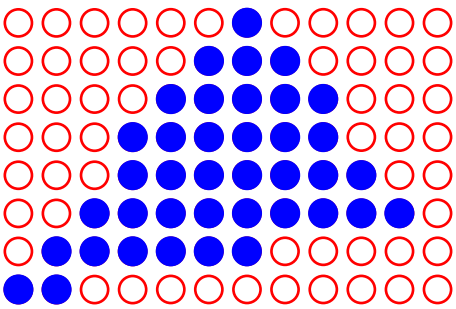
MANCHESTER 1824

Please take a pair of 3D glasses

MANCHESTER 1824

COMP27112

Computer Graphics and Image Processing



4: Polygons and pixels

Toby.Howard@manchester.ac.uk

MANCHESTER 1824

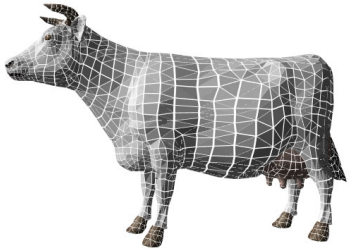
Introduction

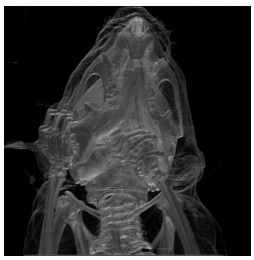
- We'll look at
  1. Polygons and triangles
  2. Scan conversion of polygons
  3. Hidden surface removal with the Z-buffer
  4. Data structures for polygonal models
  5. Where do we get polygons from?

MANCHESTER 1824

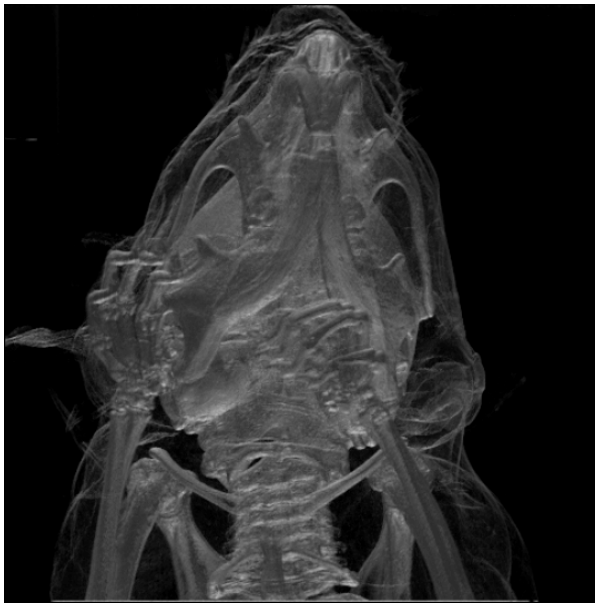
Polygons as building blocks

- The most common building block for 3D graphics is the polygon
- We usually use triangles
- Many different types of 3D objects can be modelled well with meshes of polygons…
- …but there are other techniques too which don't involve polygons (e.g., volume rendering, particle systems, covered in COMP37111)





MANCHESTER 1824

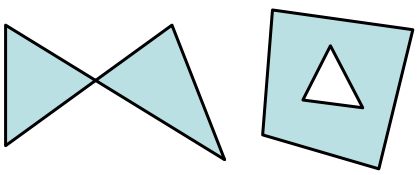


5

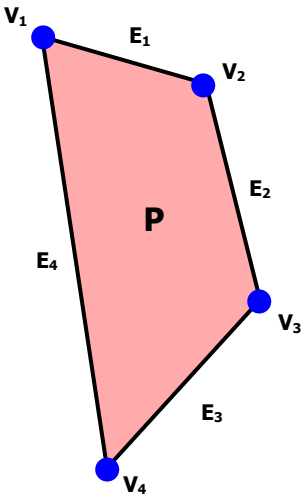
MANCHESTER 1824

What is a polygon?

- An ordered set of vertices ($V_1 \ldots V_N$)
- A set of edges between each pair of vertices ($E_1 \ldots E_N$)
- The polygon (P) is then the space bounded by the vertices
- Some polygons are not so simple…



- So we avoid polygons like that...
- ...but happens if we can't?

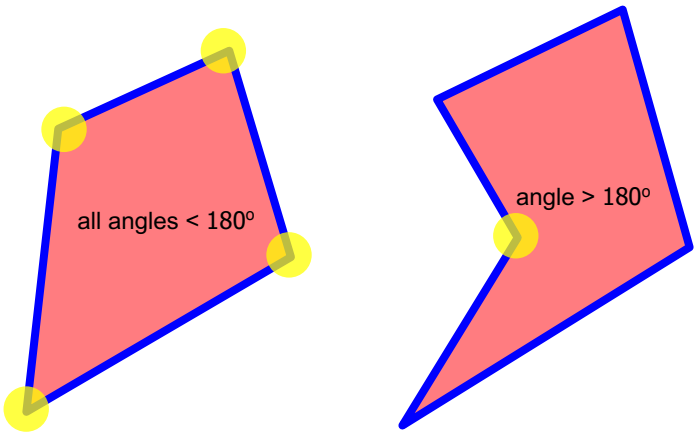


6

MANCHESTER 1824

OpenGL needs convex polygons

- A convex polygon has all interior angles < 180°



convex polygon

non-convex polygon
(aka concave polygon)

7

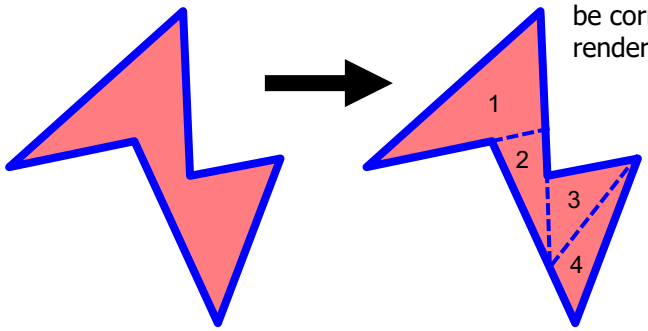
MANCHESTER 1824

Tessellation

- GLU provides functions to tessellate polygons

BEFORE: This **concave** polygon won't be rendered correctly

AFTER: A set of **convex** polygons – each of which can be correctly rendered

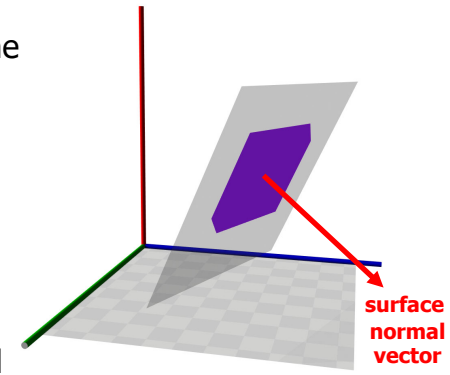


8

MANCHESTER 1824

Polygon attributes: surface normal

- The **surface normal** is a vector perpendicular to the plane of the polygon
- We can use this to give a polygon a distinguishable 'front' and 'back' …
- …and also to describe its **orientation** in 3D space
- Orientation is an essential property used extensively in lighting calculations, collisions, culling…



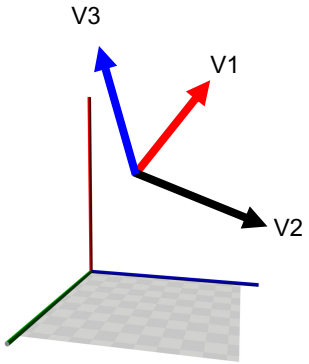
MANCHESTER 1824

The Cross Product

- is a **vector**, defined as follows:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} y_1 \times z_2 - z_1 \times y_2 \\ x_1 \times z_2 - z_1 \times x_2 \\ x_1 \times y_2 - y_1 \times x_2 \\ 1 \end{bmatrix}$$

- For two vectors, their cross product is a third vector **perpendicular** to them both (forming a right handed system)



V3 = V1 x V2

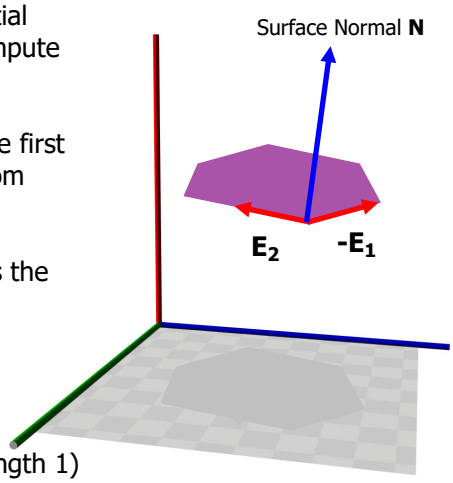
MANCHESTER 1824

Finding the surface normal

1. Choose a pair of sequential edges $\mathbf{E_1}$ and $\mathbf{E_2}$ and compute their vectors
2. Invert the direction of the first so they now emanate from their shared vertex
1. Their cross product gives the surface normal N

   $N = E_2 \times -E_1$

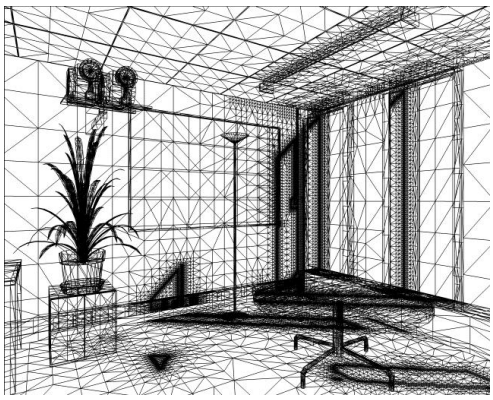
2. We then almost always normalize N (make its length 1)



MANCHESTER 1824

Representing scenes with polygons

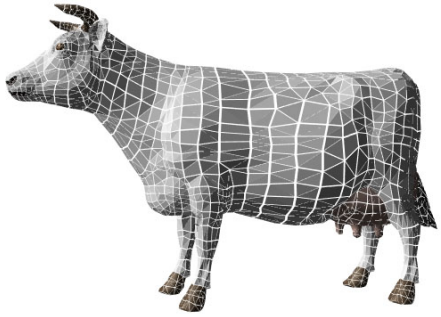
- How can we represent a scene using polygons?
- One option is to have a huge list of individual polygons, to colour them individually and to draw them all in order
- This is called **polygon soup** and has a number of undesirable properties…



MANCHESTER 1824

Problems with polygon soup

- Huge waste of storage space… most models contain surfaces not individual polygons, so we could share vertices rather than replicate them per polygon
- Complete loss of semantics… does a polygon belong to a cow… or a table…?
- Makes interaction with the model difficult



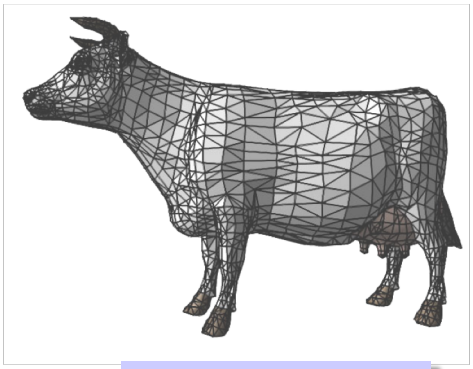
18,000 separate vertices, many repeated

13

MANCHESTER 1824

Polygon meshes

- Use linked groups of polygons, or **meshes**, to represent surfaces
- Retains semantics of surfaces…
- …but reduces storage by sharing vertices and edges
- Helps with structuring the model so we can manipulate it more easily

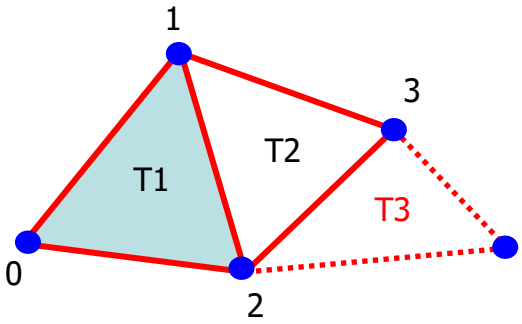


6,000 separate vertices, mostly unique/shared

14

MANCHESTER 1824

Meshes: triangle strips

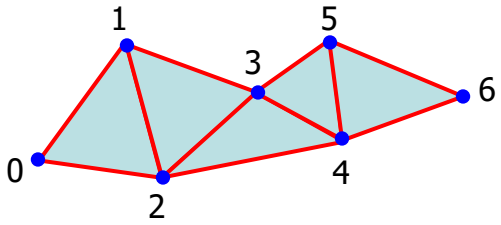


- Add one new **vertex**…
- … get one new **triangle**

MANCHESTER 1824

Meshes: triangle strips

- Collection of linked triangles
- Very widely used – efficient

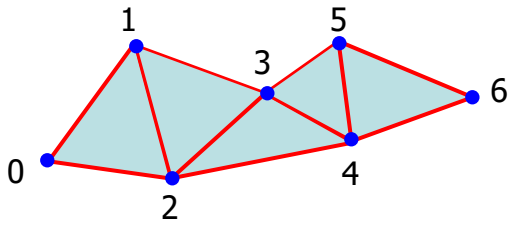


- **N** linked triangles can be defined using **N+2** vertices – compared with **3N** vertices if each triangle were defined separately

MANCHESTER 1824

Triangle strip in OpenGL

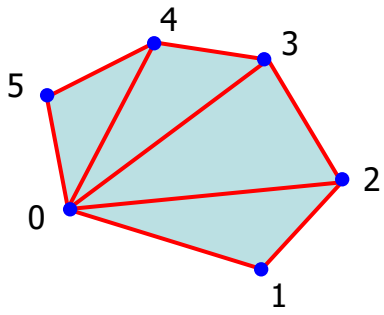
```
glBegin(GL_TRIANGLE_STRIP);
  glVertex3f(x0, y0, z0);
  glVertex3f(x1, y1, z1);
  glVertex3f(x2, y2, z2);
  glVertex3f(x3, y3, z3);
  /* and similar for more vertices */
glEnd();
```



MANCHESTER 1824

Meshes: triangle fan

- Collection of linked triangles

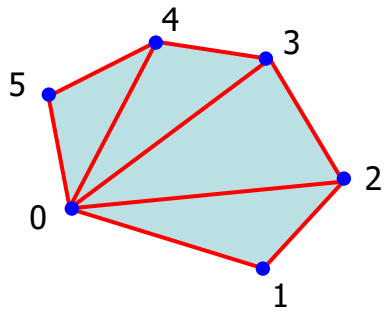


- **N** linked triangles can be defined using **N+2** vertices – compared with **3N** vertices if each triangle were defined separately

MANCHESTER 1824

Triangle fan in OpenGL

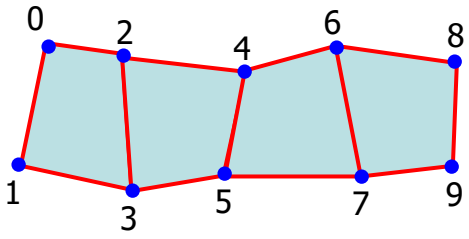
```
glBegin(GL_TRIANGLE_FAN);
  glVertex3f(x0, y0, z0);
  glVertex3f(x1, y1, z1);
  glVertex3f(x2, y2, z2);
  /* etc */
glEnd();
```



MANCHESTER 1824

Meshes: quadrilateral strips

- Collection of linked quadrilaterals (a.k.a. quads)

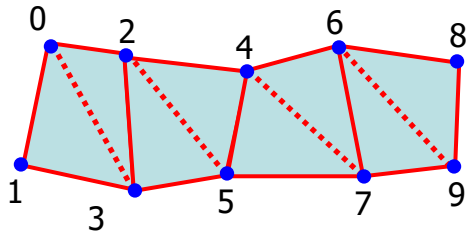


- **N** quads can be defined using **2N+2** vertices, compared with **4N** separate vertices

MANCHESTER 1824

Quad strip in OpenGL

```
glBegin(GL_QUAD_STRIP);
  glVertex3f(x0, y0, z0);
  glVertex3f(x1, y1, z1);
  glVertex3f(x2, y2, z2);
  glVertex3f(x3, y3, z3);
  /* etc */
glEnd();
```

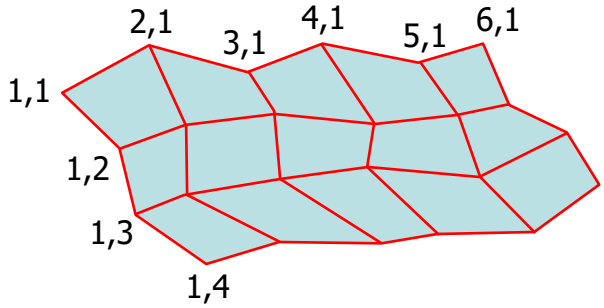


Tessellated into triangles during rendering

MANCHESTER 1824

Meshes: quadrilateral meshes

- Collection of linked quadrilaterals
  - Used in terrain modelling, and for approximating curved surfaces

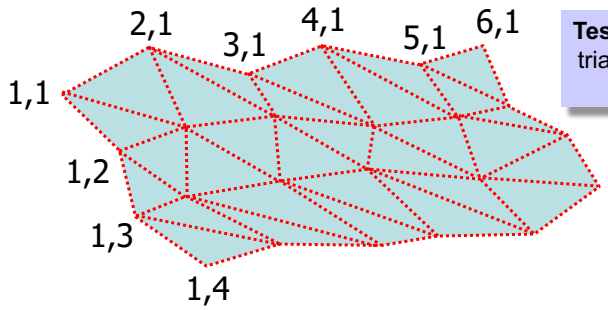


- **N** x **M** quads can be defined using **(N+1) * (M+1)** vertices, compared with **4*M*N** separate vertices

MANCHESTER 1824

Meshes: quadrilateral meshes

- Collection of linked quadrilaterals
  - Used in terrain modelling, and for approximating curved surfaces



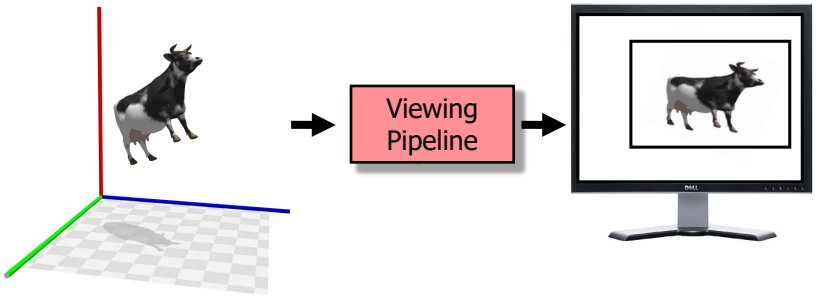
Tessellated into triangles during rendering

- **N** x **M** quads can be defined using **(N+1) * (M+1)** vertices, compared with **4*M*N** separate vertices

MANCHESTER 1824

From the model to the display

- We model in a 3D space, then take a view using a "camera" to create a 2D screen image

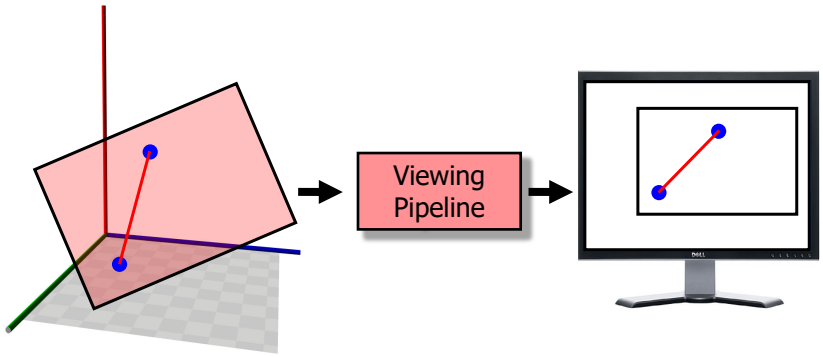


Vertices in World Coordinates

Vertices in Pixel Coordinates

MANCHESTER 1824

Getting the pixels: scan conversion



Vertices in World Coordinates

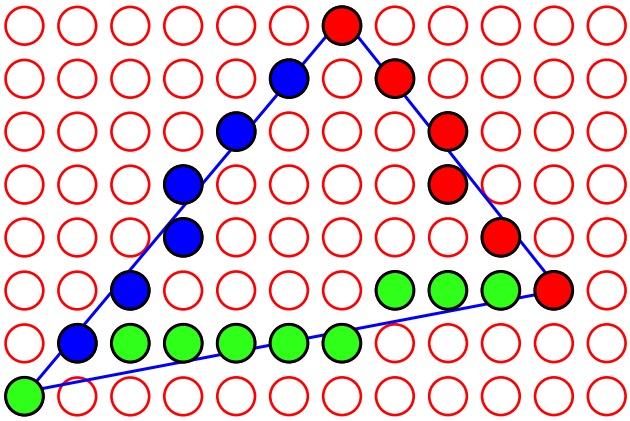
Vertices in Pixel Coordinates

25

MANCHESTER 1824

Scan-converting a line

- We sample the true geometry of the line, and **approximate** it using the nearest pixels available.



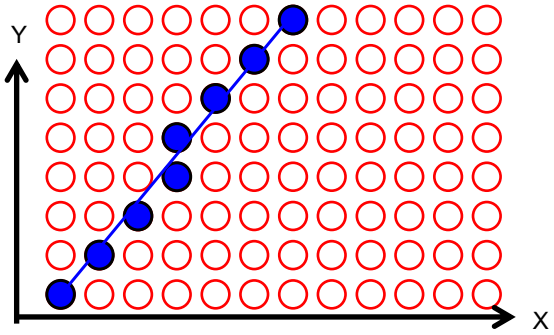
26

MANCHESTER 1824

Bresenham's algorithm

- $y = mx + c$
- As we move horizontally x changes by 1 pixel
- So $y_{n+1} = y_n + m$
- Round y_{n+1} to the nearest pixel

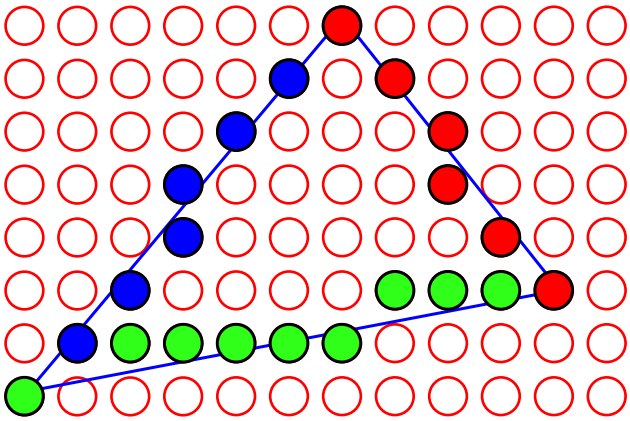
- Need to swap x and y according to gradient of the line
- Bresenham (1965) developed a fast algorithm using only integer arithmetic
- Still in use today



MANCHESTER 1824

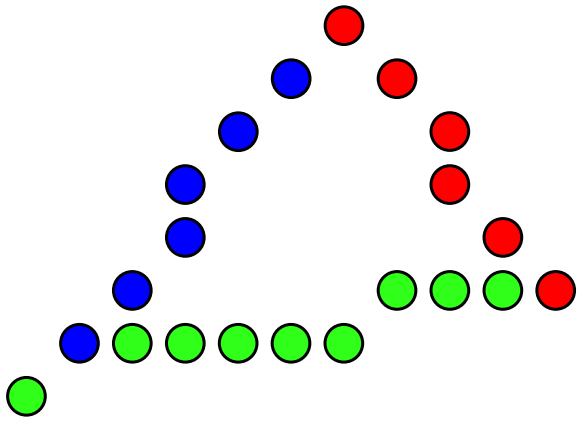
Scan-converting a line

- We sample the true geometry of the line, and **approximate** it using the nearest pixels available.



MANCHESTER 1824

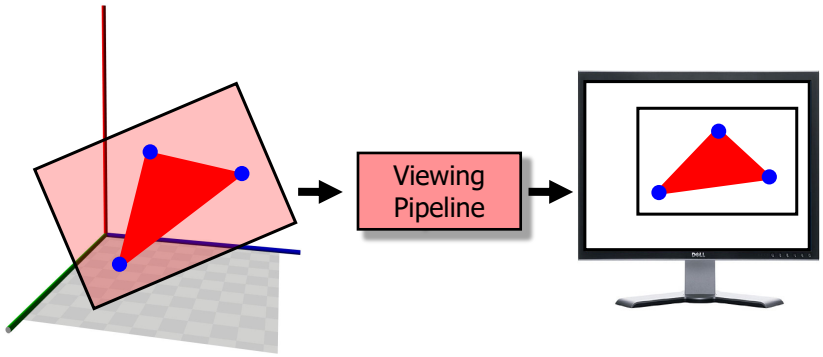
Scan converting a line



29

MANCHESTER 1824

Scan converting a polygon



Vertices in World Coordinates

Vertices in Pixel Coordinates

30

MANCHESTER 1824

Scan converting a polygon

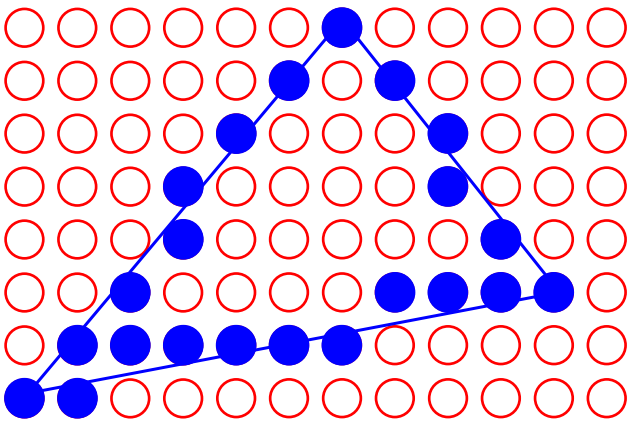
- There are many approaches to scan-converting a polygon, and we'll look at two
- The polygon has been transformed by the viewing pipeline, so we know its (x,y,z) vertex coordinates in screen space
- The (x,y) coordinates corresponds to a pixel position
- The z coordinate is a measure of the vertex's distance from the eye (or "camera"). We won't use that information just yet.

31

MANCHESTER 1824

Scan converting a triangle

- We can scan-convert each of the edges

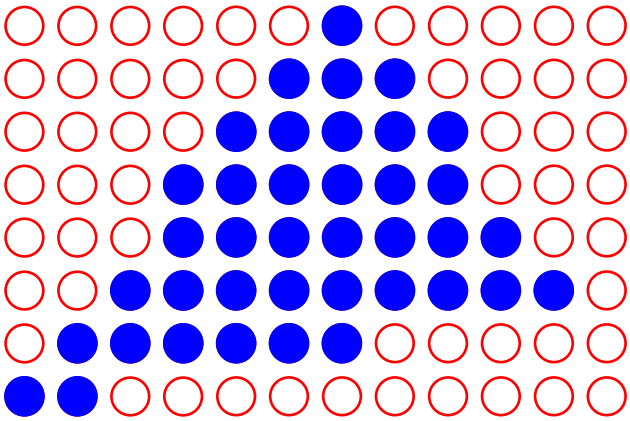


32

MANCHESTER 1824

Scan converting a triangle

- Now we can process each row of pixels and fill in the remaining interior pixels

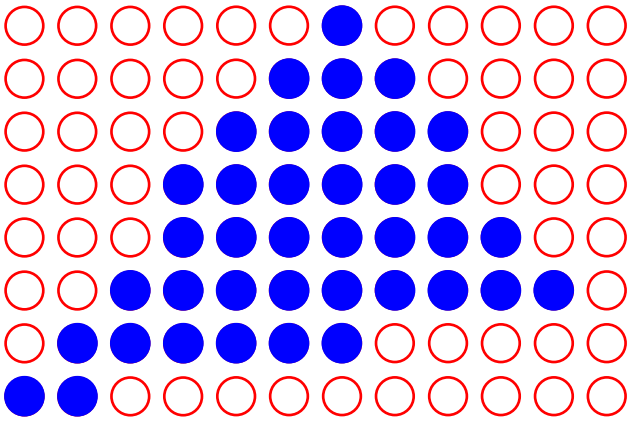


33

MANCHESTER 1824

Scan converting a triangle

- In practice, this naïve approach is never used. There are far more efficient methods, which can be implemented in hardware



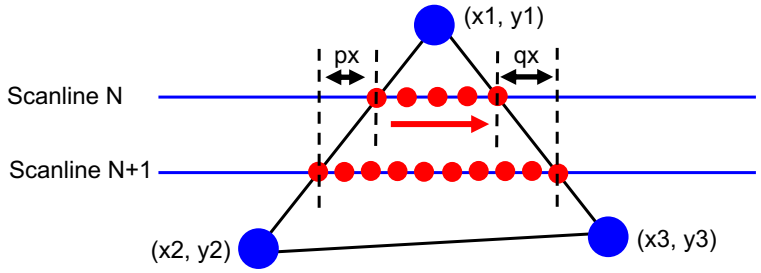
34

MANCHESTER 1824

Scan converting a triangle efficiently

- One example is the "sweep-line" algorithm

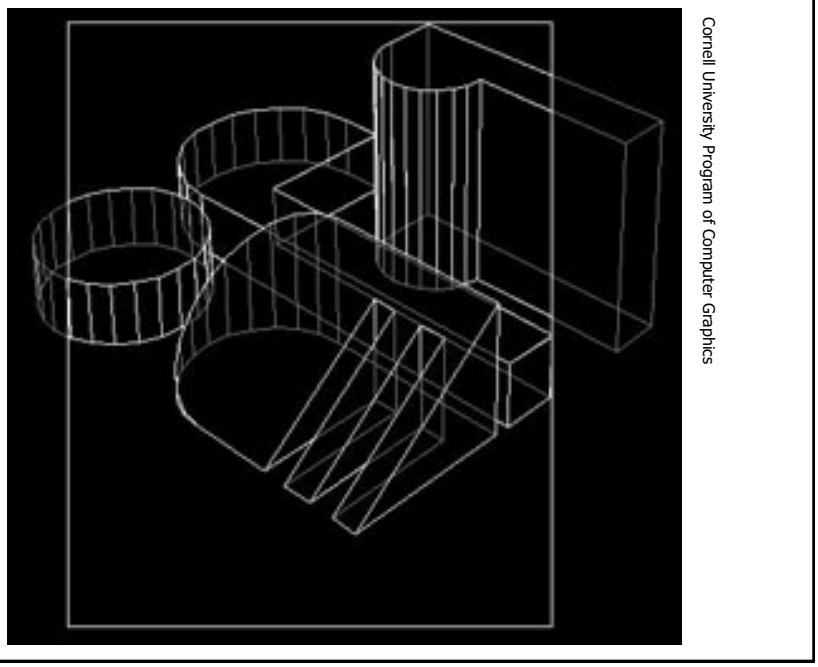
The algorithm steps down a pair of edges, starting in this example at (x1,y1), then goes down scanline by scanline, finding the start and end of the part of the scanline inside the triangle. Then it fills the pixels inside the triangle for that scanline. Efficient because we only need to compute the slopes of the edges (px and qx) once, at the beginning. It is, however, a floating-point algorithm, and we have to keep rounding to the pixel grid.

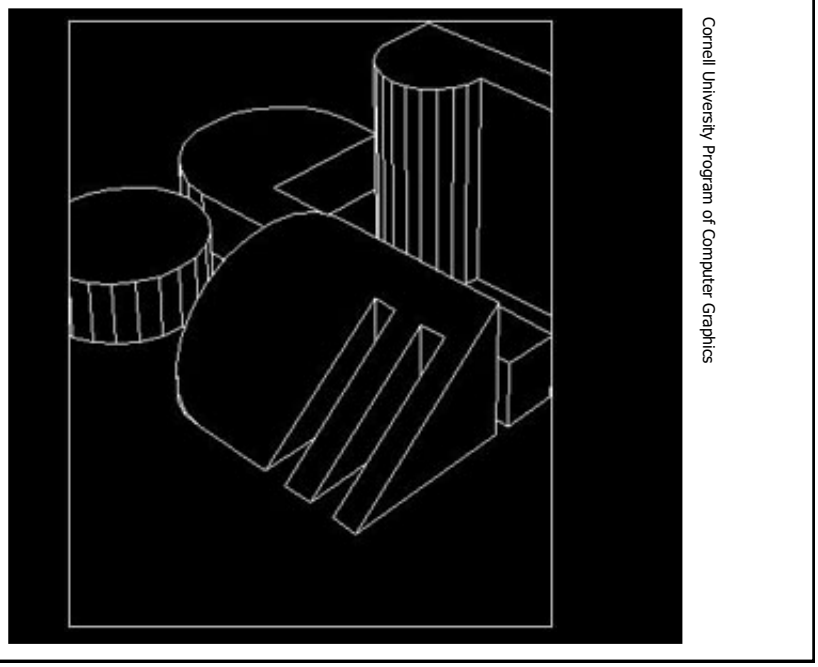


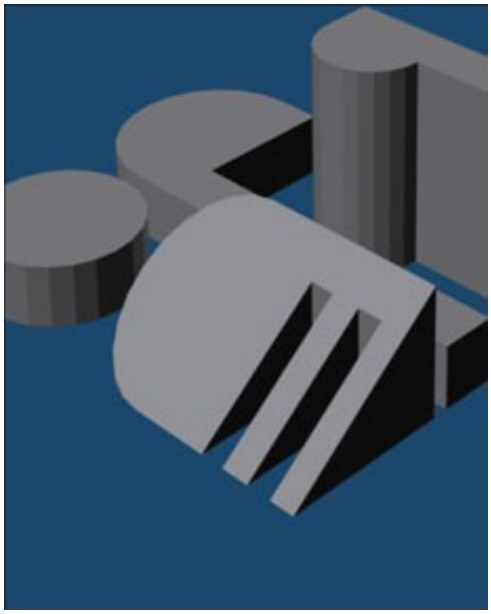
MANCHESTER 1824

Hidden surface removal

- Viewing the world from a particular viewpoint, which parts of the world can we see, and which parts can we not see, because other objects block them?
- There are two fundamentally different approaches to solving this problem:
  - Work in **3D world space**. We work it out geometrically in 3D, and then draw the result. This was the first approach used, 1960-1980s, and it is extremely hard
  - Work in **2D display space**. During scan-conversion, whenever we generate a pixel **P**, we determine whether some **other** 3D object, nearer to the eye, **also** maps to **P**. This is now the standard approach.


Cornell University Program of Computer Graphics


Cornell University Program of Computer Graphics

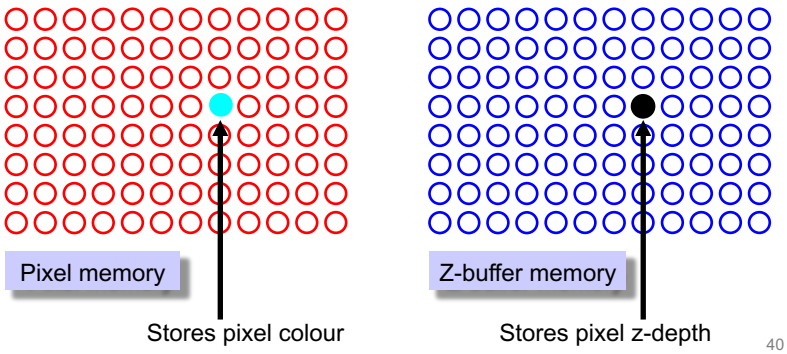


Cornell University Program of Computer Graphics

MANCHESTER 1824

The Z-buffer (aka depth-buffer)

- We introduce a new data structure called the Z-buffer
- For every pixel in the display memory, there is a corresponding entry in the Z-buffer
- The Z-buffer is used to keep a record of the z-value of each pixel



MANCHESTER 1824

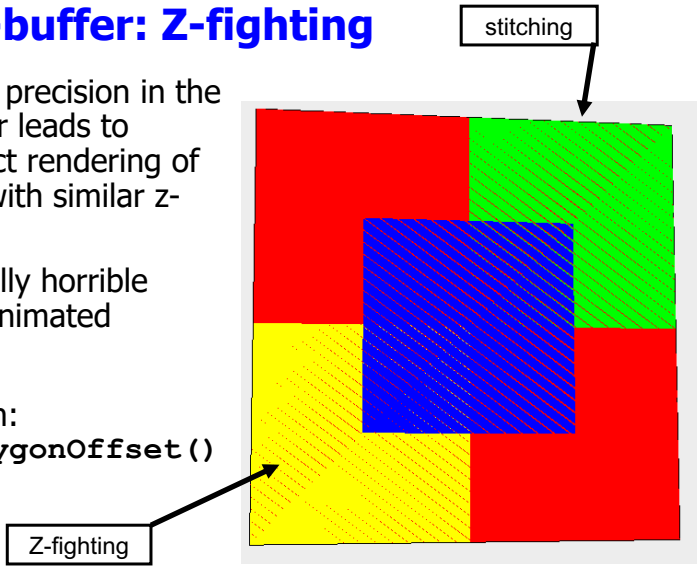
The Z-buffer algorithm

1. Initialise each pixel to desired background colour
2. Initialise each Z-buffer entry to MAXDEPTH (biggest possible number)
3. For each pixel **P** generated during scan-conversion of an object:
   - IF z-coordinate of **P** < **Z-BUFFER[P]** THEN
     // i.e., the part of the 3D object that mapped to **P**
     // is nearer to the eye than any other part of the world
     // that has so far mapped to **P**
     Compute colour of **P** // lighting, texture… later in the course
     Store colour in **P**
     Store (i.e., update) z-coordinate of **P** in **Z-BUFFER[P]**
     ELSE
     // something else has already mapped to **P** and is nearer to us
     // so don't change **P**

MANCHESTER 1824

The Z-buffer: Z-fighting

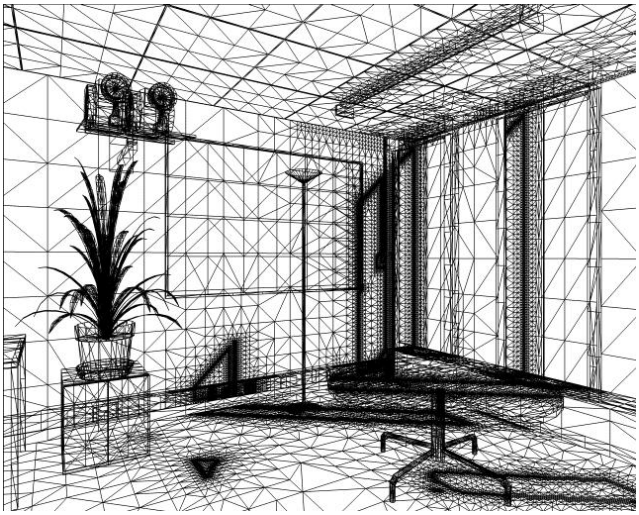
- Lack of precision in the Z-buffer leads to incorrect rendering of pixels with similar z-values
- Especially horrible when animated
- Solution: `glPolygonOffset()`



Grant James, zeuscmd.com

MANCHESTER 1824

Modelling with multiple objects

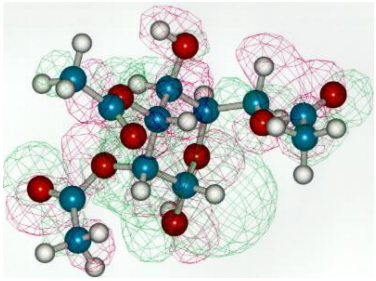


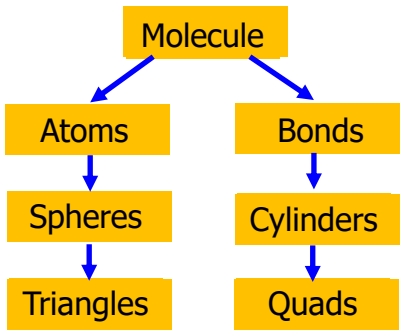
MANCHESTER 1824

Structured models

- We can represent complex things using a hierarchical structure

Ocwd.com

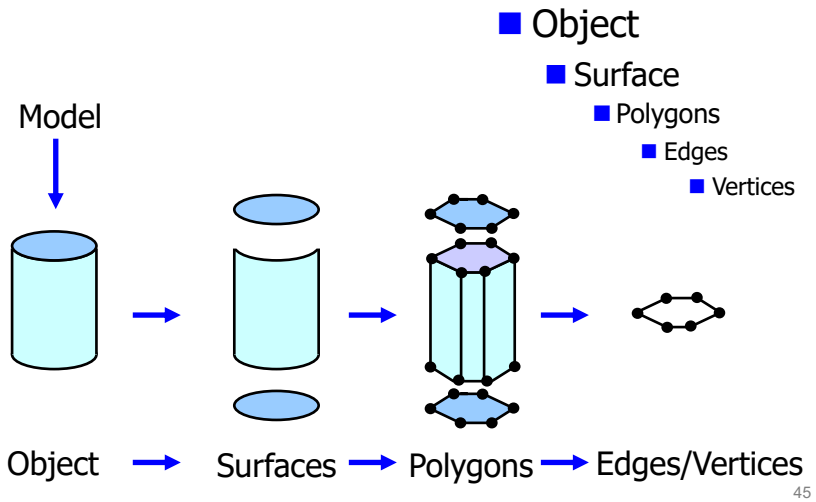




MANCHESTER 1824

Structured polygons

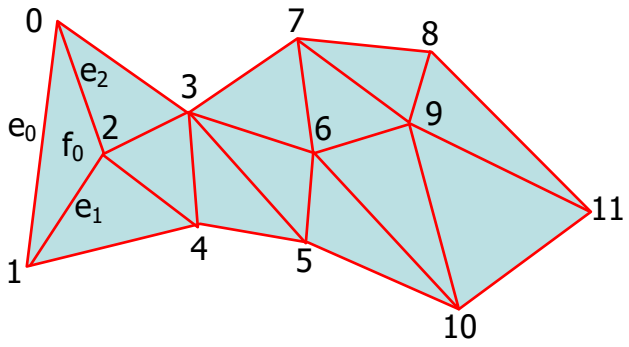
- Maintain a hierarchy of structure



MANCHESTER 1824

General polygon mesh

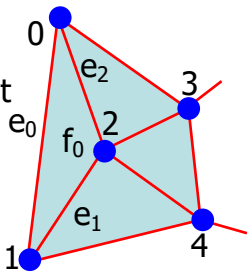
- Flexible way to define linked polygons

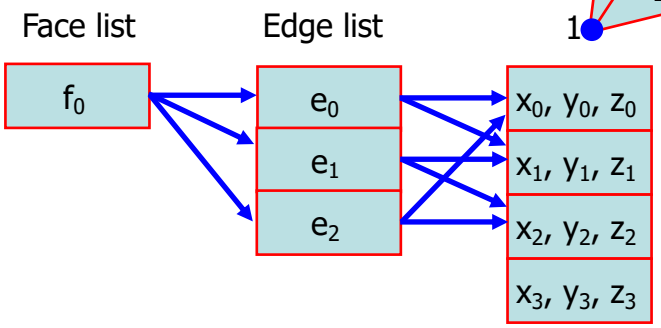


MANCHESTER 1824

Mesh data structure

- **vertex list**
- **edge list**, indexing into the **vertex** list
- **face list**, indexing into the **edge** list



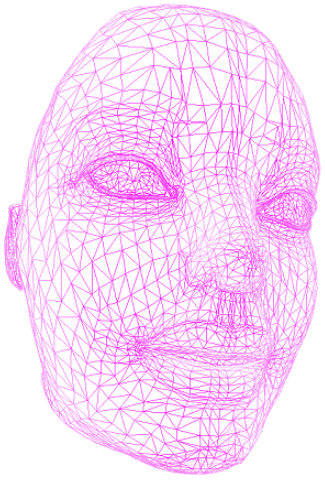


MANCHESTER 1824

Practicalities: file formats

- Meshes are often big
- Many different file formats
- The example here is a Wavefront "obj" file:
  - 2953 vertices

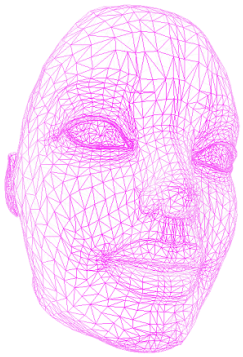
```
##  Three-D Library generated .obj file
##  data/womanheadH
##
v 0.698214 -0.204674 2.690314
v 0.685697 -0.198298 2.709758
v 0.715357 -0.148323 2.717670
v 0.716248 -0.176445 2.695745
v 0.666189 -0.129036 2.739556
v 0.704262 -0.140100 2.727099
v 0.674602 -0.190075 2.719187
v 0.655446 -0.197328 2.735766
v 0.791905 -0.143497 2.688993
```



MANCHESTER 1824

Example: the obj file format

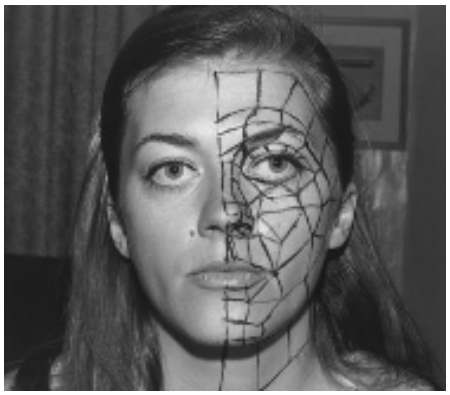
- List of vertices
  - **v** x y z
- List of vertex normals
  - **vn** x y z
- List of vertex texture coordinates
  - **vt** s t
- List of faces
  - **f** v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3...
- List of groups
  - **g** f1 f2 f3
- Full details: http://www.dcs.ed.ac.uk/home/mxr/gfx/3d/OBJ.spec



Details of obj spec are not examinable

MANCHESTER 1824

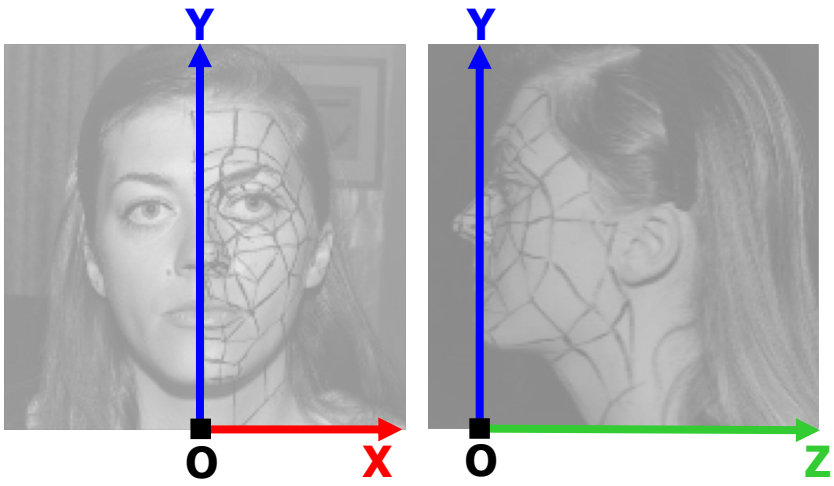
Creating Geometry: (1) by hand



- Making a polygon mesh by hand – Mme Sylvie Gouraud (1971)

MANCHESTER 1824

Creating Geometry: by hand



51

MANCHESTER 1824

Mme Gouraud (1971)

52

MANCHESTER 1824

Case Study: manual reconstruction

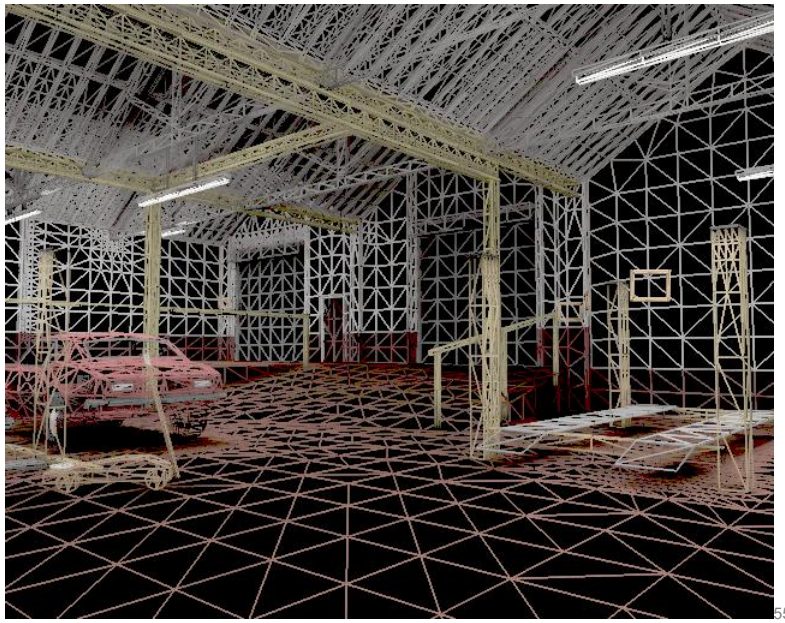
- UoM project with Greater Manchester Police

53

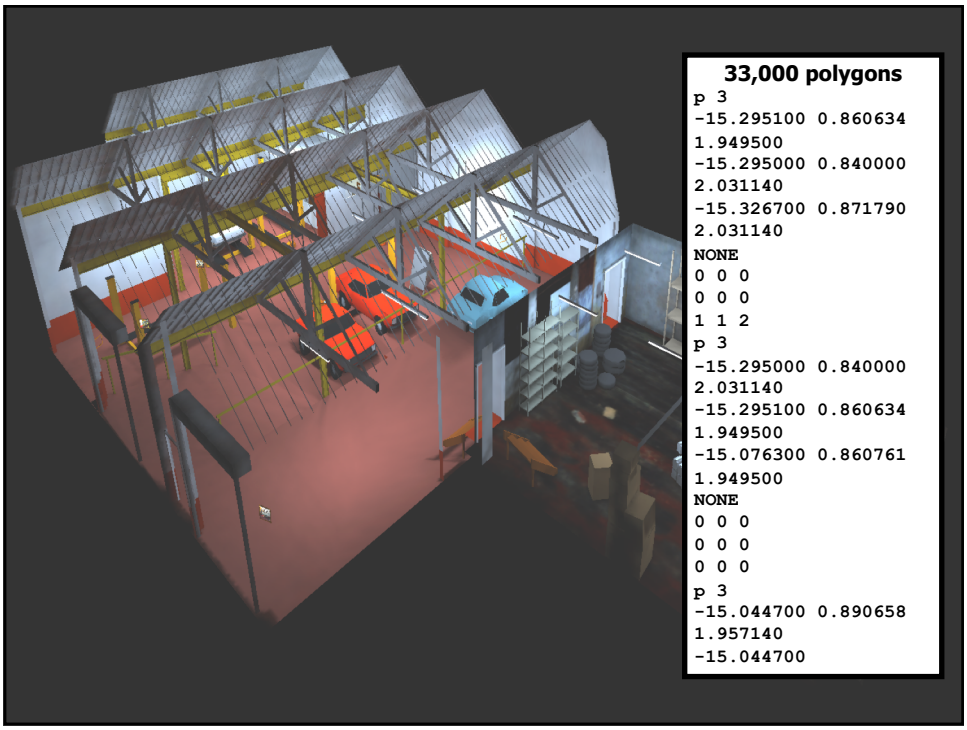
MANCHESTER 1824

54

MANCHESTER 1824



55



MANCHESTER 1824

- Manual reconstruction will usually omit much detail

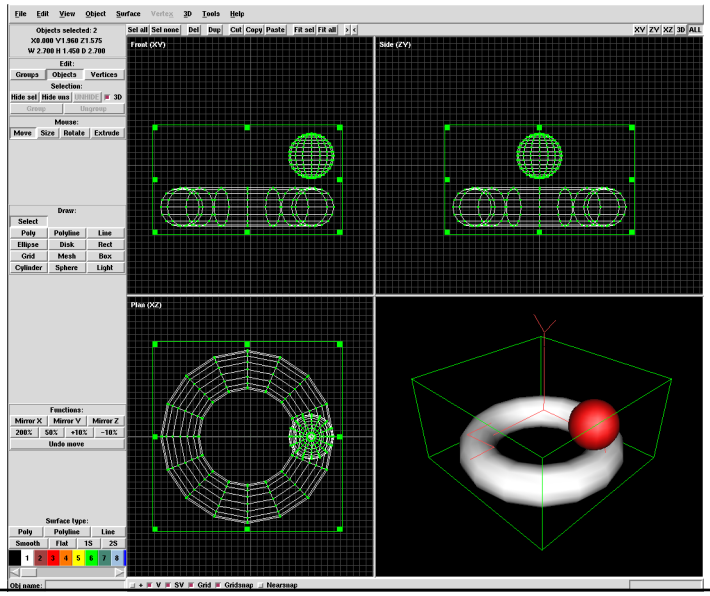
MANCHESTER 1824

Chestergate: Manual reconstruction

- Reconstruction performed entirely manually
  - Measurements from architectural plans
    - CAD model created in textual format
  - Measurements from photographs
  - Images and textures scanned from photographs
  - 20 person-months of effort (approx)
  - How to do better (research topic)

MANCHESTER 1824

Creating Geometry: (2) Interactive modelling



MANCHESTER 1824

Creating Geometry: Interactive modelling

MANCHESTER 1824

Creating Geometry: (3) Directly from the world

MANCHESTER 1824

Mme Gouraud (laser scan)

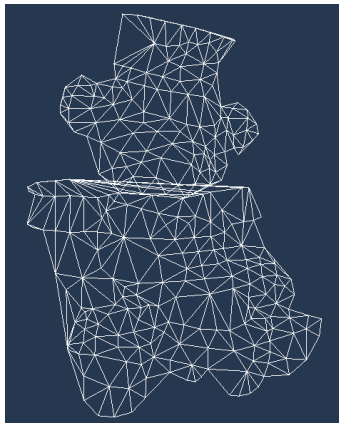
MANCHESTER 1824

63

MANCHESTER 1824

Creating Geometry: (4) Directly from the world

- Capture geometry directly from images/video
- Important area of ongoing research (see COMP37111)



64

MANCHESTER 1824

What do do next

1. OpenGL manual – Tutorial Chapters 1-6
2. Do Coursework 1 (deadline 15 Feb)
3. Start looking at Lab 1