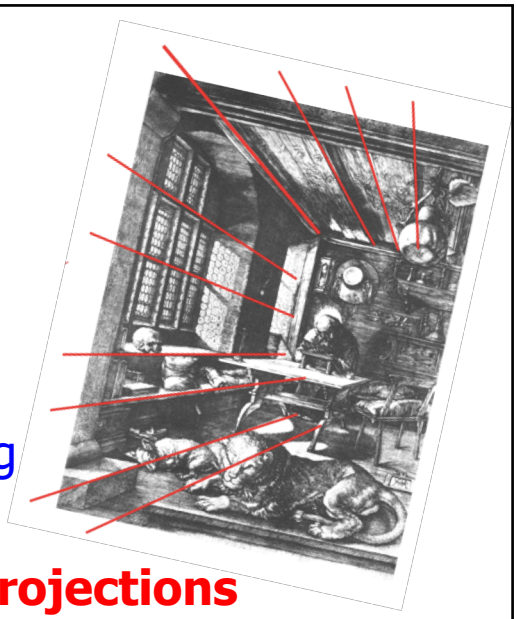
MANCHESTER 1824

COMP27112

Computer Graphics and Image Processing



6: Viewing 2: projections

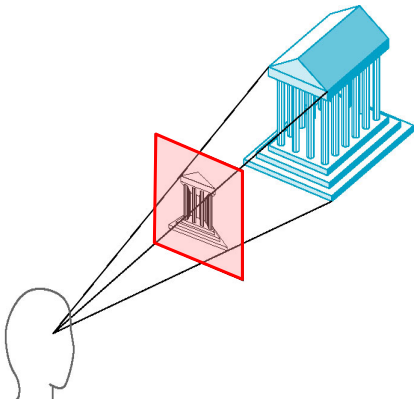
Toby.Howard@manchester.ac.uk

MANCHESTER 1824

Introduction

- In part 2 of our study of Viewing, we'll look at

1. Recap of Lecture 5
2. The theory of geometrical planar projections
3. Classes of projections
   1. Perspective
   2. Parallel
4. Basic mathematics of projections
5. Matrix representations of projections
6. Projection in OpenGL



Temple figures courtesy: Prof. Ed Angel, University of New Mexico

MANCHESTER 1824

Recap on 3D Viewing

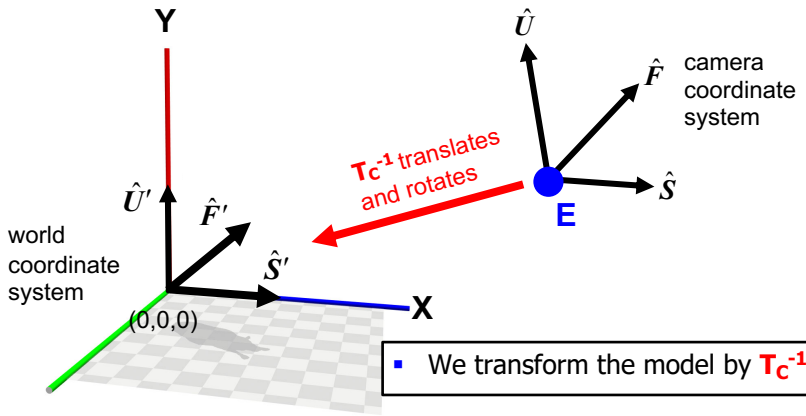
Lecture 5

	Real World	Computer Graphics
1	Arrange the scene into the desired composition	Set Modelling Transformation
2	Point the camera at the scene	Set Viewing Transformation
3	Choose the camera lens, or adjust the zoom	Set Projection Transformation
4	Determine the size and shape of the final photograph	Set Viewport Transformation

MANCHESTER 1824

Deriving the viewing transformation

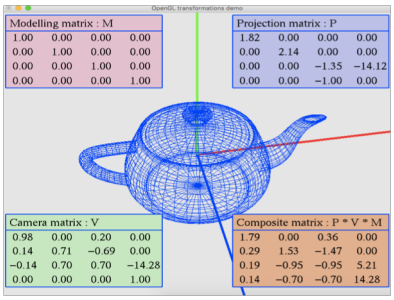
- Now we have the camera axes, we can derive T_C^{-1} which translates & rotates the $\hat{S}\,\hat{U}\hat{F}$ camera system to (0,0,0):



- We transform the model by T_C^{-1}

MANCHESTER 1824

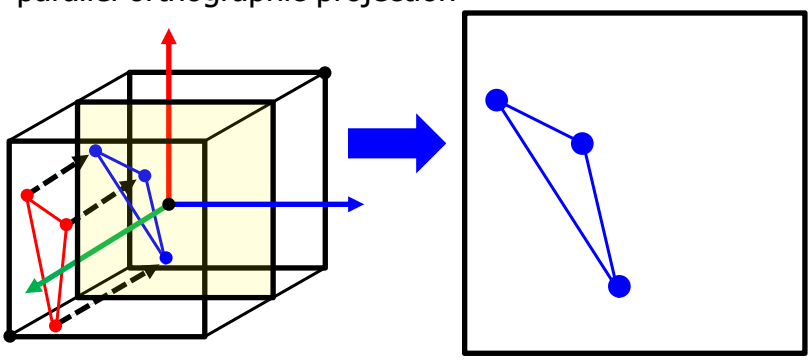
- Do experiments with the modelling and the viewing matrices
- /opt/info/courses/OpenGL/lectureExamples/transformation4
- control keys:
  - `rotate: x y z X Y Z`
  - `translate: a b c A B C`
  - `scale: s t u S T U`
  - `reset: 0`
  - `make singular (y=0): .`
  - `move eye point: i j k I J K`
  - `move look point: l m n L M N`
  - `field of view: < >`
  - `perspective terms in model matrix: p q r P Q R`
  - `change model: +`
  - `change view up vector: v V`



MANCHESTER 1824

OpenGL's default projection

- OpenGL renders the model to the screen with its default parallel orthographic projection



- This always happens and can't be turned off
- So we need to get the right thing ready in that cube, to be projected

MANCHESTER 1824

3D Viewing: the camera analogy

	Real World	Computer Graphics
1	Arrange the scene into the desired composition	Set Modelling Transformation
2	Point the camera at the scene	Set Viewing Transformation
3	Choose the camera lens, or adjust the zoom	Set Projection Transformation
4	Determine the size and shape of the final photograph	Set Viewport Transformation

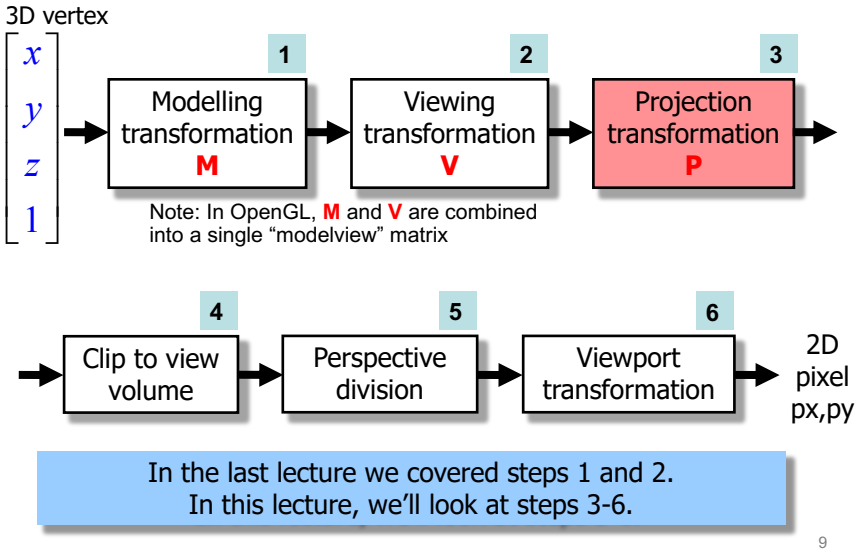
MANCHESTER 1824

The camera analogy

Step 3: Choose a camera lens (wide-angle, zoom...)

MANCHESTER 1824

The 3D Viewing Pipeline



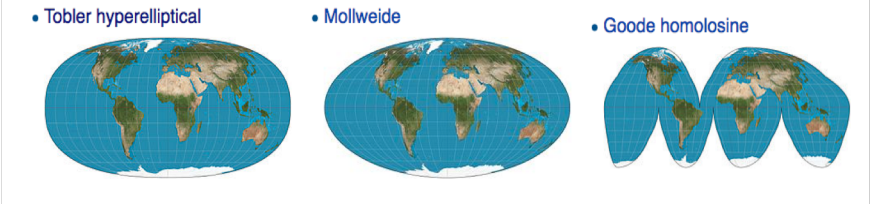
In the last lecture we covered steps 1 and 2.
In this lecture, we'll look at steps 3-6.

MANCHESTER 1824

Planar geometric projections

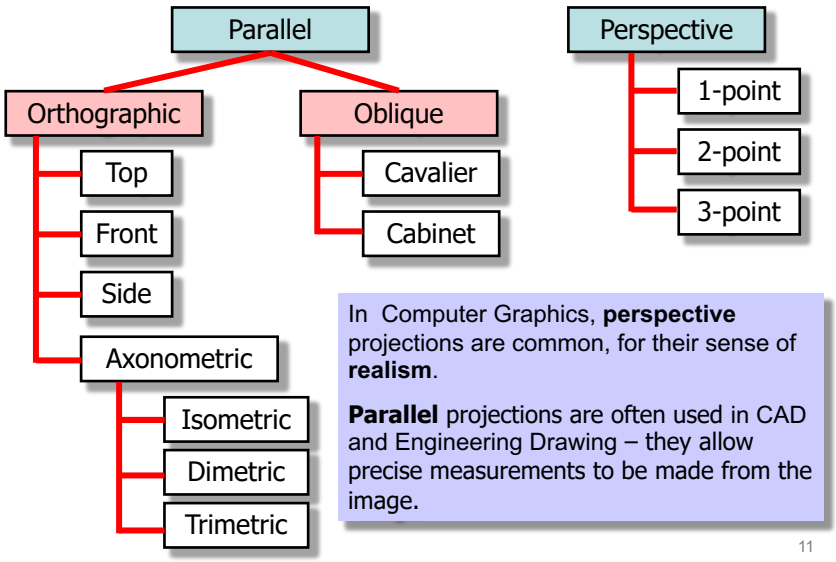
- Two classes of planar geometric projection:
  - parallel projection
  - perspective projection

- Note that we consider only **planar geometric projections**: these project lines to lines. Other kinds of projections change lines into curves and vice versa (used in cartography, for example), which we will not study. Examples:



MANCHESTER 1824

Planar Geometric Projections



In Computer Graphics, **perspective** projections are common, for their sense of **realism**.

Parallel projections are often used in CAD and Engineering Drawing – they allow precise measurements to be made from the image.

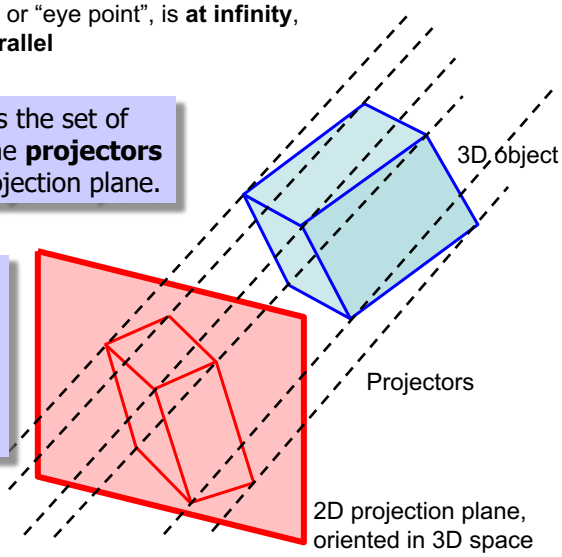
MANCHESTER 1824

Parallel projection

Centre of projection, or "eye point", is **at infinity**, so projectors are **parallel**

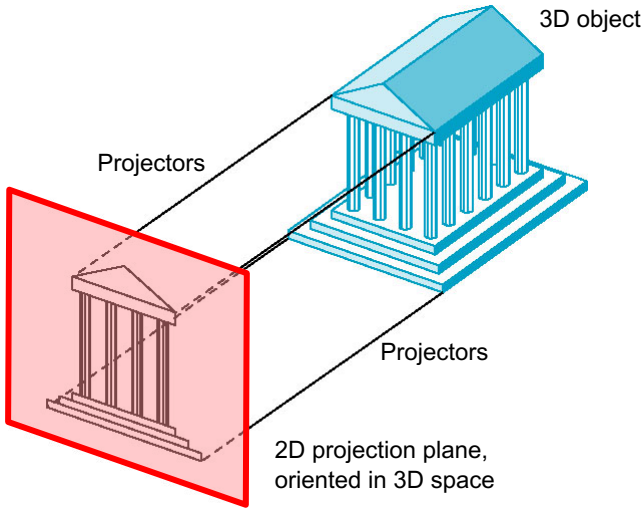
The **projection** is the set of points at which the **projectors intersect** the projection plane.

Parallel edges remain parallel in the projection. Angles between edges may be **distorted**.



MANCHESTER 1824

Parallel projection

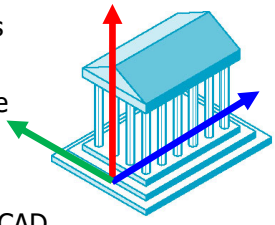


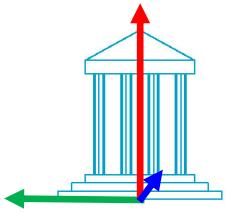
13

MANCHESTER 1824

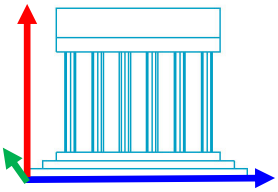
Types of parallel projection 1: orthographic

- In orthographic parallel projection, projectors are **perpendicular** to the projection plane
- The projection plane is **parallel** to **one** plane (XY, XZ, YZ) of the world
- Projected image shows only 2 of the 3 axes
- No distortion of lengths or angles, useful for CAD

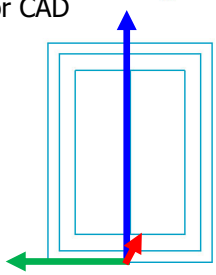




Ortho: front (YZ)

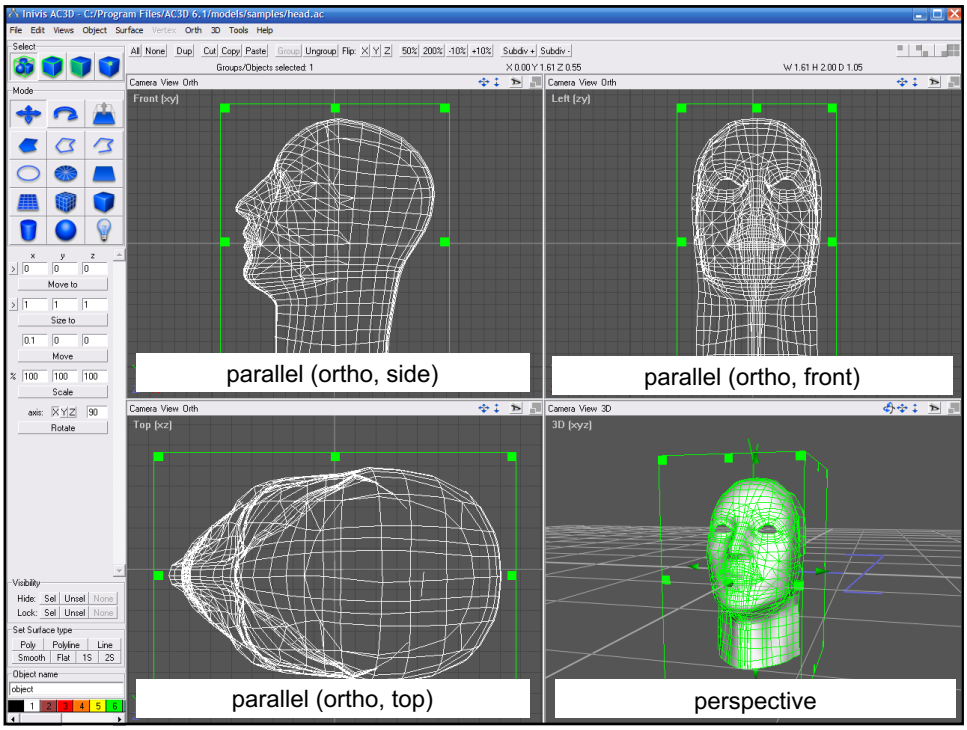


Ortho: side (XY)



Ortho: top (XZ)

14



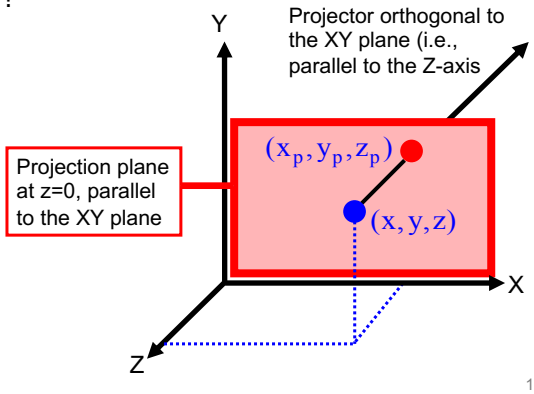
MANCHESTER 1824

Computing orthographic projection

- For a projection plane located at **z=0**, parallel to the XY plane
- A 3D point (x, y, z) will project to (x_p, y_p, z_p) on the plane

- What is (x_p, y_p, z_p)?

- By definition,
  - $x_p = x$
  - $y_p = y$
  - $z_p = 0$



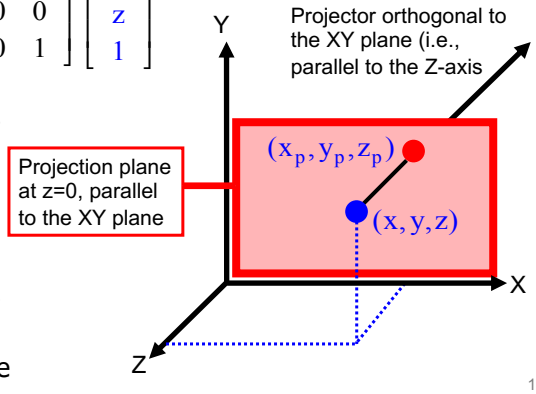
MANCHESTER 1824

Matrix for orthographic projection

- We can express the projection as the matrix

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

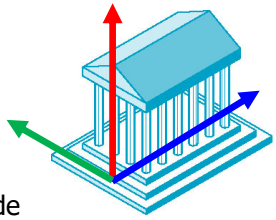
- This is the same as scaling by (1,1,0)
- Notice this matrix is **singular**
- We can derive similar matrices for other positions of the projection plane

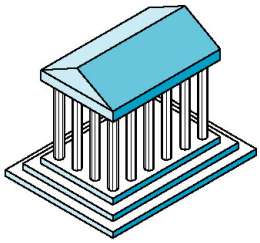


MANCHESTER 1824

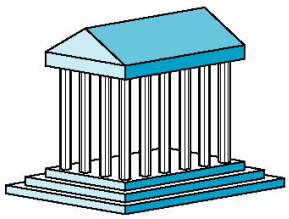
Types of parallel projection 2: axonometric

- in axonometric parallel projection, projectors are **perpendicular** to the projection plane
- the projection plane can have **any orientation**
- We can see the 3 axes. Distortion of lengths or angles, but measurements can still be made

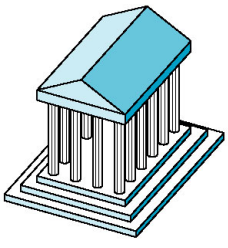




Isometric: projection plane is symmetrical to (X,Y,Z)



Dimetric: projection plane is symmetrical to 2 of (X,Y,Z)

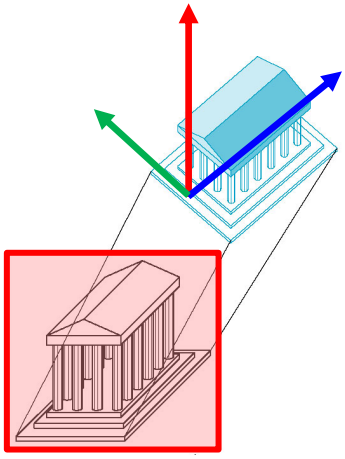


Trimetric: projection plane placed arbitrarily

MANCHESTER 1824

Types of parallel projection 3: oblique

- The most general case
- Projectors can make **any angle** with the projection plane
- The projection plane can have **any orientation**

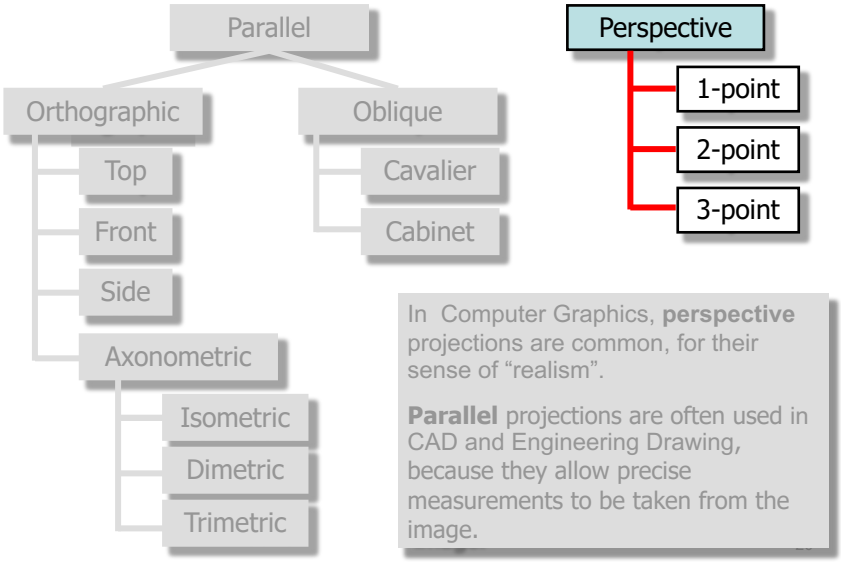


- We see the three axes. Distortion of lengths or angles, but measurements can still be made
- We can of course derive matrices to perform **all the parallel projections** we have seen (but we will not go into the details in this course, so not examinable)

19

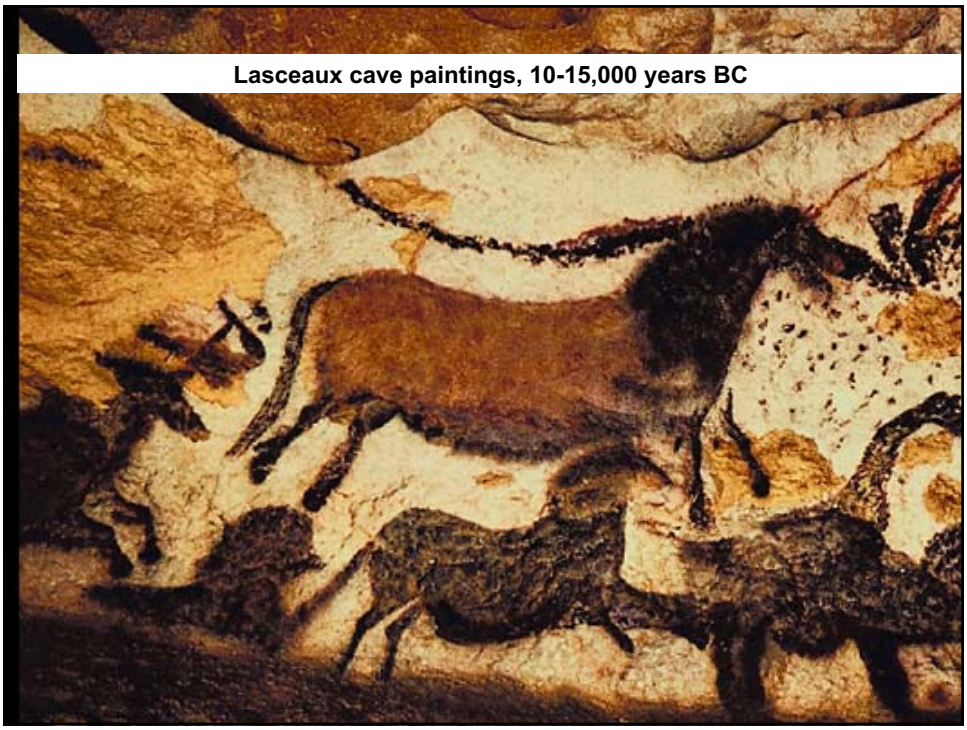
MANCHESTER 1824

Planar Geometric Projections



In Computer Graphics, **perspective** projections are common, for their sense of "realism".

Parallel projections are often used in CAD and Engineering Drawing, because they allow precise measurements to be taken from the image.



Lasceaux cave paintings, 10-15,000 years BC

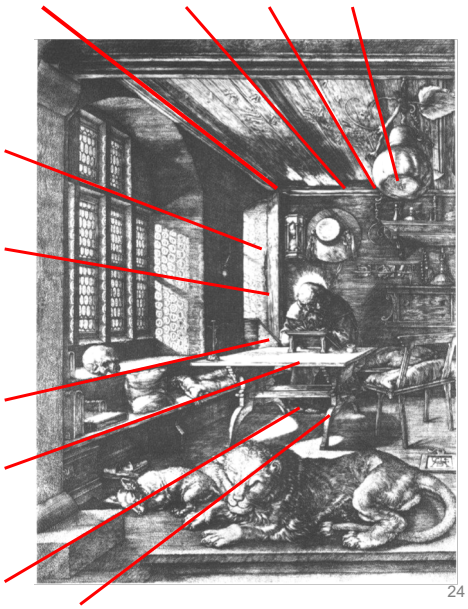
Reconstruction of the temple of Jerusalem. From William of Tyre, *Histoire d'Outremer* 1200-1300 AD

Christ Handing the Keys to St. Peter by Pietro Perugino (1481-82)

MANCHESTER 1824

Perspective

- During the Renaissance (C14-17), artists began to understand and use perspective
- Engraving by Albrecht Dürer, St Jerome dans sa Cellule (1514)

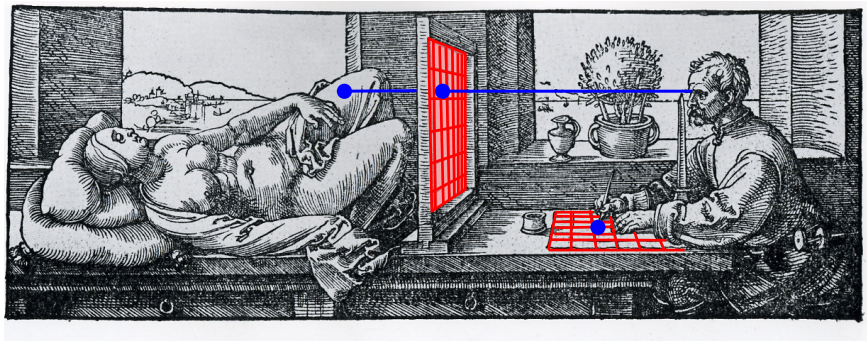


24

MANCHESTER 1824

Perspective machines

- Dürer and others used devices to help them draw with correct perspective



MANCHESTER 1824

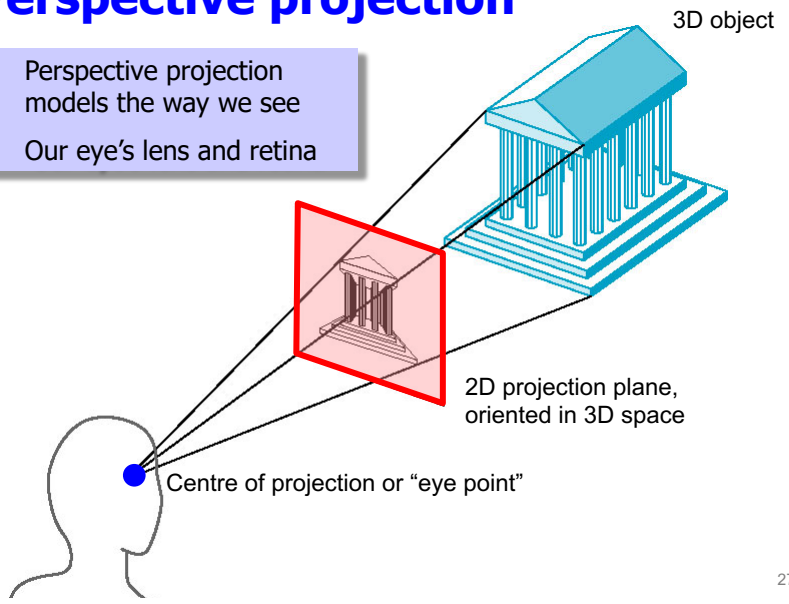
Modern perspective machines

www.morpholioapps.com

MANCHESTER 1824

Perspective projection

- Perspective projection models the way we see
- Our eye's lens and retina

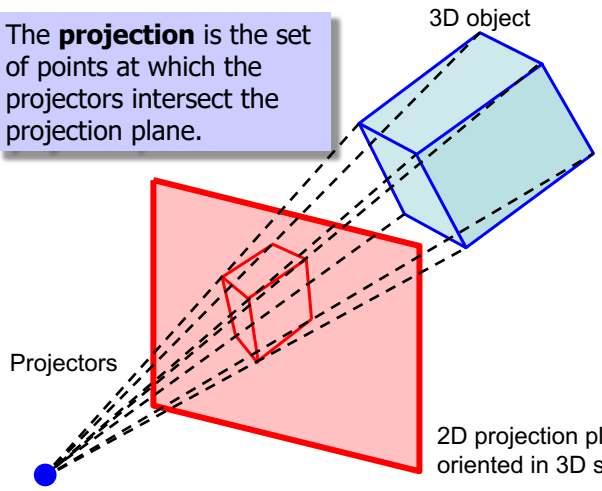


MANCHESTER 1824

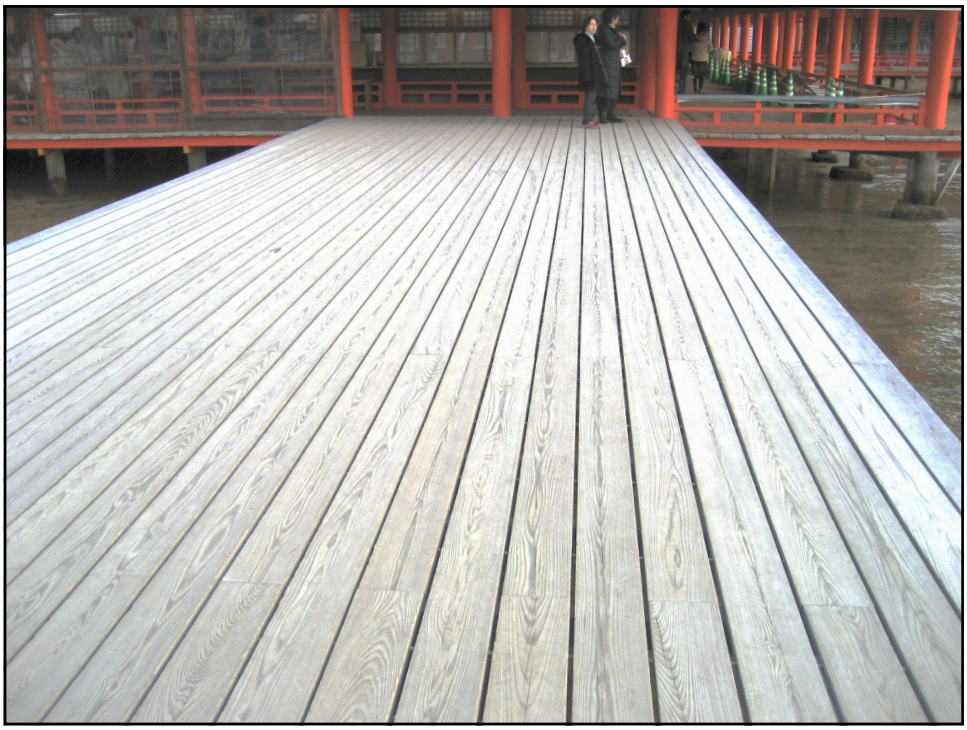
Perspective projection

Centre of projection or "eye point", is **a point**, so projectors **converge**

The **projection** is the set of points at which the projectors intersect the projection plane.

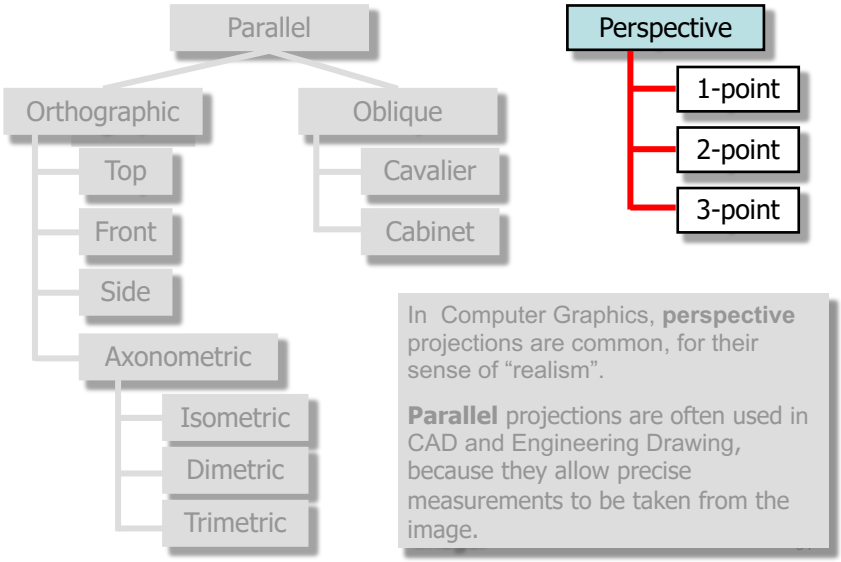


Objects further away from the COP become **smaller**. Edges that were parallel may converge. Angles between edges may be **distorted**.



MANCHESTER 1824

Planar Geometric Projections



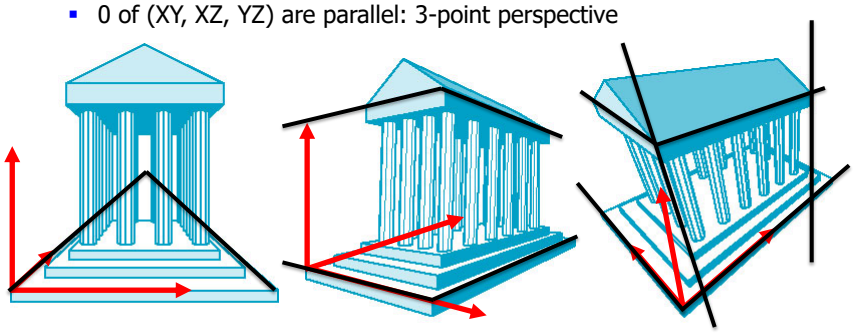
In Computer Graphics, **perspective** projections are common, for their sense of "realism".

Parallel projections are often used in CAD and Engineering Drawing, because they allow precise measurements to be taken from the image.

MANCHESTER 1824

Perspective projections

- Which (XY, XZ, YZ) planes of the world are parallel to the projection plane determines how many **vanishing points** are seen in the projected image
  - 2 of (XY, XZ, YZ) are parallel: 1-point perspective
  - 1 of (XY, XZ, YZ) are parallel: 2-point perspective
  - 0 of (XY, XZ, YZ) are parallel: 3-point perspective



1-point perspective | 2-point perspective | 3-point perspective

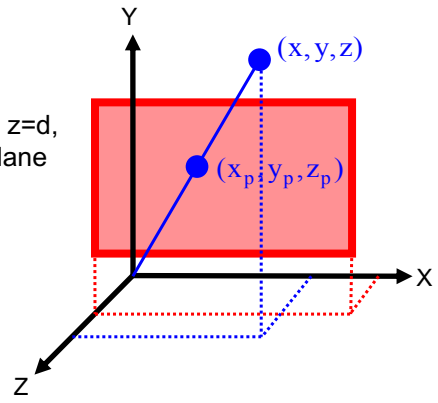
32

MANCHESTER 1824

Computing perspective

- Projection plane located at z=d, and parallel to the XY plane
- A point (x,y,z) will project to (x_p,y_p,z_p) on the projection plane
- What is (x_p,y_p,z_p)?
- By definition, $z_p = d$

Projection plane, at z=d, parallel to the XY plane



MANCHESTER 1824

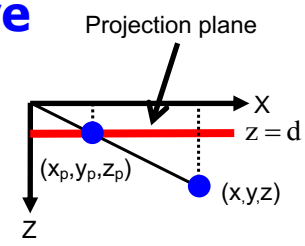
Computing perspective

- We need to find x_p and y_p
- Looking at the top view, looking down the y-axis, by similar triangles, we have:

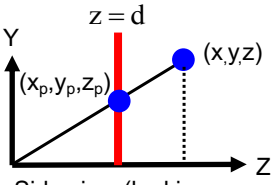
$$\frac{x_p}{d} = \frac{x}{z} \rightarrow x_p = \frac{dx}{z} \rightarrow x_p = \frac{x}{z/d}$$

- Similarly, in a side view, with our eye at the origin and looking along the x-axis, we have:

$$y_p = \frac{y}{z/d}$$



Top view (looking down Y axis)



Side view (looking down the x-axis)

MANCHESTER 1824

Perspective projection matrix

- We can express this transformation as:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

- Applying this matrix to (x,y,z,1) gives:

$$\begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

- Recall (Lecture 3 slide 25) that we must always have **w = 1**, so we divide all elements by z/d to give:

$$\begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix}$$

- This is called **perspective division**

Notice how x and y are now dependent on z

MANCHESTER 1824

Perspective matrices: summary

- We derived the matrix to perform a 1-point projection where the projection plane is parallel to the XY plane.

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d_z & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

We've renamed **d** to $\mathbf{d_z}$ just to remind us that it's a z-coordinate

- We can also derive (details omitted, so not examinable) a matrix which expresses the most general case, **3-point projection**, where the projection plane is not parallel to any of the XY, XZ or YZ planes:

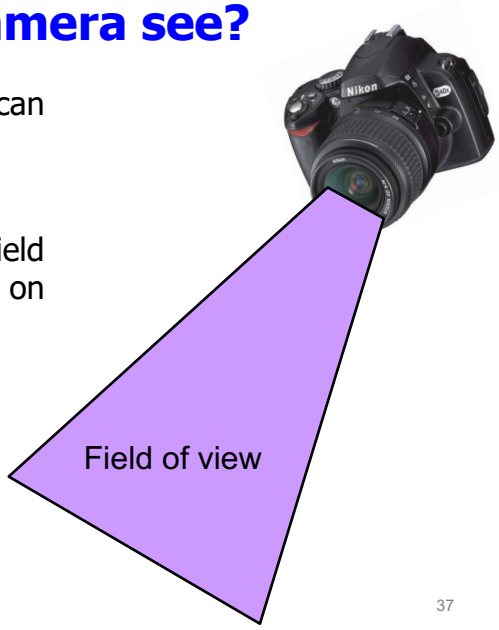
$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/d_x & 1/d_y & 1/d_z & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The projection plane intersects the X,Y and Z axes at ($\mathbf{d_x}$ $\mathbf{d_y}$ $\mathbf{d_z}$)

MANCHESTER 1824

What can a camera see?

- what part of the world can a camera see?
- objects in front of it
- only objects within its field of view, which depends on its lens (114° for the human eye)
- effectively only a finite distance
  (very distant objects will project too small to be seen)



MANCHESTER 1824

Viewing volumes and clipping

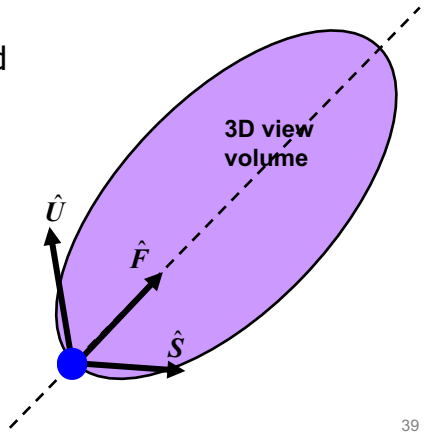
- So, having specified the position and orientation of the camera ...
- …we now describe the field of view of the camera by defining a 3D **view volume**

- Only those parts of 3D objects which lie inside the view volume are displayed – all objects are **clipped** against the view volume
- We'll look at clipping shortly, but first we'll see how the view volumes are defined

MANCHESTER 1824

Defining the view volume

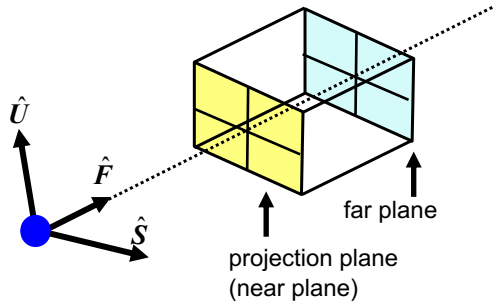
- In **Viewing (1)** we established a coordinate system for the camera – with axes $\hat{S}\,\hat{U}\,\hat{F}$
- We define a 3D **view volume** which is attached to the camera
- As we would expect, the shapes of the volumes for parallel and perspective projection are different.



MANCHESTER 1824

View volume for parallel projection

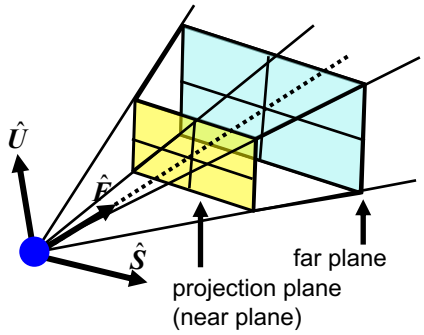
- A view volume is a 3D shape, defined by six planes
- For parallel orthographic projection, it's a **cuboid**
- The cuboid is defined by a near plane (the projection plane) and a far plane, and top/bottom, left/right planes
- The near and far planes are orthogonal to the camera's $\hat{F}$ axis



MANCHESTER 1824

View volume for perspective projection

- For perspective projection, this view volume is a **frustum**, a truncated pyramid
- The frustum is defined by a near plane (the projection plane) and a far plane
- The near and far planes are orthogonal to the camera's $\hat{F}$ axis



MANCHESTER 1824

Perspective: a problem

- A projection matrix transforms a 3D point to a 3D point with W <> 1
- For example, for a 1-point perspective projection:

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ w_p \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

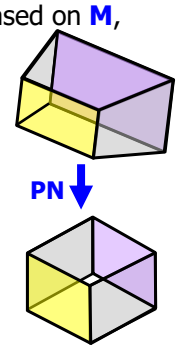
In this case $w_p = z/d$

- So we need to divide through by w, to get (x_p, y_p, z_p)
- z_p = d, which means we have **lost** the original 3D object's z-coordinates – they've all been set to d.
- This is unhelpful, because we want to **keep** the z depth information, to do hidden-surface removal.

MANCHESTER 1824

Solution: Projection normalization

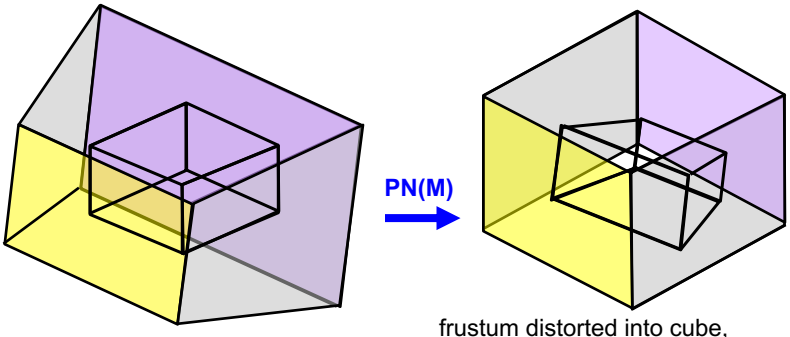
- We need a perspective transformation which preserves depth information.
- Suppose we have a particular perspective transformation expressed by frustum **F** and matrix **M**
- We can derive a new transformation matrix **PN(M)**, based on **M**, that **distorts F into a cube**
- Transforming our model by **PN(M)**, and then taking an orthographic projection…
- …produces **exactly the same result** as performing our original perspective transformation **M**, with one difference: the z depth values are preserved
- This new technique is called **projection normalization**, and OpenGL creates **PN(MP)** for us automatically.



MANCHESTER 1824

Solution: Projection normalization

- **PN(M)** distorts the perspective frustum to a unit cube
- An object inside the frustum is also distorted
- If we then apply OpenGL's default **orthographic projection view (Lecture 5 slide 14)**, we get the same view as if we had applied a perspective projection to the original undistorted object



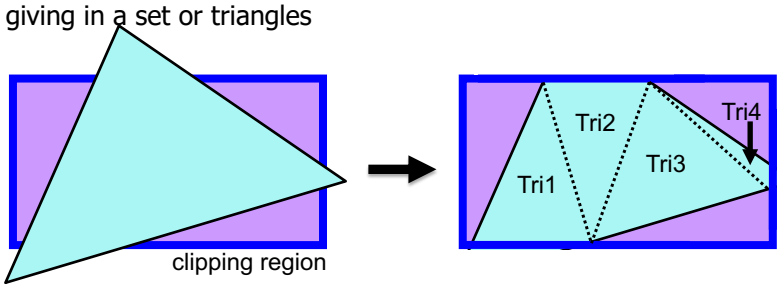
frustum and object

frustum distorted into cube, and object distorted

MANCHESTER 1824

The clipping operation

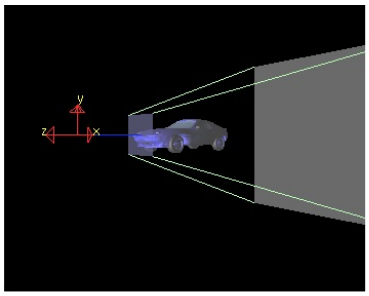
- Clipping takes place in the cube produced by projection normalization (easier than clipping against a frustum)
- Clipping is easiest to visualise in 2D
- Here is a polygon clipped against a rectangular clipping region, giving in a set or triangles

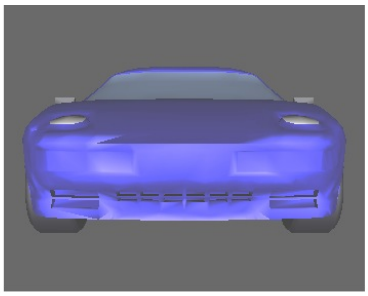


- There are efficient algorithms for clipping in 2D and 3D

MANCHESTER 1824

Viewing volume and clipping



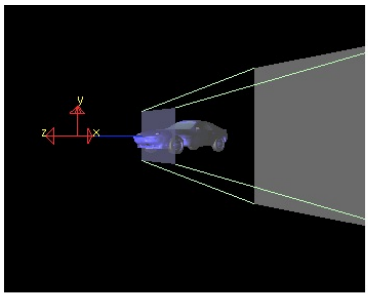


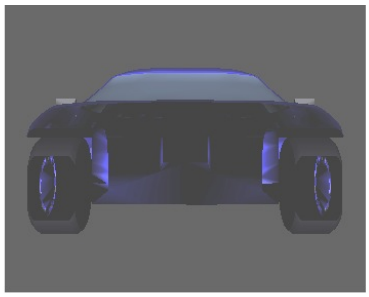
Command manipulation window

```
                fovy  aspect  zNear  zFar
gluPerspective( 36.0  , 1.00  , 1.0   , 10.0  );
    gluLookAt( 0.00   , 0.00  , 2.00  ,   <- eye
               0.00   , 0.00  , 0.00  ,   <- center
               0.00   , 1.00  , 0.00  );  <- up
Click on the arguments and move the mouse to modify values.
```

MANCHESTER 1824

Viewing volume and clipping





Command manipulation window

```
                    fovy  aspect  zNear  zFar
gluPerspective(  36.0  , 1.00  , 1.3   , 10.0  );
    gluLookAt(  0.00  , 0.00  , 2.00  ,   <- eye
                0.00  , 0.00  , 0.00  ,   <- center
                0.00  , 1.00  , 0.00  );  <- up
```

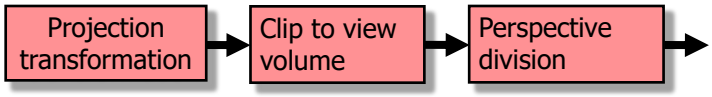
Click on the arguments and move the mouse to modify values.

7

MANCHESTER 1824

Perspective division

- The clipping operation returns a set of (x,y,z,w) vertices defining polygons which are inside the view volume
- OpenGL now performs the **perspective division by w** to convert these (x,y,z,w) values to (x,y,z) 3D points



- OpenGL now performs its default ortho projection

48

MANCHESTER 1824

3D Viewing: the camera analogy

	Real World	Computer Graphics
1	Arrange the scene into the desired composition	Set Modelling Transformation
2	Point the camera at the scene	Set Viewing Transformation
3	Choose the camera lens, or adjust the zoom	Set Projection Transformation
4	Determine the size and shape of the final photograph	Set Viewport Transformation

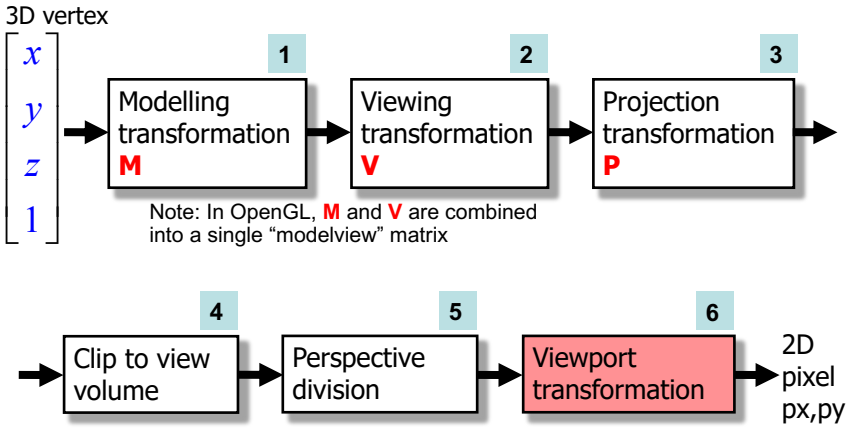
MANCHESTER 1824

The camera analogy

Step 4: Decide the size of the final photograph

MANCHESTER 1824

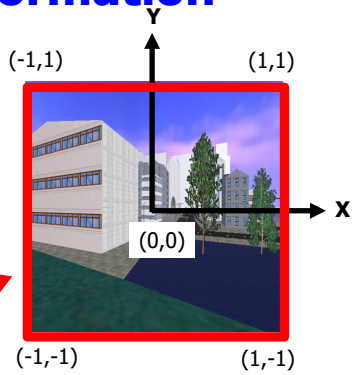
The 3D Viewing Pipeline

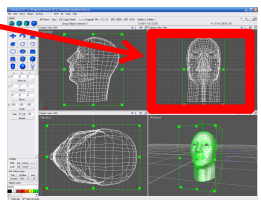


MANCHESTER 1824

The Viewport Transformation

- After perspective division, we now have 3D coordinates
- OpenGL scales these into [-1,+1] in X, Y and Z
- In the final step, using only the X and Y values, OpenGL computes a transformation from [-1,+1] to the display screen
  - Either the whole **window**
  - Or a part of it called a **viewport**
- **But we retain Z**, to remove hidden surfaces, using the z-buffer during rasterisation





MANCHESTER 1824

Summary of 3D viewing

We can now summarise the steps of the viewing process

1. The modelling transformation arranges objects in our 3D world
2. The viewing transformation transforms the world to give the same view as if it were being photographed by a camera
3. The projection transformation performs a parallel/perspective projection within limits (the clip planes)
4. Those parts of the 3D world outside the clip planes are discarded
5. If it's a perspective view, the perspective division flattens the image
6. The viewport transformation maps the final image to a position in part of the display screen window.

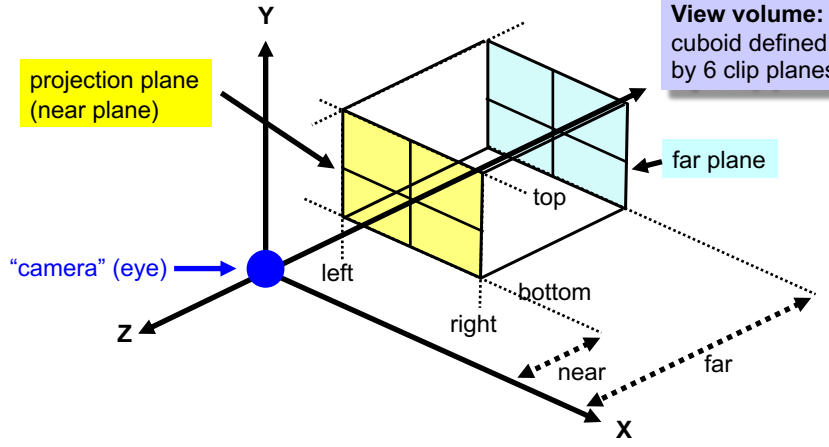
MANCHESTER 1824

Reference material on OpenGL viewing

- The following slides describe in more detail the programming steps for using projections in OpenGL
- Refer to these for your lab exercises.

MANCHESTER 1824

Orthographic projection in OpenGL



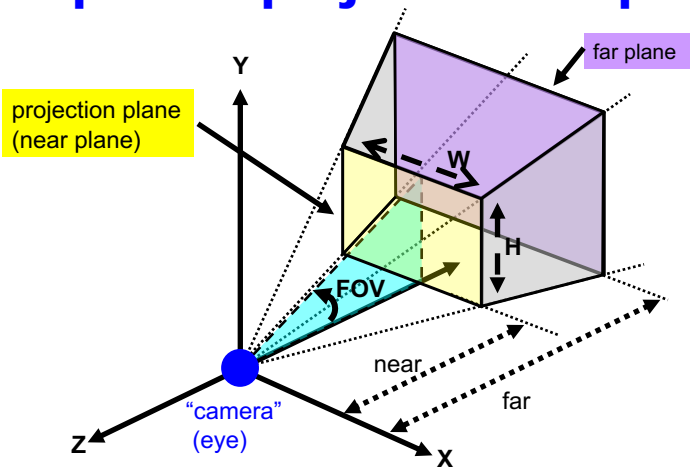

```
glOrtho (GLdouble left, GLdouble right,
         GLdouble top,  GLdouble bottom,
         GLdouble near, GLdouble far);
```

OpenGL

55

MANCHESTER 1824

Perspective projection in OpenGL



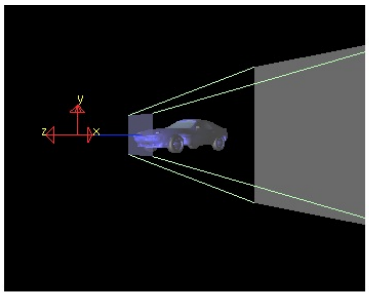

```
gluPerspective (GLdouble fovy,
                GLdouble aspect,
                GLdouble near, GLdouble far);
```

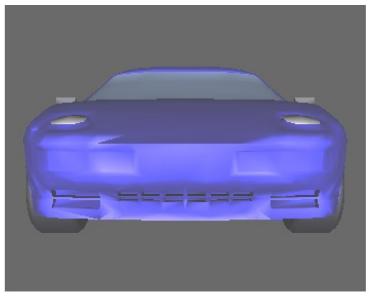
OpenGL

56

MANCHESTER 1824

Changing field of view (36°)





Command manipulation window

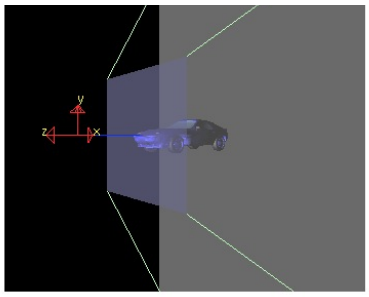
```
                fovy  aspect  zNear  zFar
gluPerspective( 36.0  , 1.00  , 1.0   , 10.0  );
    gluLookAt( 0.00   , 0.00  , 2.00  ,    <- eye
               0.00   , 0.00  , 0.00  ,    <- center
               0.00   , 1.00  , 0.00  );   <- up
```

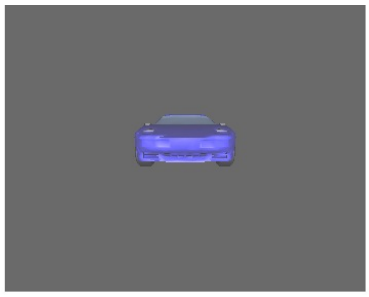
Click on the arguments and move the mouse to modify values.

7

MANCHESTER 1824

Changing field of view (92°)





Command manipulation window

```
                fovy  aspect  zNear  zFar
gluPerspective( 92.0  , 1.00  , 1.0   , 10.0  );
    gluLookAt( 0.00   , 0.00  , 2.00  ,    <- eye
               0.00   , 0.00  , 0.00  ,    <- center
               0.00   , 1.00  , 0.00  );   <- up
```

Click on the arguments and move the mouse to modify values.

8

Julian Beever
www.julianbeever.net