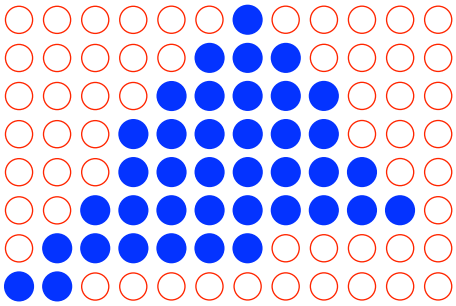
MANCHESTER 1824

COMP27112

Computer Graphics and Image Processing



4: Polygons and pixels

Toby.Howard@manchester.ac.uk

1

MANCHESTER 1824

Introduction

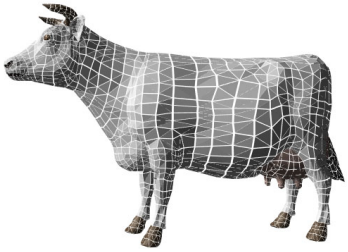
- We'll look at
  - Properties of polygons: convexity, winding, faces, normals
  - Scan conversion of polygons
  - Hidden surface removal with the Z-buffer
  - Data structures for polygonal models
  - Where do we get polygons from?

2

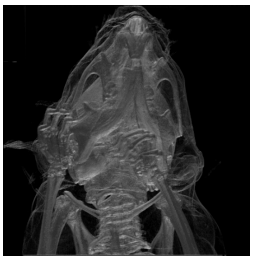
MANCHESTER 1824

Polygons as building blocks

- The basic unit of 2D raster graphics is the pixel
- The most common basic unit of 3D graphics is the polygon
- The most common polygon is the triangle



- 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)

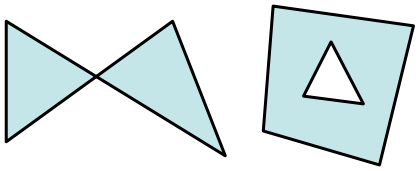


3

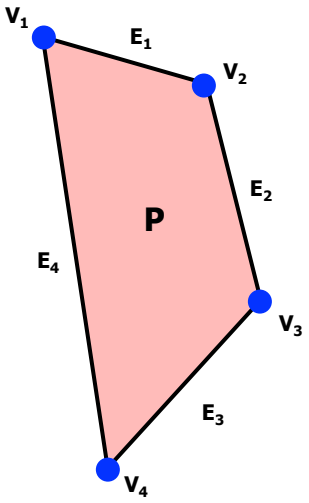
MANCHESTER 1824

What is a polygon?

- An ordered set of vertices (V_1…V_4)
- A set of edges between each pair of vertices (E_1…E_4)
- The polygon (P) is then the space bounded by the vertices
- Some polygons are not so simple…



- So we avoid polygons like that

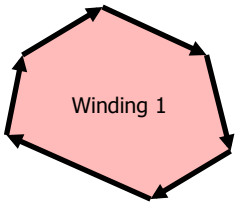


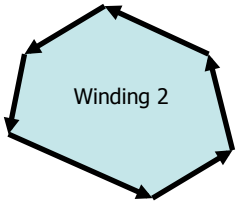
4

MANCHESTER 1824

Polygon attributes: winding

- The order in which the edges connect the vertices gives a polygon a "winding"
- There are two winding orders: **clockwise** and **counter clockwise** (but which is which depends on which side you view from)
- We need to use a consistent winding for **all** the polygons in a scene



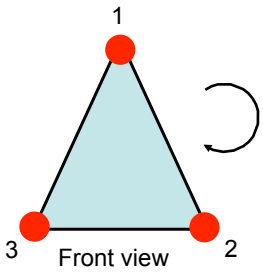


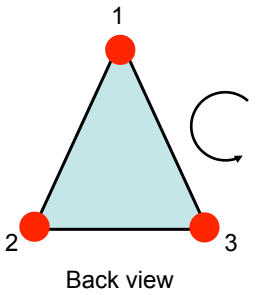
5

MANCHESTER 1824

Polygon attributes: faces

- If we have a consistent winding, then the polygon has two defined faces – a **front** and a **back**
- If all the vertices lie on plane, then we have a planar polygon (which is almost always what we want)
- Triangles are always planar



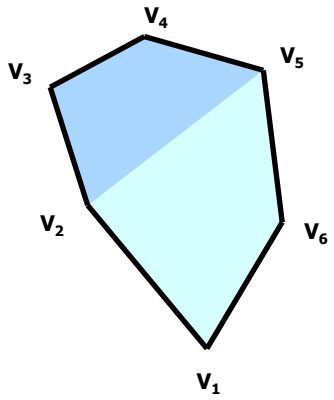


6

MANCHESTER 1824

Polygon attributes: faces

- Non-planar polygons have vertices that do not all lie on a single plane
- Although they have two "sides" they don't have obvious faces in the same way
- This can cause problems with some graphical algorithms…
- …so generally we assume our polygons are planar
- Most graphics systems will only guarantee to draw planar polygons



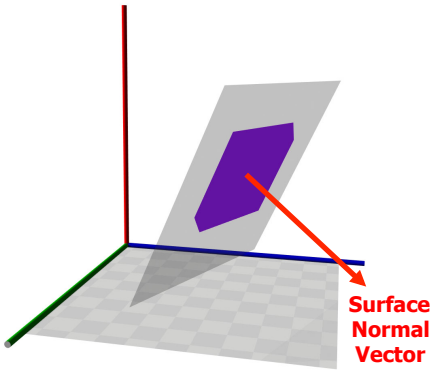
Here, [V_1 V_2 V_5 V_6] lie on one plane, but [V_2 V_3 V_4 V_5] lie on another. This is a non-planar polygon

7

MANCHESTER 1824

Polygon attributes: surface normal

- With a consistent winding order, we can define a property called the **surface normal**
- This 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…



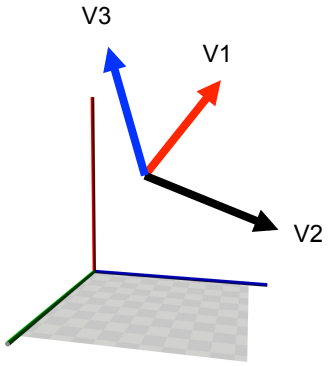
8

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

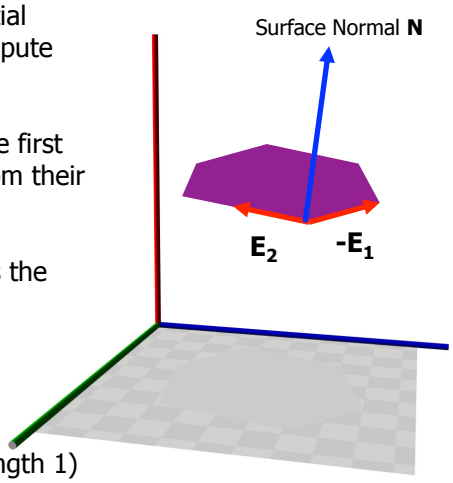
9

MANCHESTER 1824

Finding the surface normal

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

   $N = E_2 \times -E_1$
4. We then almost always normalize N (make its length 1)

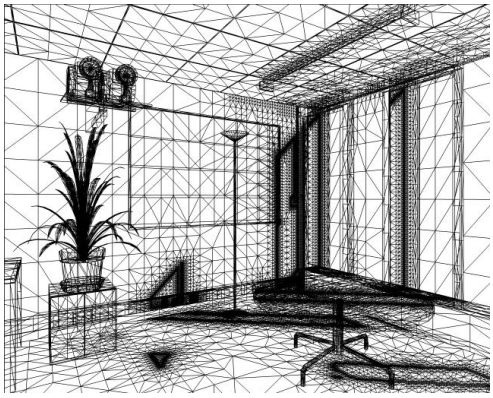


10

MANCHESTER 1824

Representing things 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…

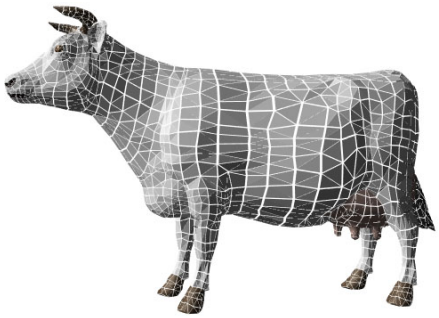


11

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…?
- Leads to 'brute force' rendering, and makes interaction with the model complex



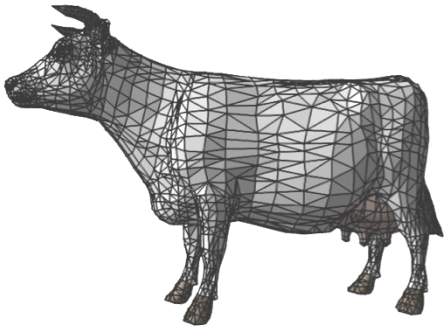
18,000 separate vertices, many repeated

12

MANCHESTER 1824

Polygon meshes

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

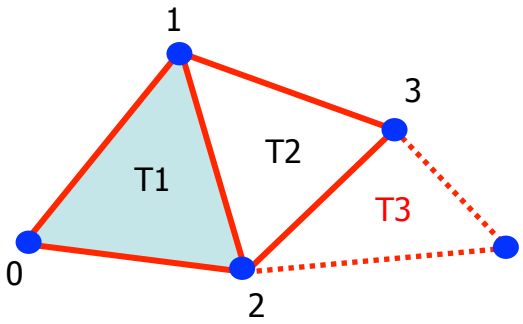


6,000 separate vertices, mostly unique/shared

13

MANCHESTER 1824

Meshes: triangle strips



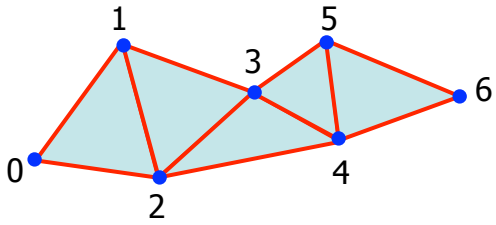
- Add one new **vertex**…
- … get one new **triangle**

14

MANCHESTER 1824

Meshes: triangle strips

- Collection of linked triangles
- Very widely used – efficient



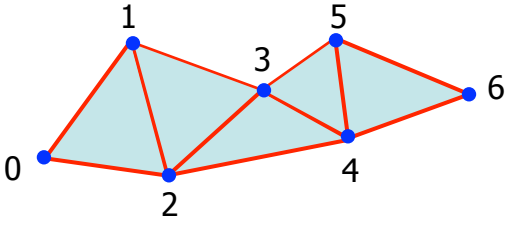
- **N** linked triangles can be defined using **N+2** vertices – compared with **3N** vertices if each triangle were defined separately

15

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 other vertices */
glEnd();
```

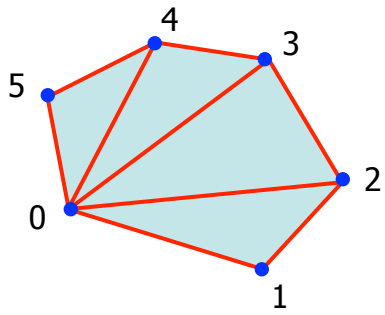


16

MANCHESTER 1824

Meshes: triangle fan

- Collection of linked triangles



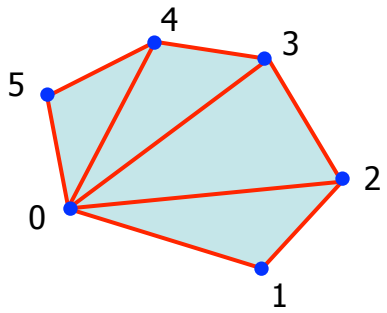
- **N** linked triangles can be defined using **N+2** vertices – compared with **3N** vertices if each triangle were defined separately

17

MANCHESTER 1824

Triangle fan in OpenGL

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

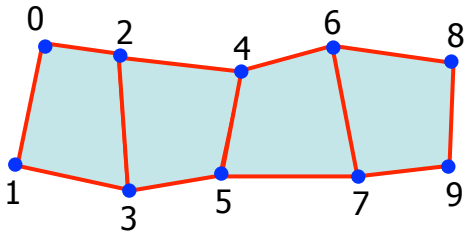


18

MANCHESTER 1824

Meshes: quadrilateral strips

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



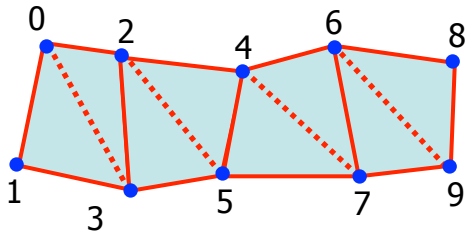
- **N** quads can be defined using **2N+2** vertices, compared with **4N** separate vertices

19

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);
  /* and other vertices */
glEnd();
```



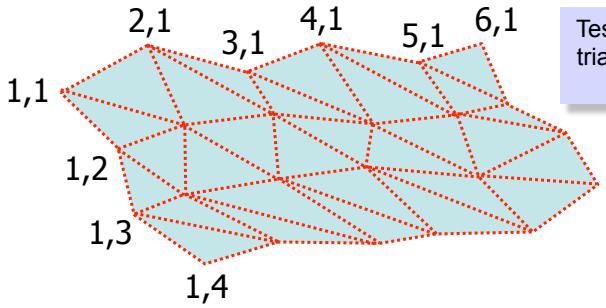
Tessellated into triangles during rendering

20

MANCHESTER 1824

Meshes: quadrilateral meshes

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



Tessellated into triangles during rendering

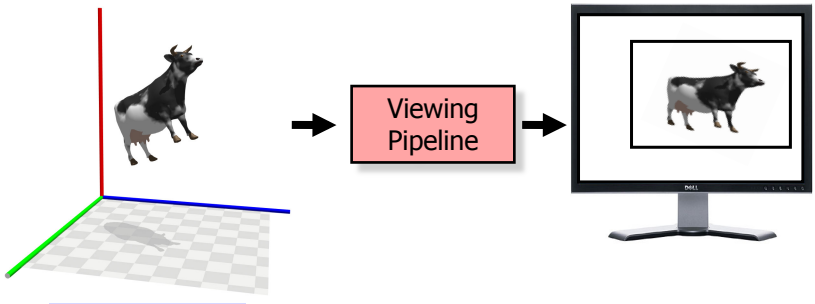
- **N** x **M** quads can be defined using **(N+1) * (M+1)** vertices, compared with **4*M*N** separate vertices

21

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



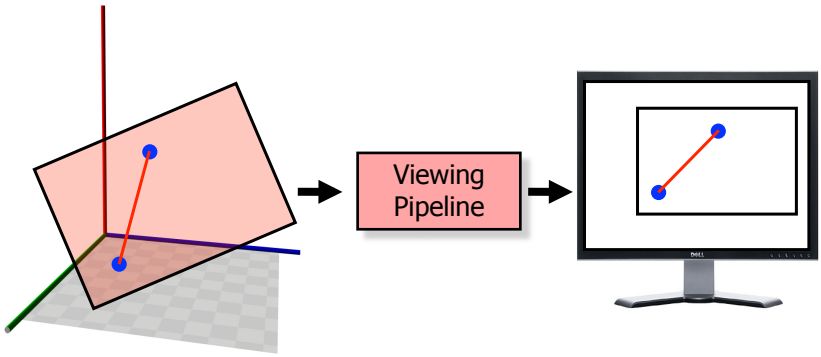
Vertices in World Coordinates

Vertices in Pixel Coordinates

22

MANCHESTER 1824

Getting the pixels: scan conversion



Vertices in World Coordinates

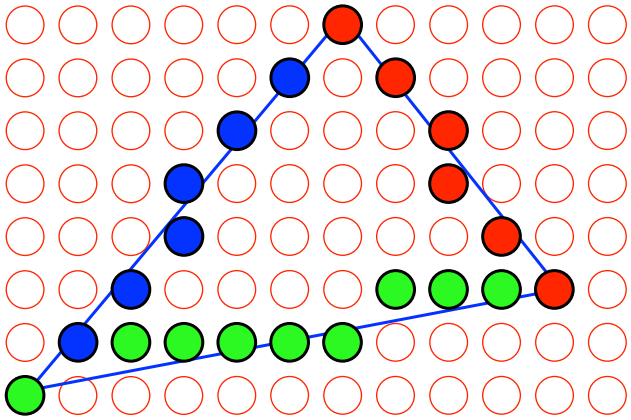
Vertices in Pixel Coordinates

23

MANCHESTER 1824

Scan-converting a line

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



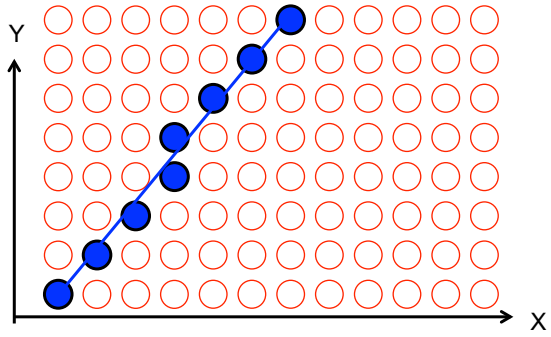
24

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

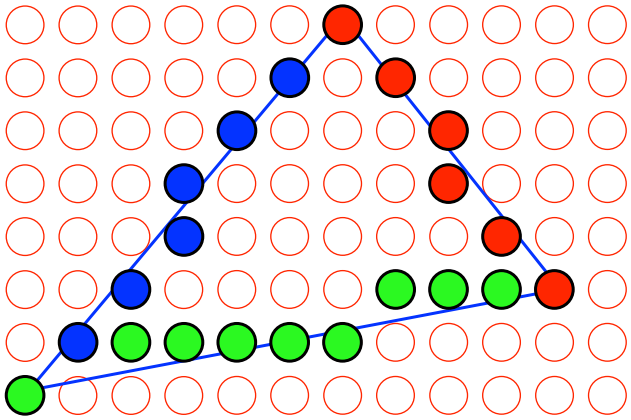


25

MANCHESTER 1824

Scan-converting a line

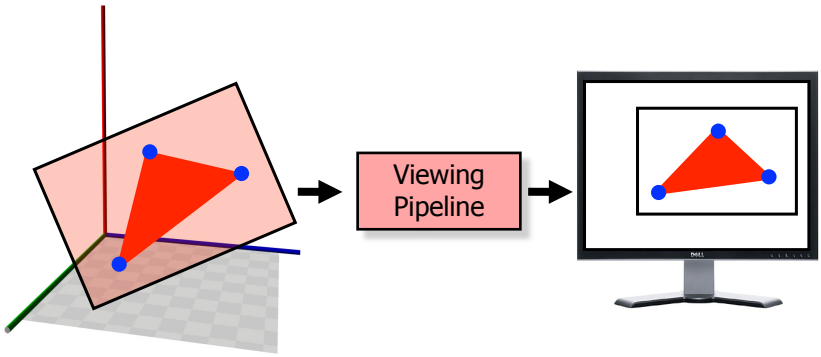
- We sample the true geometry of the line, and **approximate** it using the nearest pixels available.



26

MANCHESTER 1824

Scan converting a polygon



Vertices in World Coordinates

Vertices in Pixel Coordinates

27

MANCHESTER 1824

Scan converting a polygon

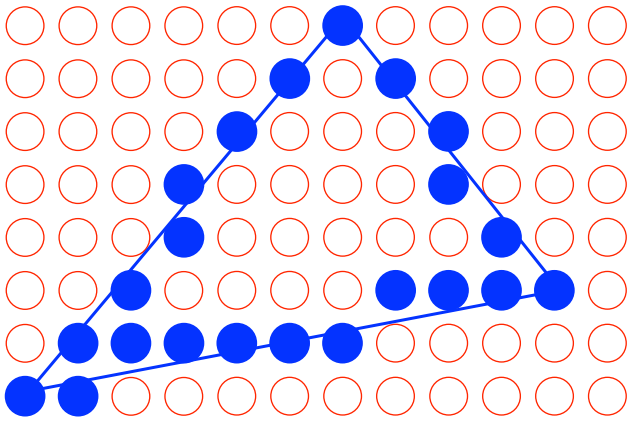
- There are many approaches to scan-converting a polygon, and we'll look at two
- The polygon has been transformed by the viewing pipeline, so we know its (x,y,z) vertex coordinates in screen space
- The (x,y) coordinates corresponds to a pixel position
- The z coordinate is a measure of the vertex's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

28

MANCHESTER 1824

Scan converting a triangle

- We can scan-convert each of the edges

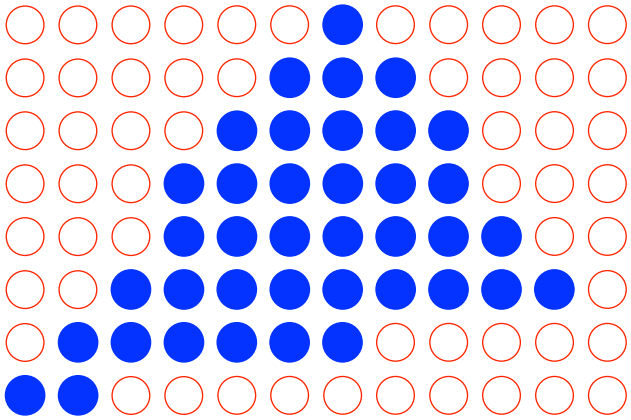


29

MANCHESTER 1824

Scan converting a triangle

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

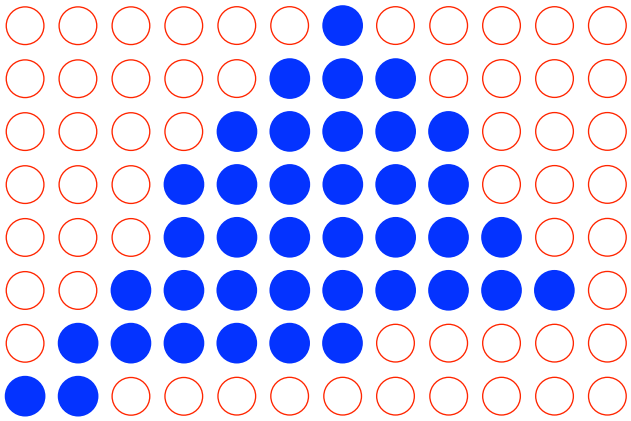


30

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



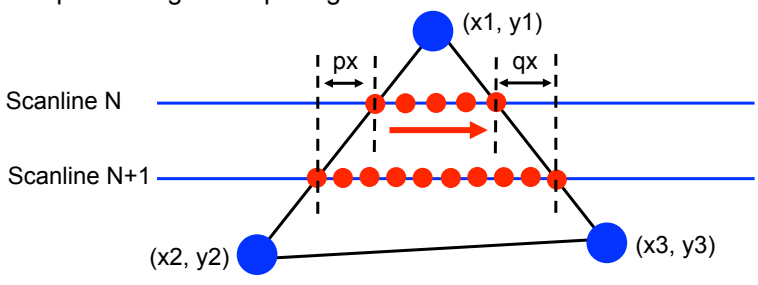
31

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.



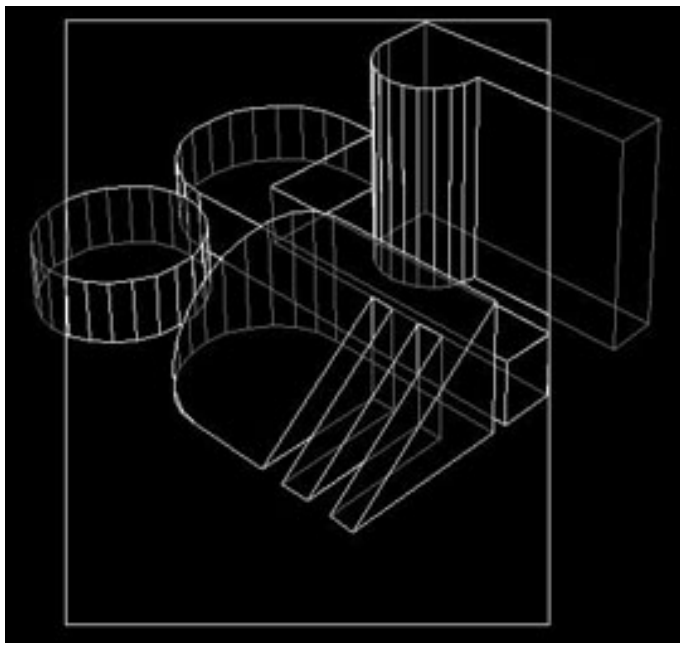
32

MANCHESTER 1824

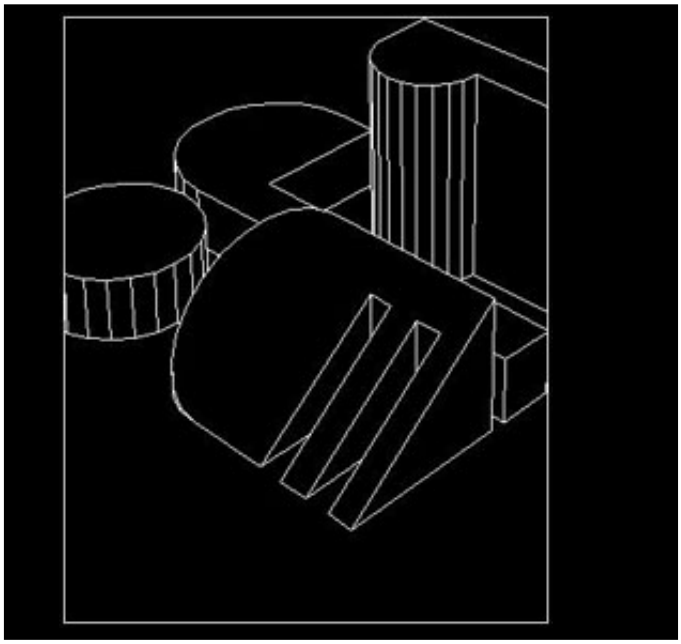
Hidden surface removal

- Question: 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 (partially) block them?
- There are two fundamentally different approaches that we can take to solving this problem:
  - We can solve it in **world space**. We can try to work it out geometrically in 3D, and then draw the result. This was the first approach used, 1960-1980ish, and it is extremely hard
  - We can do it in **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.

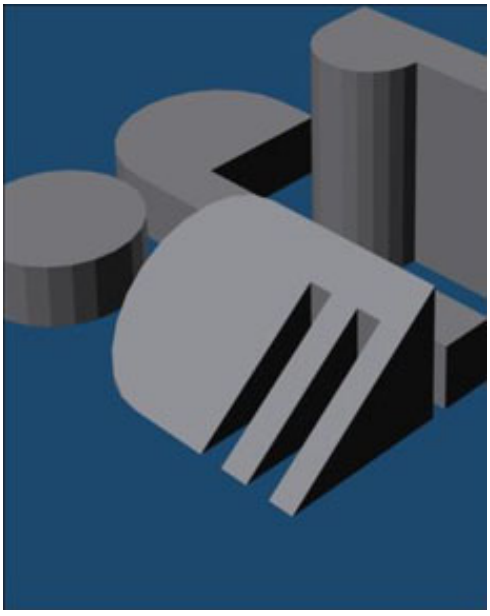
33



Cornell University Program of Computer Graphics



Cornell University Program of Computer Graphics

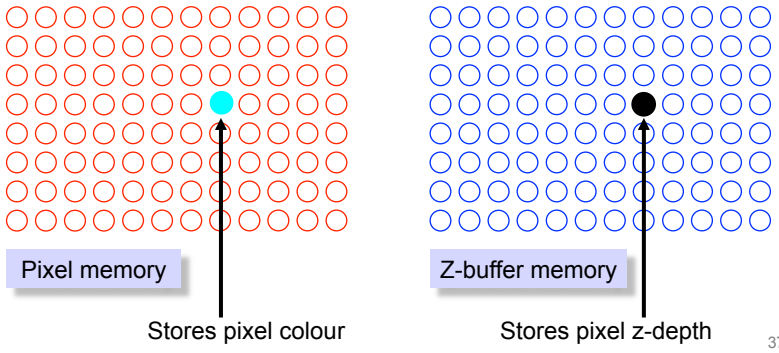


Cornell University Program of Computer Graphics

MANCHESTER 1824

The Z-buffer (or 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



37

MANCHESTER 1824

The Z-buffer algorithm

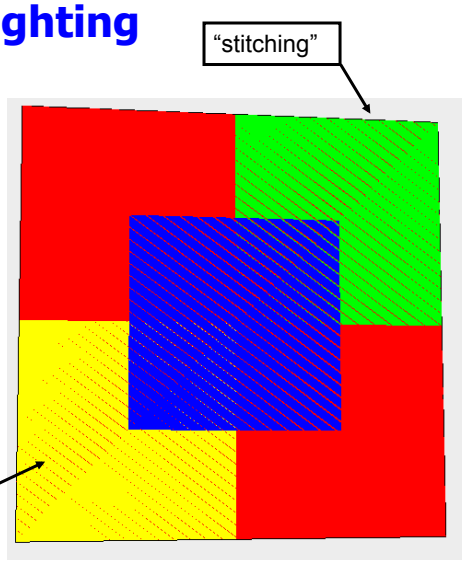
1. Initialise each pixel to desired background colour
2. Initialise each Z-buffer entry to MAXDEPTH (biggest possible number)
3. For each pixel **P** generated during scan-conversion of an object:
   - IF z-coordinate of **P** < **Z-BUFFER[P]** THEN
     // i.e., the part of the 3D object that mapped 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**

38

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()`

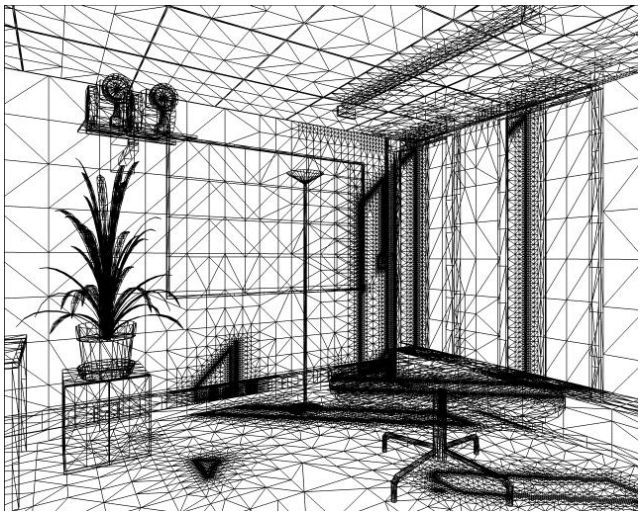


Grant James, zeuscmd.com

39

MANCHESTER 1824

Modelling with multiple objects

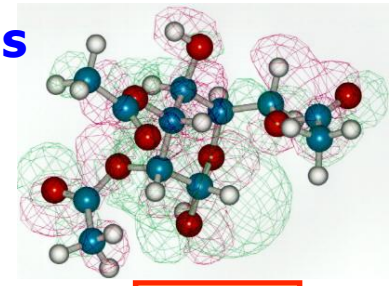


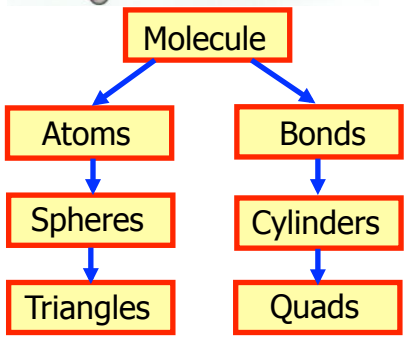
40

MANCHESTER 1824

Structured models

- We can represent complex models using a hierarchical structure
- ~~Later in the course we'll look at how to give structure to complex models, using scene graphs~~ (no, I've decided not to cover this.)
- For now we'll consider data structures for keeping track of polygons



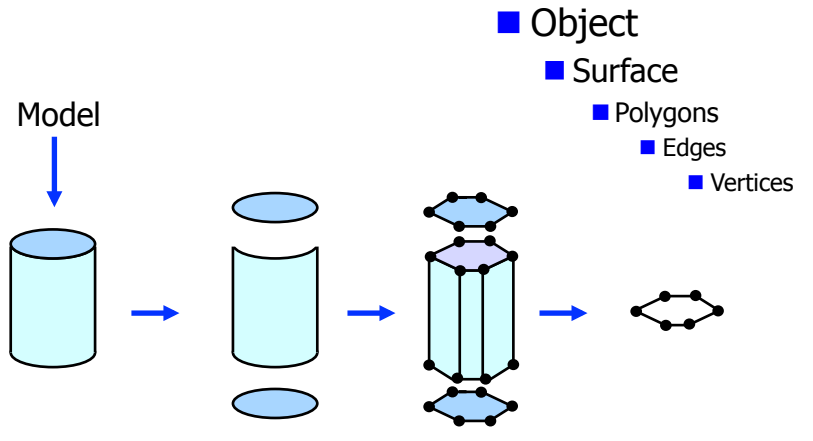


41

MANCHESTER 1824

Structured polygons

- Maintain a hierarchy of structure

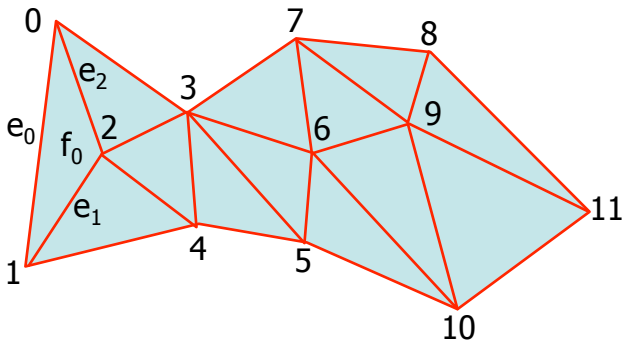


42

MANCHESTER 1824

General polygon mesh

- Flexible way to define linked polygons

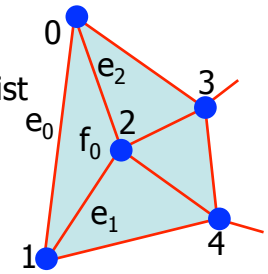


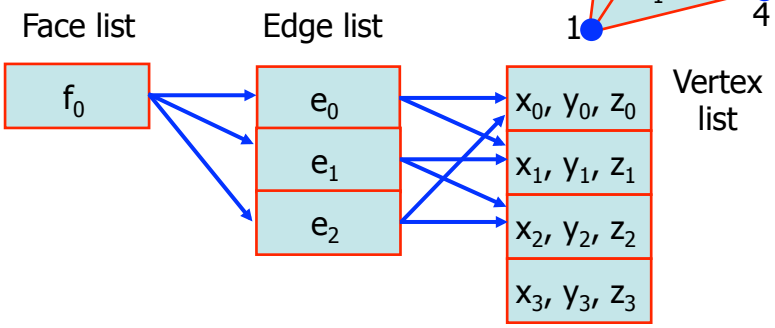
43

MANCHESTER 1824

Mesh data structure

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





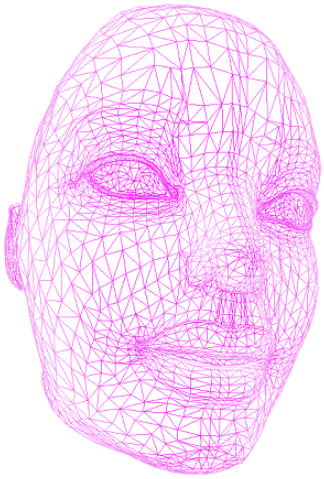
44

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
```

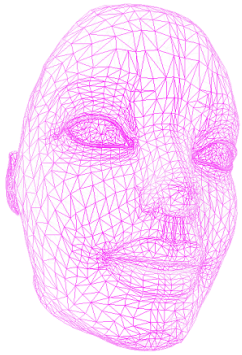


45

MANCHESTER 1824

Example: the obj file format

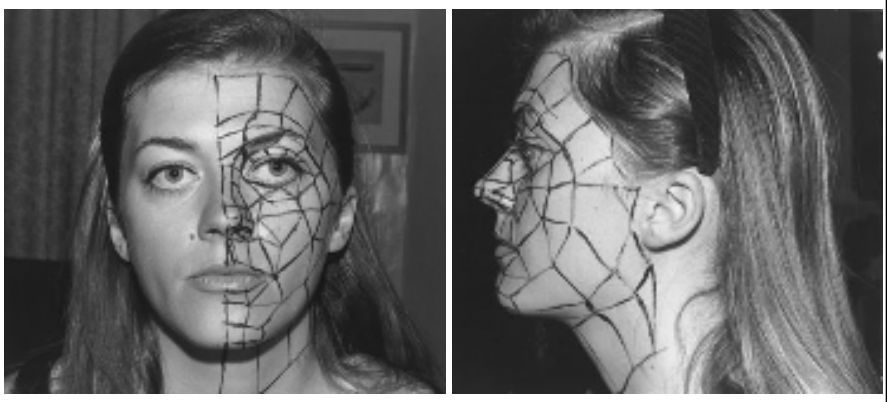
- List of vertices
  - **v** x y z
- List of vertex normals
  - **vn** x y z
- List of vertex texture coordinates
  - **vt** s t
- List of faces
  - **f** v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3...
- List of groups
  - **g** f1 f2 f3
- Browse the full details, and get a feel for the problem, at http://www.dcs.ed.ac.uk/home/mxr/gfx/3d/OBJ.spec



46

MANCHESTER 1824

Creating Geometry: (1) by hand

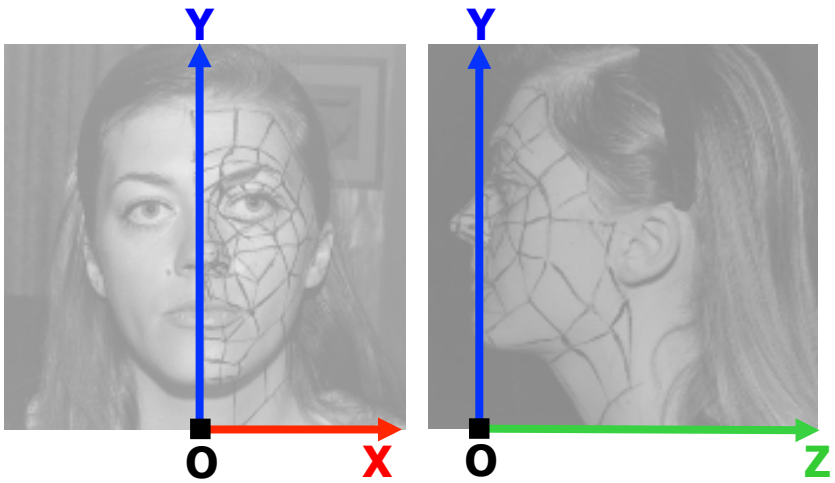


- Making a polygon mesh by hand – Mme Sylvie Gouraud (1971)

47

MANCHESTER 1824

Creating Geometry: by hand



48

MANCHESTER 1824

Mme Gouraud (1971)

49

MANCHESTER 1824

Case Study: manual reconstruction

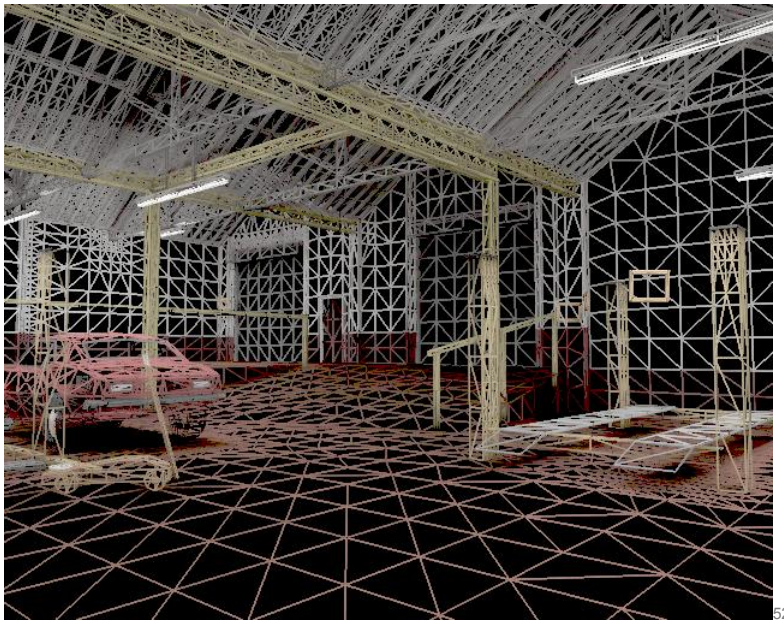
- Chestergate: Project with Greater Manchester Police

50

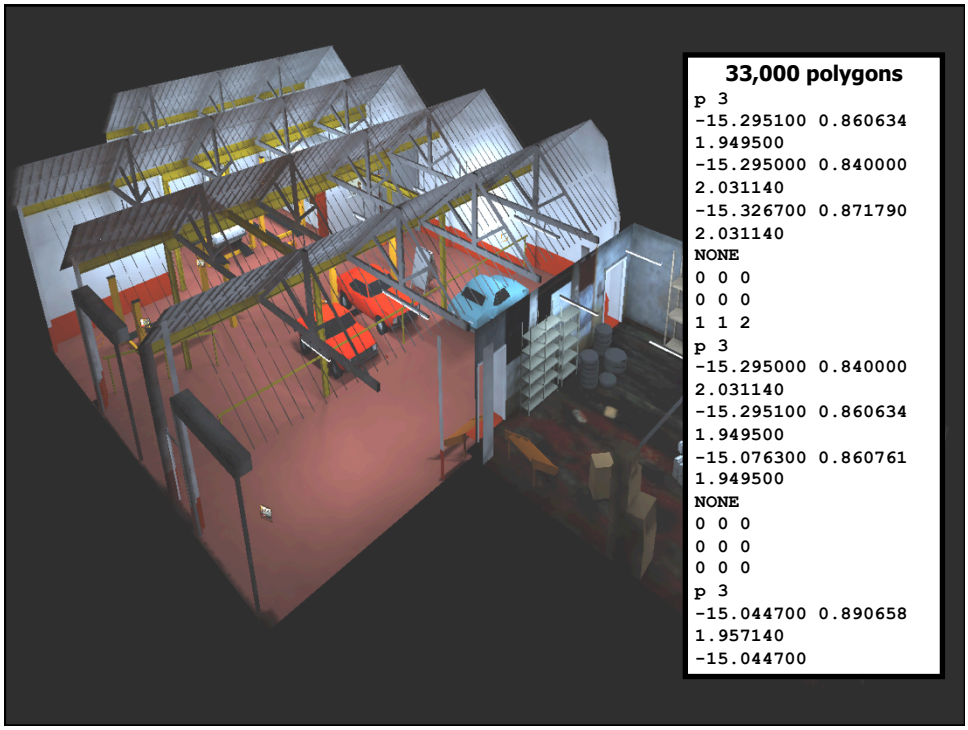
MANCHESTER 1824

51

MANCHESTER 1824



52



MANCHESTER 1824

- Manual reconstruction may omit much detail
  - This is very difficult to avoid (but how could we improve this?)

54

MANCHESTER 1824

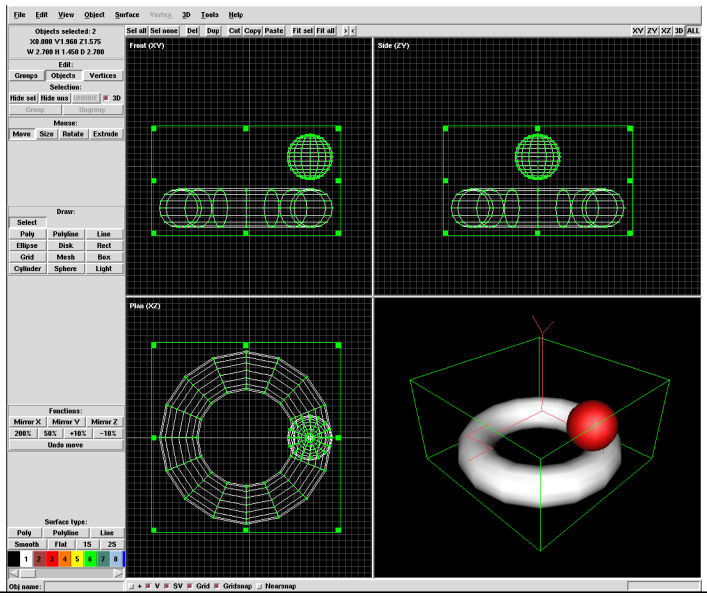
Chestergate: Manual reconstruction

- 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)

55

MANCHESTER 1824

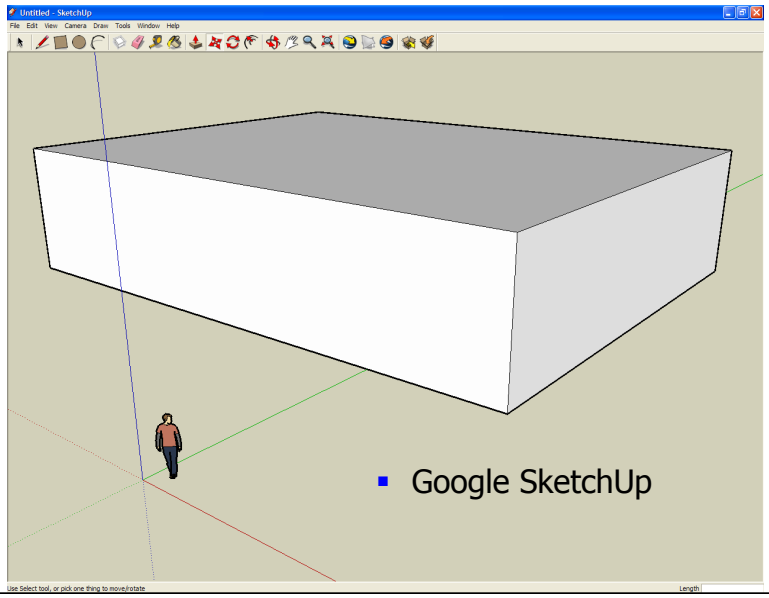
Creating Geometry : (2) Interactive modelling



56

MANCHESTER 1824

Creating Geometry : Interactive modelling



- Google SketchUp

57

MANCHESTER 1824

Creating Geometry: (3) Directly from the world

58

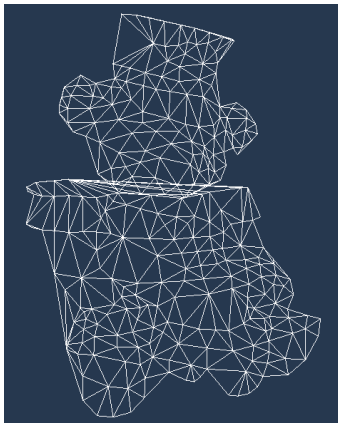
MANCHESTER 1824

Mme Gouraud (1999)

59

MANCHESTER 1824

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



60