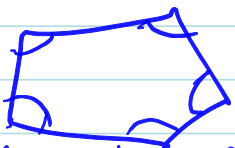
3D lecture: you should have actually been there!!!

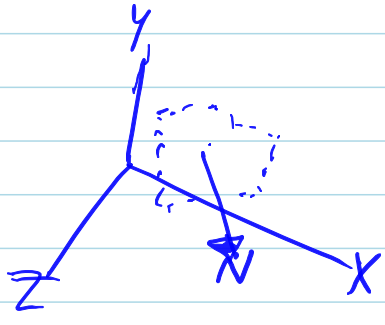
- 3D: image rendered in LEFT and RIGHT eyes
- rotation about WHICHEVER axis is up (not defo X, Y, Z)
- All models on screen to same size ☺
  - Camera placing algorithm (correct distance)
  - Object CENTRE
    - ! PREPROCESS all vertices
    - ! Max and min, figure out CENTRE using averages

POLYGONS & PIXELS

- Δ: most common polygon (flat 👍)
- ordered set of vertices, with connected edges
- ! look out for HOLES and CROSSOVER POINTS
- ! openGL only needs convex polygons:

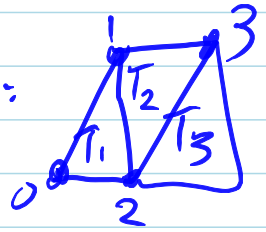


  - ALL L's $< 180°$
- can always TESSELLATE polygons into smaller triangles (esp. concave → convex 👍)
- surface normal: perpendicular vector to plane of polygon
  - orientation, FRONT & BACK
  - $\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} y_1 z_2 - z_1 y_2 \\ x_1 z_2 - z_1 x_2 \\ x_1 y_2 - x_2 y_1 \\ 1 \end{bmatrix}$



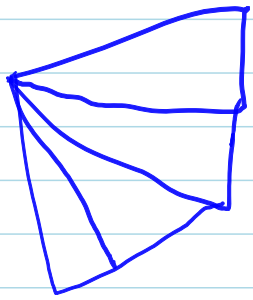
- $N = E_2 \times -E_1$ (any two edges E_1, E_2)
- Why is POLYGON SOUP bad when representing scenes:
  - WASTE of space

(1) Solution, use meshes:



- +1 vertex → ++Δ!

☹ N linked Δ's defined NOT with 3N, but only N+2 vertices!

-  : triangle fan

- quad strips, TESSELLATED into Δ strips during rendering

☹ in a MESH, N·M quads defined with (N+1)·(M+1) vertices rather than 4·N·M

- How to CONVERT lines?
  - sample true geometry of line, approximate using nearest pixels available
  - Bresenham Algo: $y = mx + c$ (horizontal pixel value; vertical one)

- BA not good: you don't have transparent shading of pixel (blocky colours)

- Converting Polygons?
  - viewing pipeline: world → pixel coordinates
  - Scan convert a Δ: naïve + expensive approach = convert each EDGE Δ
    - solution: sweep line algo

1) step down pair of edges
2) go down scanline by scanline
   - find start & end of each scanline
3) Fill pixels inside Δ for THAT scanline

(✓) ONLY calculate gradient once ☺ efficient

- Hidden surface removal
  - How we convert 3D to 2D representations...?
    - working in 3D space was too hard ☹ 2D easier
  - z buffer (depth buffer)
    - stores z-depth of a pixel p
    - if (p.z value closer than old one)
      UPDATE it
      else if (further)
      REJECT it
  - Before doing above, MUST:
    1) initialise pixels = background colour
    2) initialise all depth values = MAX_depth

- rotating a 3D shape: polygon offset to the z axis
- HIERARCHICAL structures: object
  - surfaces
    - polygons
      - edges
        - vertices
- Mesh = facelist
  - edges & vertices
- In general: manual reconstruction omits LOTS of details