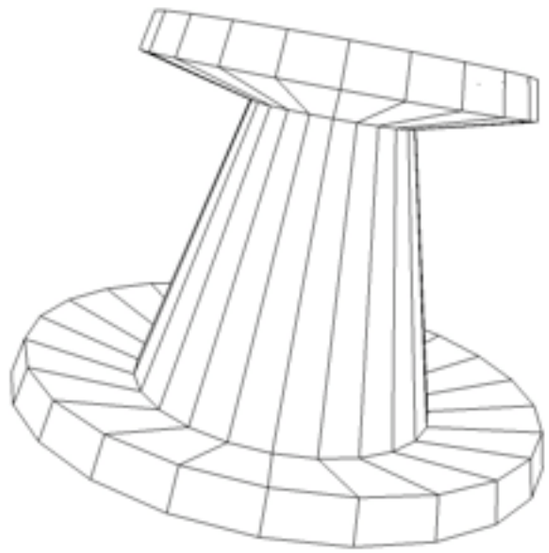


Three-Dimensional Primitives

- OpenGL is inherently 3D API.
- All vertices are stored as 3D vectors in homogeneous form.
- For 2D graphics, 2D vectors were used by setting the *z* coordinate to 0. In 3D the *z* coordinate is no longer necessarily 0.







Inward and Outward Pointing Faces

- We can describe a polygon in 3D by specifying a sequence of vertices. If the vertices do not all lie in the same plane we may get unpredictable results on different implementations of the API.
- It is best to ensure that the vertices of an individual polygon all lie in the same plane.
- In 3D a polygon has 2 faces. We can display either or both of them.
- A face is **outward-facing** if the vertices are traversed in a counter clockwise order when the face is viewed from the outside.
- If you orient your hand in the direction the vertices are traversed, the thumb points outward (**right-hand rule**).

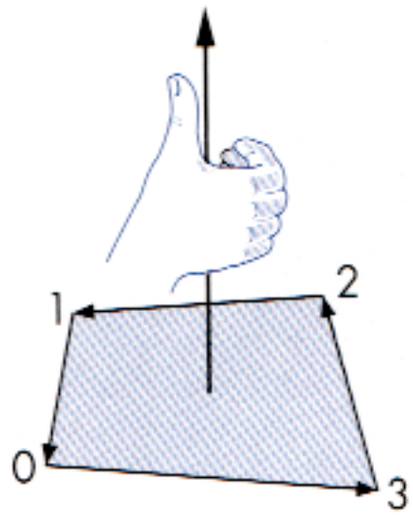


Figure 4.29 Traversal of the edges of a polygon.

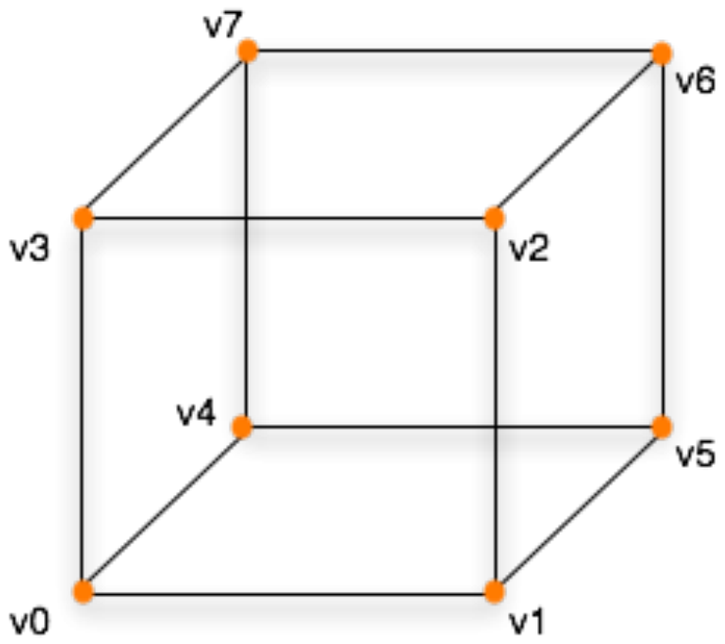
Polyhedra

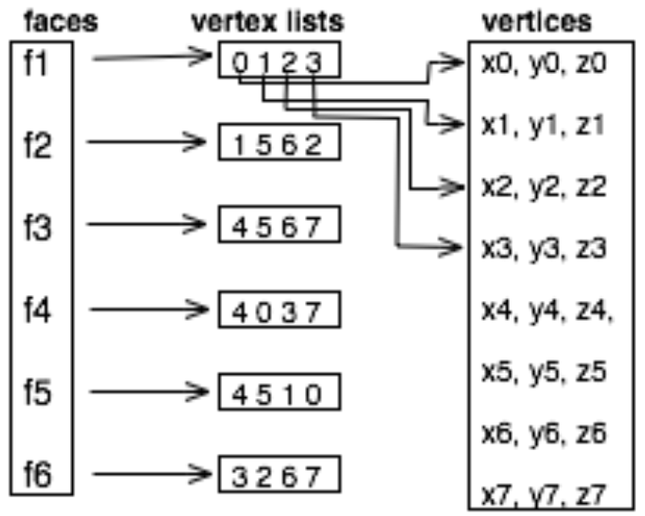
- Complex objects are usually represented by large collections of polygons.
- Each polygon belongs to a larger structure, a *polyhedron*, where polygons are connected edge-to-edge A simple polyhedron has the following properties:
  - Each edge connects exactly two vertices, and is the boundary between exactly two faces;
  - Each vertex is a meeting point for at least three edges;
  - No two faces intersect, except along their common edge.
- Provided there are no holes the polyhedron satisfies *Euler's Rule:*

  $V - E + F = 2$

Data Structures for 3D representation

- Representations vary depending on the use to which we wish to put them.
- For graphics, it is often necessary to capture a cubes **topology** as opposed to its **geometry.**
- Many 3D polygonal objects share vertices — hence the **vertex list** data structure may be useful.





Winged Edge Data Structure

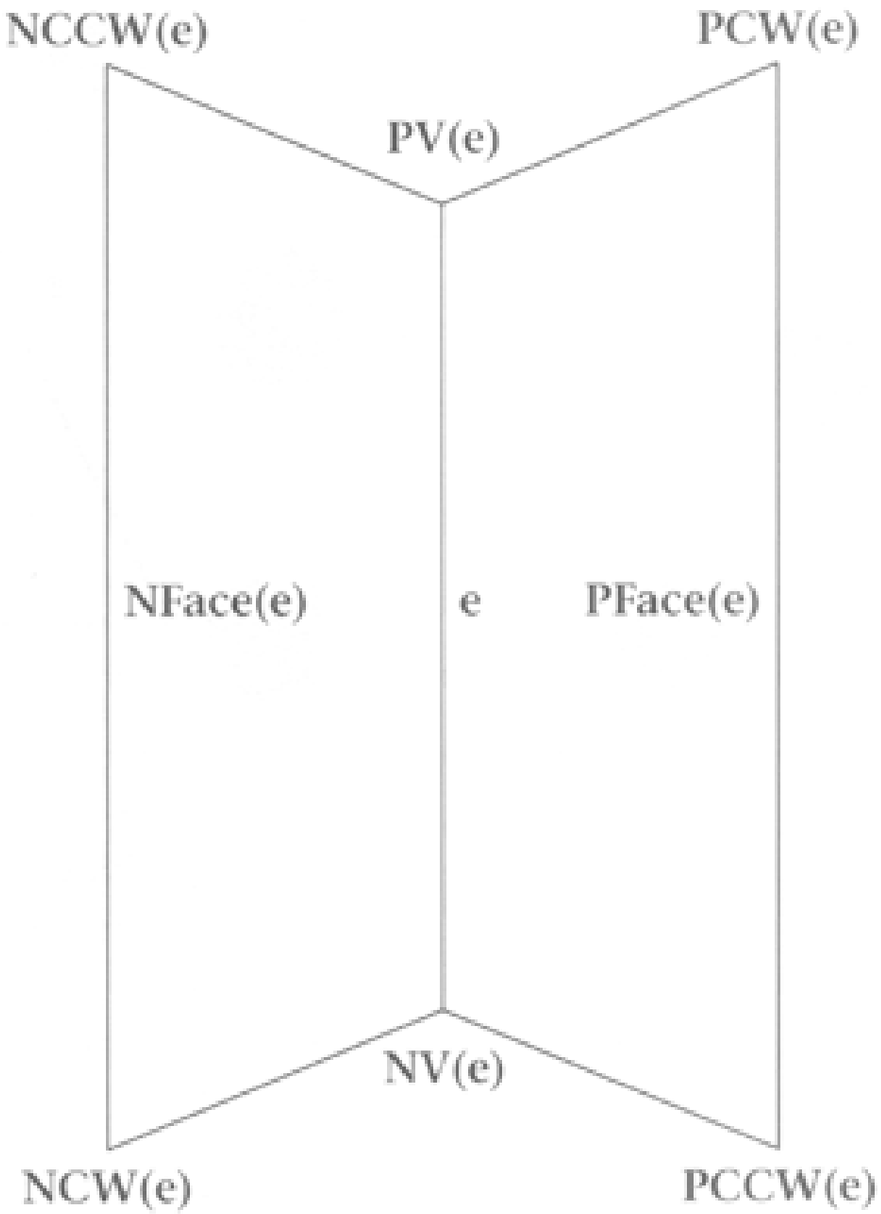
Ref: Slater et. al. pp 170–172

- Vertex-Face data structures can be limiting when more complex queries or modifications to the database are required. The *Winged-Edge* data structure (Baumgart 1975) allows a rich set of queries:
  - for any face, find all of the edges in a CW/CCW order;
  - for any face, traverse all of the vertices;
  - for any vertex, find all the faces that meet at that vertex;
  - for any vertex, find all the edges that meet at that vertex;
  - for any edge, find its two vertices;
  - for any edge, find its two faces;
  - for any edge, find the next edge on a face in CW/CCW order

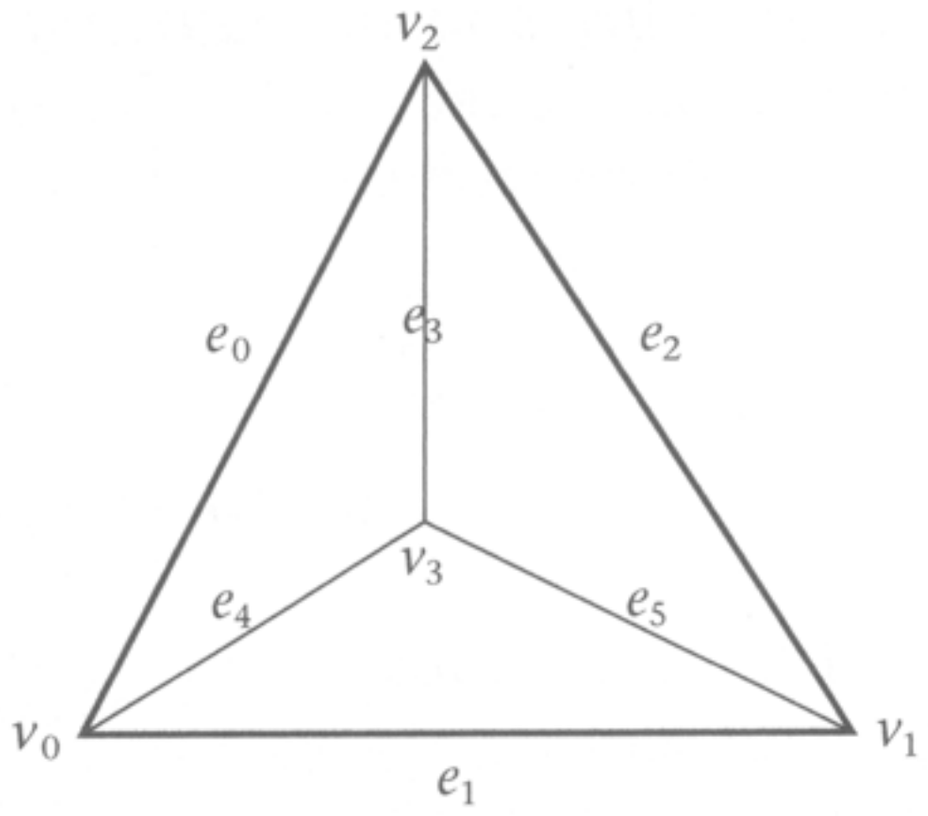
Winged Edge Data Structure

Ref: Slater et. al. pp 170–172

- Polyhedron represented as a *Body*
- *Body* contains rings (doubly linked lists) called *Vertex, Edge,* and *Face* (plus information linking to other Bodies).
- *Vertex Ring:* vertex data (x,y,z), previous and next vertices, associated edge ring.
- *Edge Ring:* previous and next edges, “wings”: neighbouring vertices and faces
- Vertex ring: geometry <–> Edge ring: topology
- *Face Ring:* links to next and previous faces, edge ring



N = Next
P = Previous
CW = Clockwise
CCW = Counterclockwise
V = Vertex
e is a particular edge



Vertices	Faces
v_0	F0 = 0, 3, 1
v_1	F1 = 0, 2, 3
v_2	F2 = 3, 2, 1
v_3	F3 = 0, 1, 2