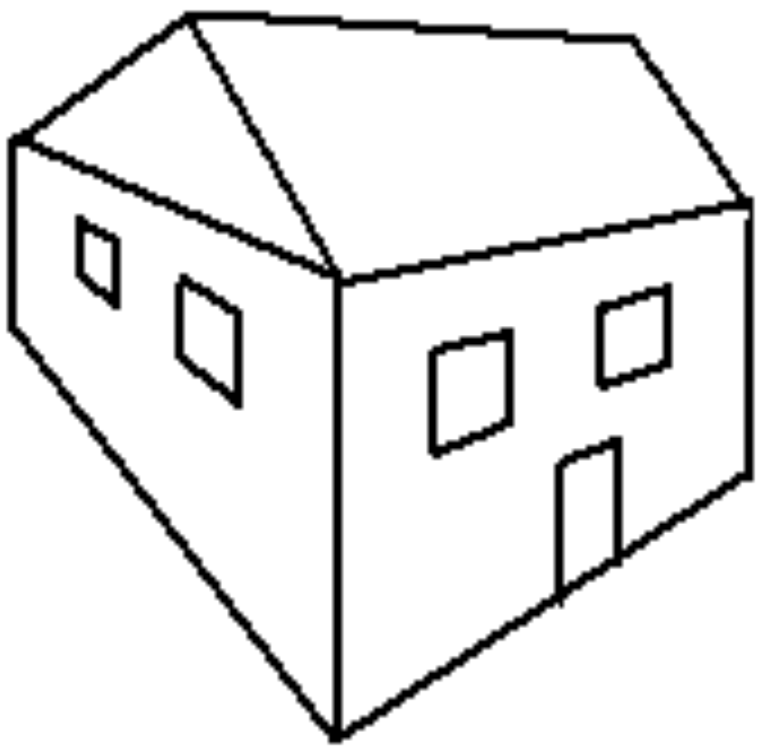
Computer Graphics

COMPH3016

Lecturer: Simon Mcloughlin

Lecture 4

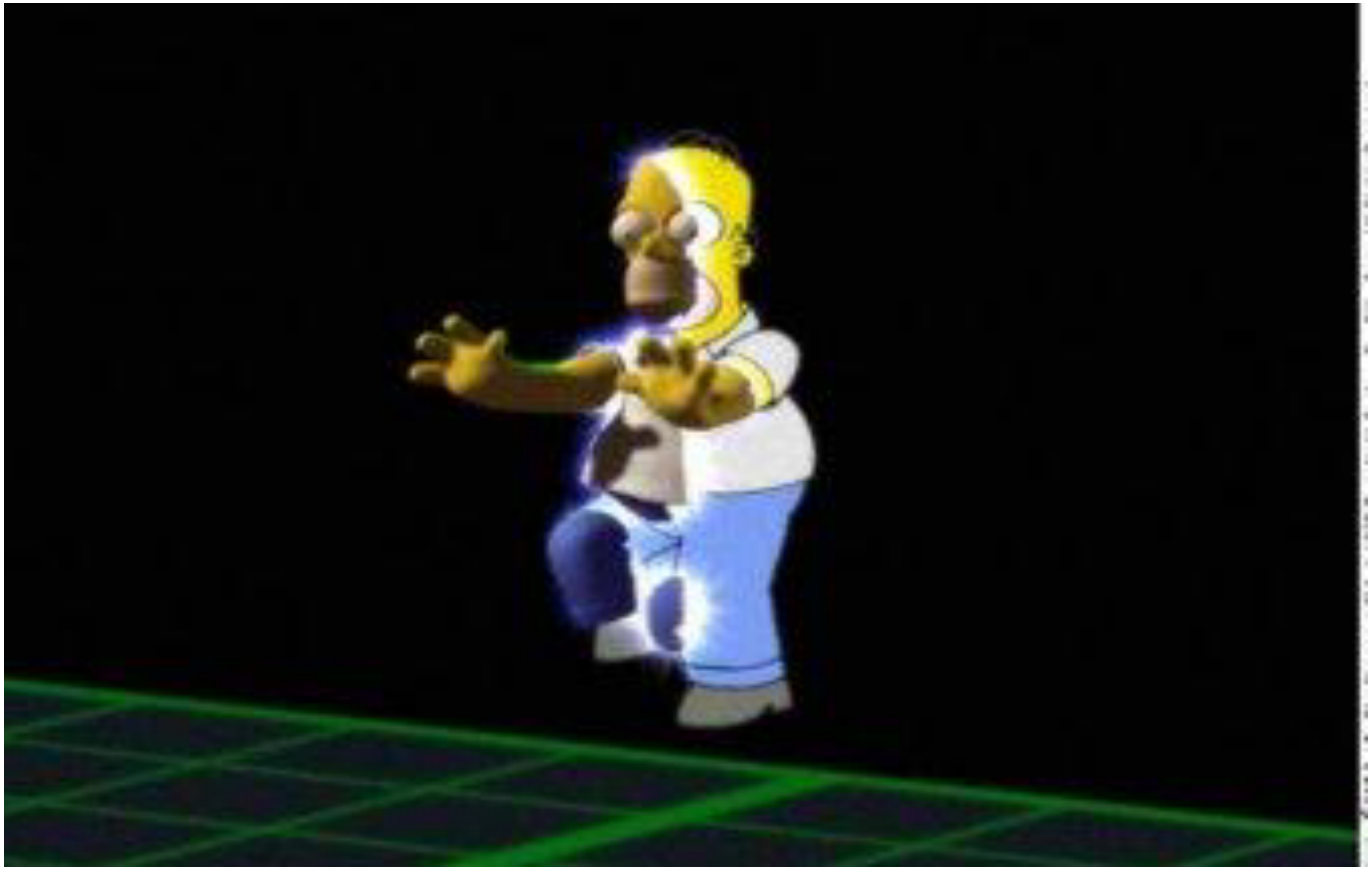


Review and Overview

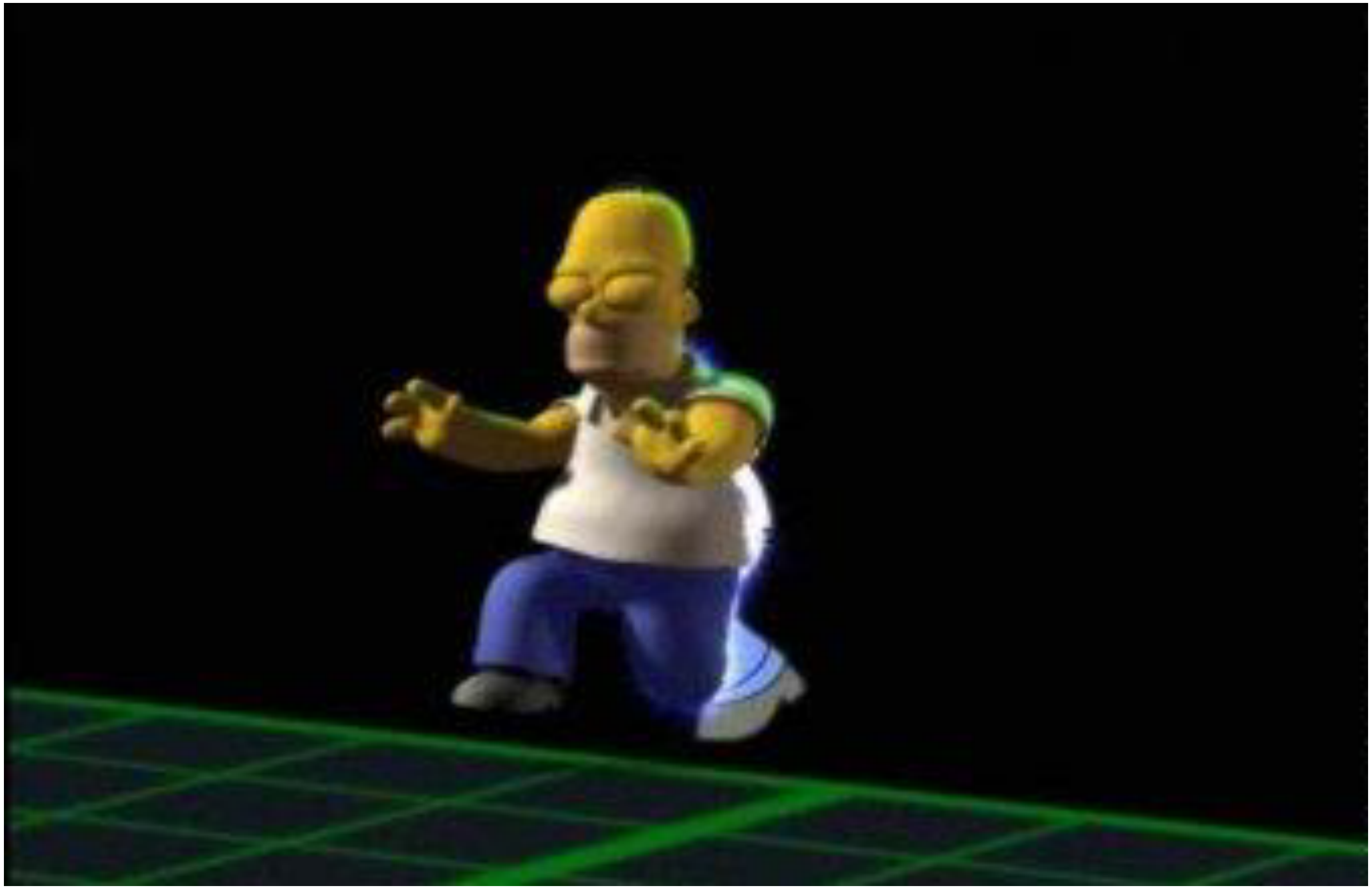
- Last week we looked at the how objects in a 2-d world coordinate system are **transformed to device coordinates** through a viewport
- We also looked at **clipping** which is the process of determining which object or object parts are inside a given window in the world coordinate system
- Most techniques and applications we have examined to date have been in the 2-d graphics realm (modeling simple output primitives, geometric transformations, viewing transformations, etc.)
- Today we are going to look at some **3-d graphics concepts**
- In particular we will see how objects defined in a **3-d world coordinate system are transformed to device coordinates**
- This is achieved through the use of **projection models**

Stepping Into the Third Dimension

Stepping Into the Third Dimension



Stepping Into the Third Dimension



Stepping Into the Third Dimension

Stepping Into the Third Dimension

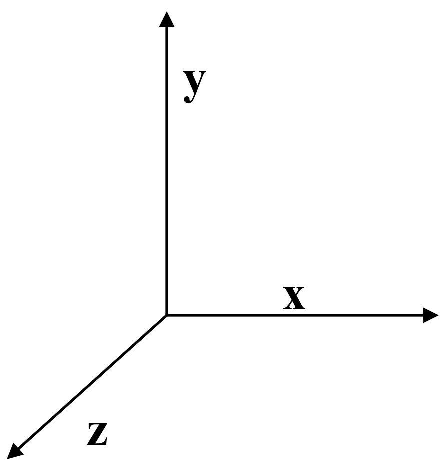
- Pictures on a screen are always 2 dimensional.
- In 2D graphics and animation we are dealing with descriptions of objects that are defined within a 2D world
  – Lines
  – Circles
  – Squares
  – 2D bitmaps/Sprites
- Surely we could also have descriptions of objects that are **defined within a 3D world?**
- In this way we could get the **software to generate images** of these objects from **any direction**, design interactive systems so that we could interact with these objects, move through virtual words, play games etc …
- You have all played Quake/Counter Strike/Halo/Max Payne etc ….
- The following are some screenshots from Max Payne …

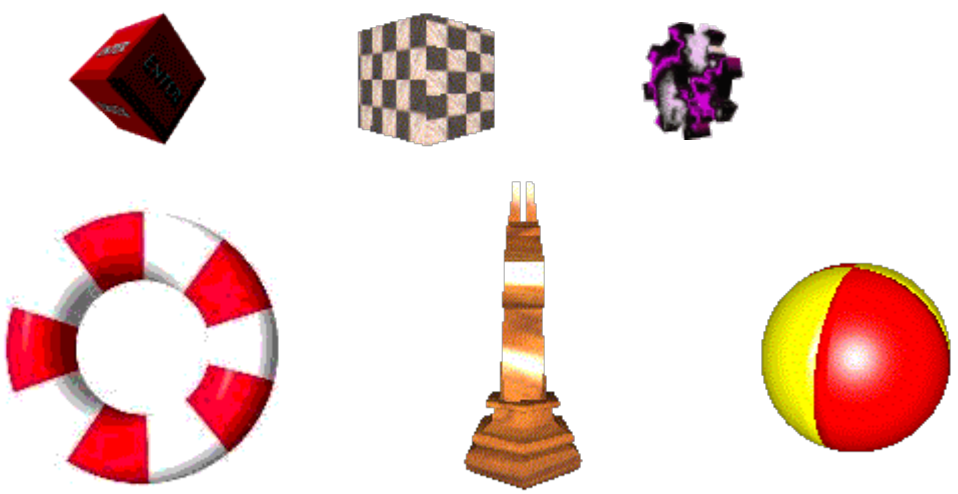

7 + 49
Pump-Action Shotgu


3 + 0
Desert Eagle

3D Descriptions

- What form would these 3D descriptions take? How are the objects represented?
- In 3D graphics all objects are stored in terms of mathematical descriptions.
- However these must be defined in terms of 3D geometry. This implies that points have an x, a y, and a z coordinate, we still have lines, but also planes, spheres, boxes and so on.

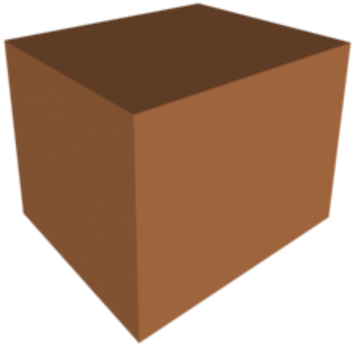




3D Descriptions

- There are certain fundamental 3D objects, usually known as primitives, which can be easily defined, stored and represented.
- Examples of primitives:

Box



What information would be needed for a description of a box?

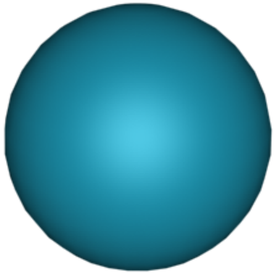
Simple, all that is needed is to store its length, width, and height …

What are these?

Just numbers ….

So it's easy …..

Sphere



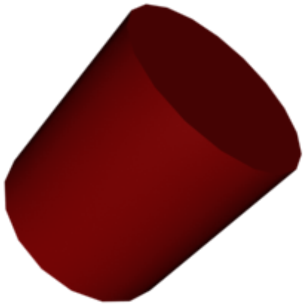
What information would be needed for a description of a sphere

Just the radius.

That's all.

3D Descriptions

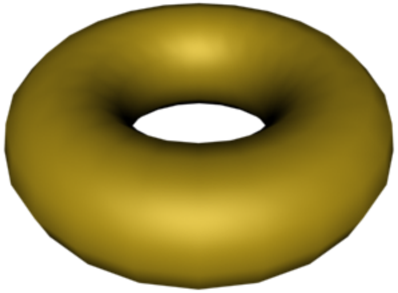
Cylinder



What information would be needed for a description of a cylinder?

The radius and the height

Torus



A torus is common CG (Computer Graphics) primitive also. To store a description of a torus we need to store two radii (radius 1 and radius 2)

Radius 1 – distance from centre point to outer edge of torus
Radius 2 – radius of ring

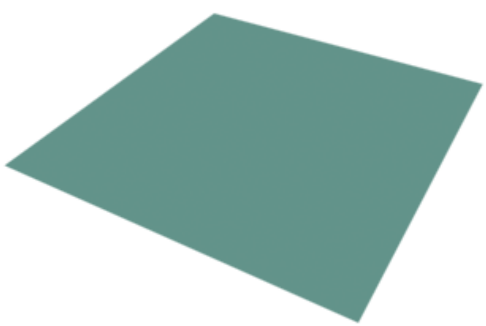
3D Descriptions

Cone

To represent a cone we will need to know 2 radii and a height.

Radius 1 – radius of bottom
Radius 2 – radius of top (in the one on the left its probably zero)
Height – obvious ….

Plane

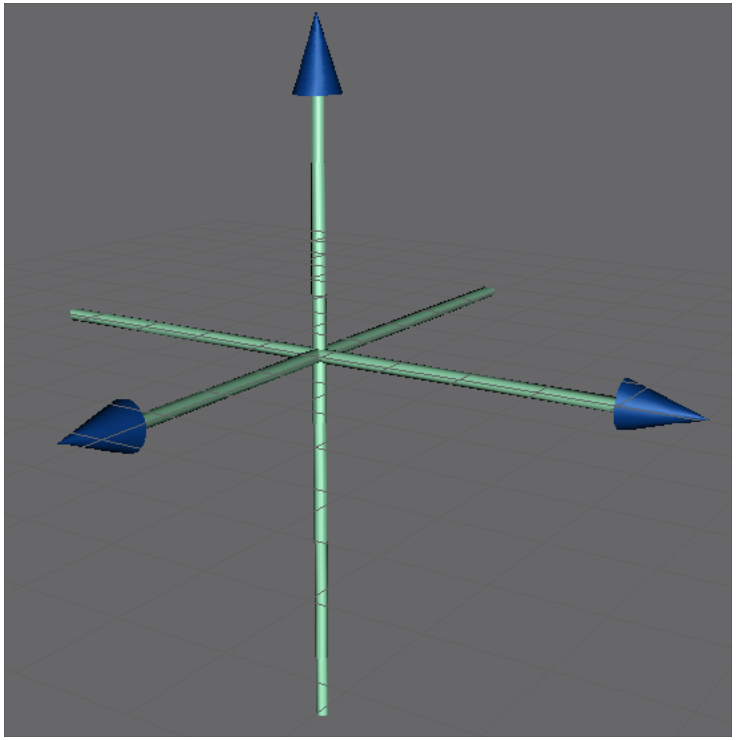


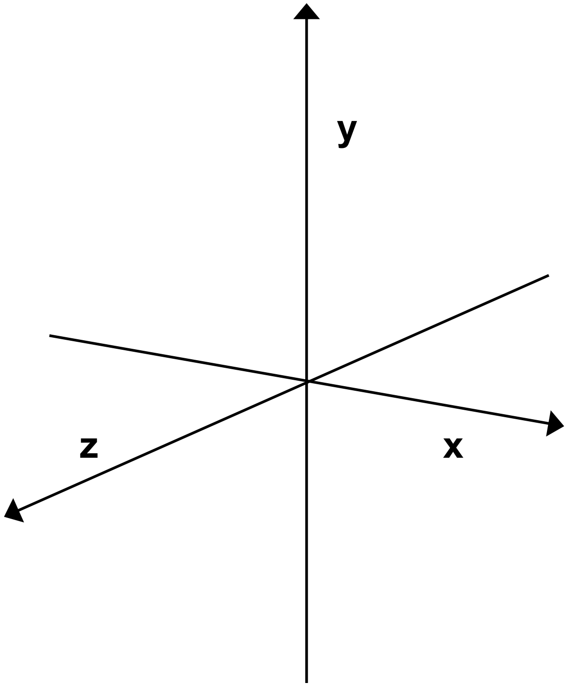
Flat surface that is infinitely thin. Doesn't exist in the real world?

To represent it we just need to store

Length
Width

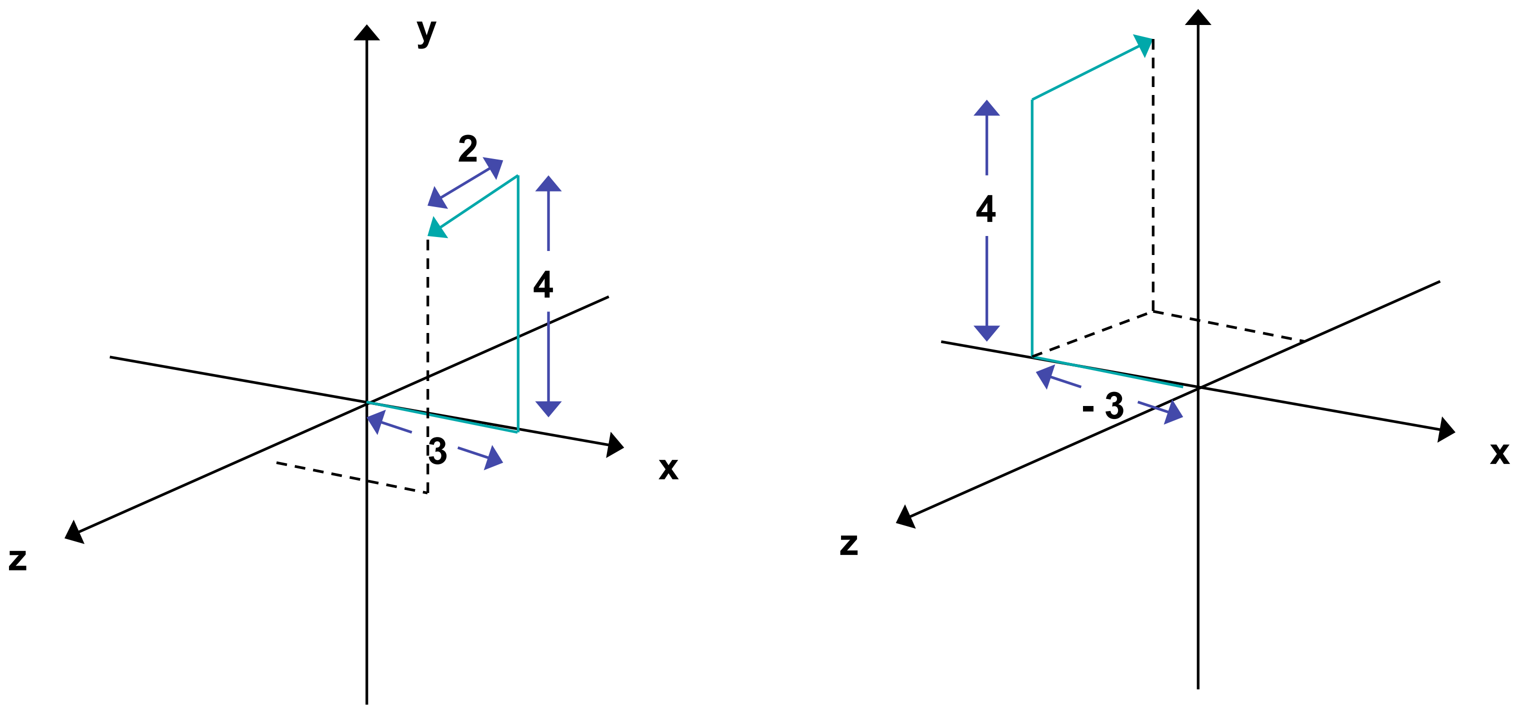
- Let's move into the third dimension.
- In order to describe points and objects in 3 dimensions we need a coordinate system with 3 axes. This means we have a *z* as well as an *x* and a *y*.





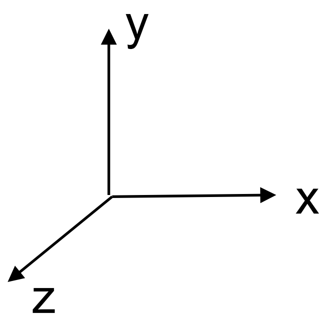
3D Points

- To describe the position of a point in such a coordinate space we must give three numbers (x,y,z).
- So for example the point (3,4,2) and (-3,4,-4) would be positioned as follows:

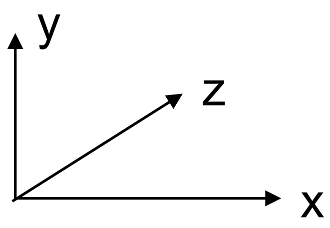


3D Coordinate Systems – Left Handed versus Right Handed

- What's the difference between a left handed and right handed coordinate system? The direction of the x,y,z axis with respect to each other
- OpenGL – Right Handed coordinate system



- DirectX – Left Handed coordinate system



3D Transforms

- Moving things, rotating things, squashing things, scaling things …
- We’ve seen these in 2D, let’s try ‘em out in 3D.
- Well firstly we have *translation*. This is just the same as its 2D equivalent except we have a z-translation amount too.

$$x' = x + tx$$

$$y' = y + ty$$

$$z' = z + tx$$

- As before (x,y,z) is the point we are translating and (tx,ty,tz) are the translation amounts

- And in matrix form using homogenous coordinates:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Second one is *scaling* . This is just the same as its 2D equivalent except we have a z-scaling amount too.

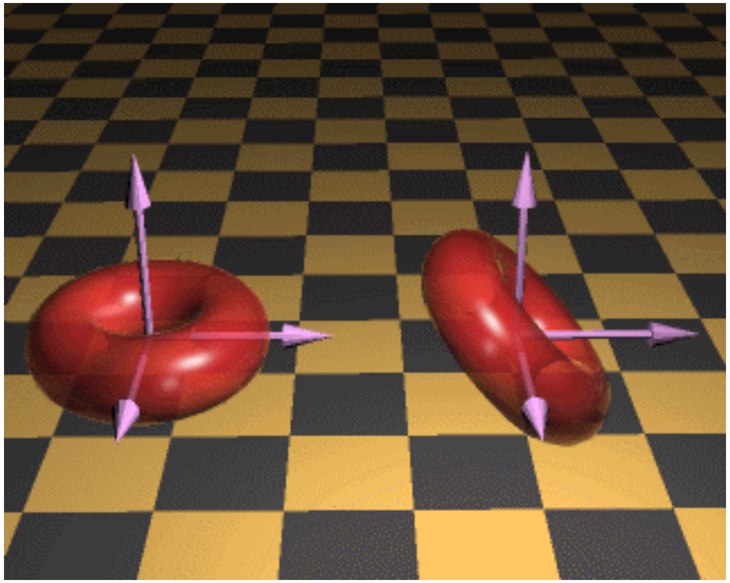
$$x' = x.sx$$
$$y' = y.sy$$
$$z' = z.sz$$

- As before (x,y,z) is the point we are scaling and (sx,sy,sz) are the scaling amounts

- And in matrix form using homogenous coordinates:

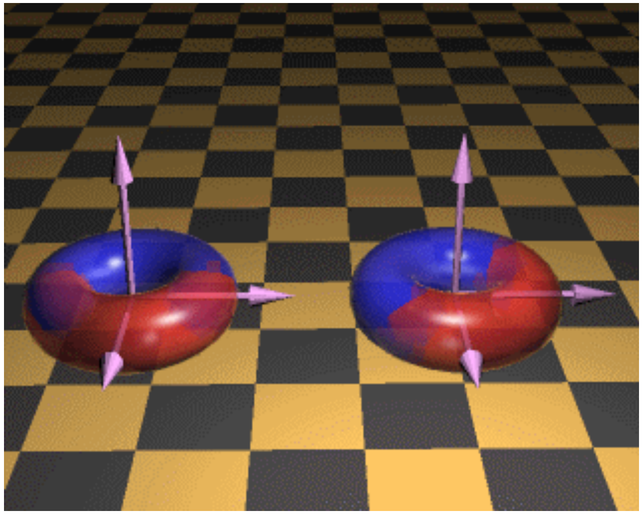
$$Ts = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- We now move on to *rotation*. This gets a little bit trickier in 3 dimensions.
- In 3D we can't just specify a rotation about a point. Why not?
- We have to rotate about an axis in 3D space.
- We define three rotations. Rotation about the x axis, rotation about the y axis, and rotation about the z axis.

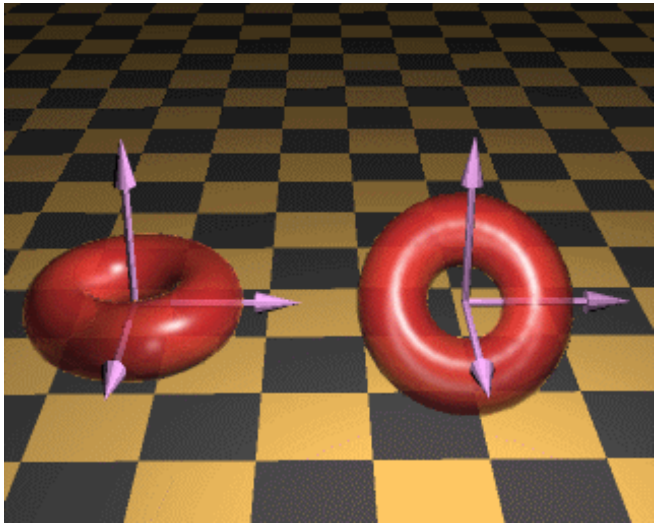


rotation about the z axis

3D Transforms



rotation about y axis



rotation about x axis

- Formulas for these rotations are as follows:

X Axis Rotation

$$x' = x$$
$$y' = y\cos\theta - z\sin\theta$$
$$z' = y\sin\theta + z\cos\theta$$

Y Axis Rotation

$$x' = x\cos\theta + z\sin\theta$$
$$y' = y$$
$$z' = x(-\sin\theta) + z\cos\theta$$

Z Axis Rotation

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

$$TR_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation about x-axis in matrix form using homogenous coordinates

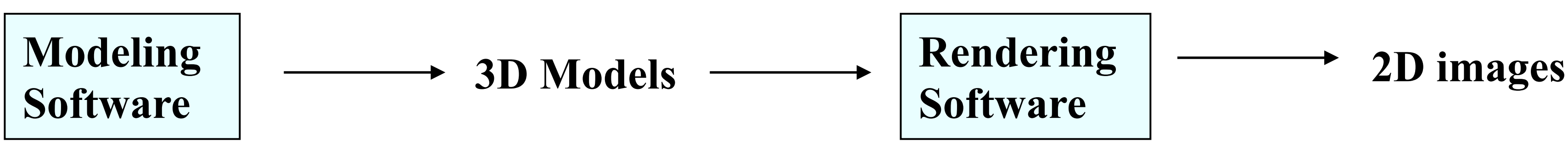
- What are the other rotation matrices?

The Rendering Pipeline

- So we can create a 3D scene using these models and store its description in some recognized format.
- How do we generate images from this?
- Generating a realistic 2D image from a 3D description is called **rendering** and the steps involved in doing so are referred to as the **rendering pipeline**.
- Sometimes one piece of software does the modeling and another does the rendering … sometimes the same one does both.

The Rendering Pipeline

- So we can create a 3D scene using these models and store its description in some recognized format.
- How do we generate images from this?
- Generating a realistic 2D image from a 3D description is called **rendering** and the steps involved in doing so are referred to as the **rendering pipeline**.
- Sometimes one piece of software does the modeling and another does the rendering … sometimes the same one does both.



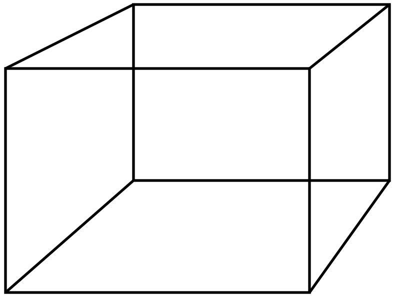
The Rendering Pipeline

- So how does rendering work?
- Need to specify a **viewpoint**, **view direction** and so on.
- Then the software has to compute a **2 dimensional projection** as seen from that viewpoint that can be displayed on the screen.
- The software has to figure out **which objects are visible** from the current viewpoint.
- Has to do this in a perspectively correct manner.
- Also has to **consider lighting** e.g. where are the light sources? how bright are they? What colour are they? What colour are the objects? Where are the shadows? Are there textures?
- And if it is an interactive application like a game it has to do all of this in real time!

Rendering

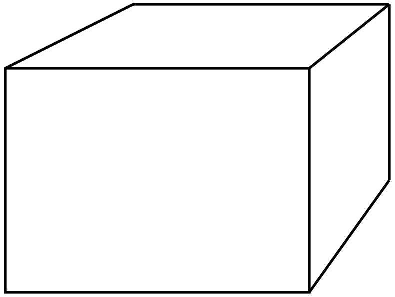
- So 3D objects exist inside the computer but we have to generate 2D images of them.
- This has to be done rapidly.
- Two things:
  - image has to be **perspectively correct** (e.g. objects which are further away appear smaller, **Dougal**).
  - the effects of light on the surfaces should be simulated in order to provide something which is vaguely **visually realistic**.
- Mathematics of perspective projection have been understood for centuries so it is not hard to write a program which takes a model and generates a wire-frame view of it.

Rendering



What's wrong with this?

- Sense of depth is present but …
  - we can see through it
  - not realistic

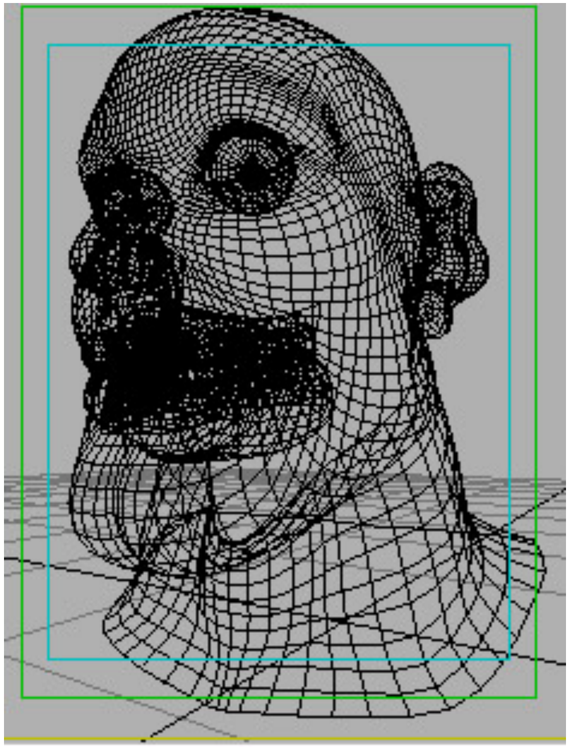


This is better. Why? What's been done here?

Rendering

- Removing the hidden lines removes all geometric ambiguity also and makes the objects appear solid.
- Next step is to **colour in the polygons**.
- There are a variety of lighting models which can be employed to do this.
- These commonly rely on a technique known as the **Phong model** which computes a **colour/illumination value for a point** on an object's surface based on its surface characteristics (colour, reflectivity) and distance from, and orientation with respect to, the various light sources in the scene.
- i.e. if a point is close to a light it will be brighter ..
- The slide overleaf shows a polygonal head model that has gone through this process …

Rendering



Head model with hidden surfaces removed.

Head model with lighting applied.

Three dimensional graphics concepts

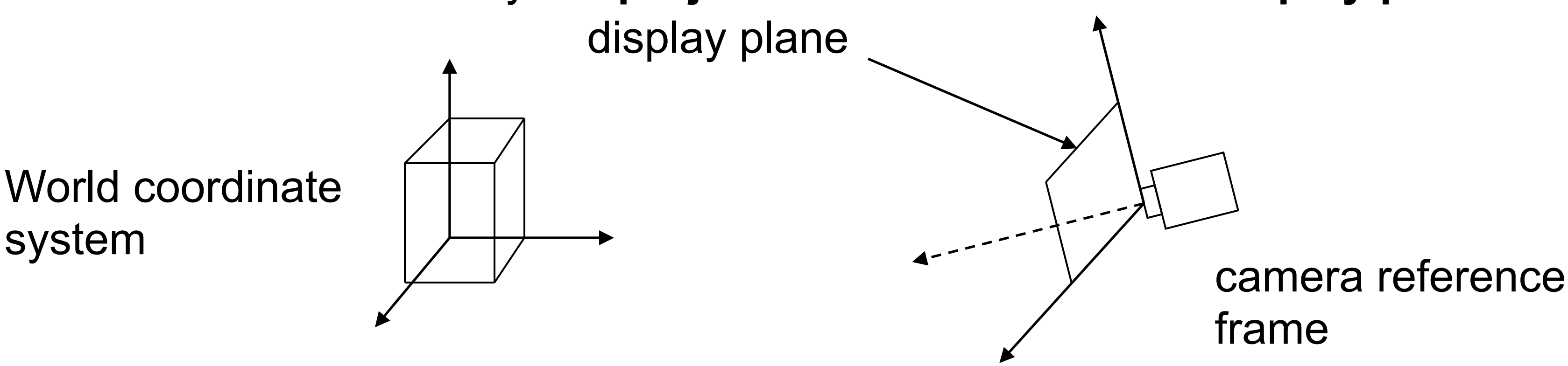
- To create a display of objects modeled in a three dimensional world coordinate system we must first **set up a coordinate reference frame for a 'virtual camera'.** This is called the **viewing reference** frame

- The viewing reference frame defines the position and orientation of the **'camera film'**

- The camera film is simply a plane onto which 3-d viewing coordinates are projected

- The procedure is similar to taking a photograph of the world around us, that is, 3-d objects are projected onto a 2-d plane

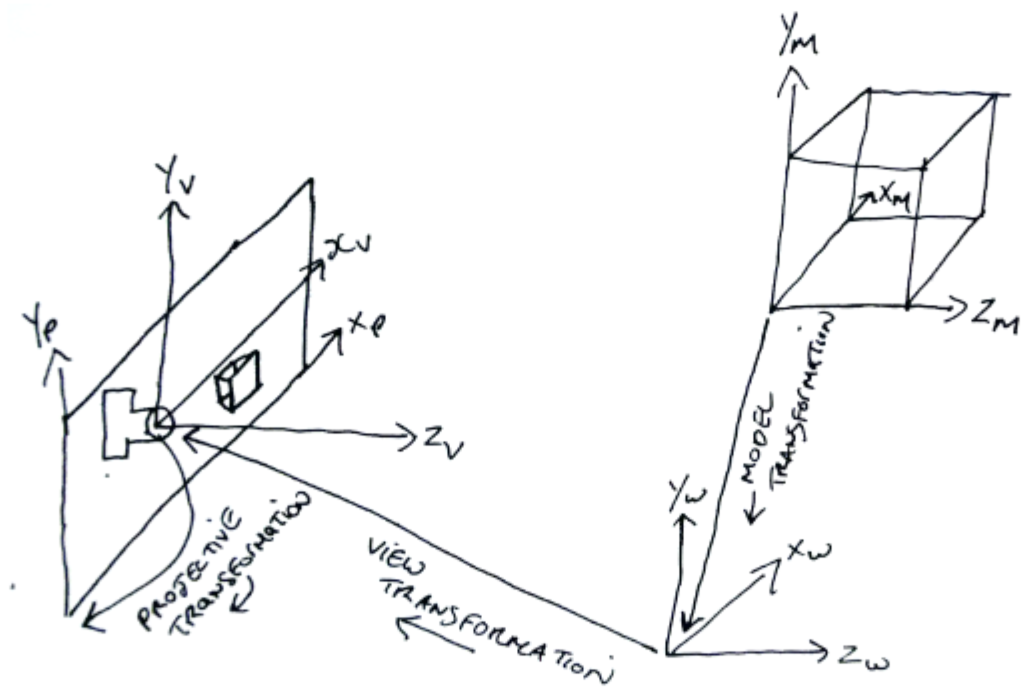
- Objects are transformed **from world coordinates to viewing frame coordinates** and then they are **projected onto the selected display plane**



OpenGL rendering pipeline

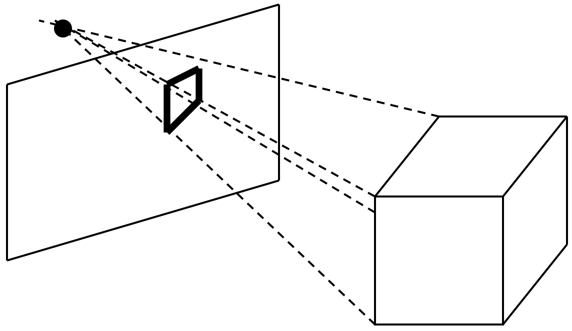
- First the object is created in its own coordinate system, the model coordinate system
- When it is introduced to openGL it will be coincident with the world coordinate system but it may be transformed – the relationship between the model and world coordinate system are maintained in the model matrix
- Before the object is projected it must be transformed into the viewing coordinate system, maintained by the view matrix
- The it can it can projected

  using the projection matrix



Perspective projection

- This type of projection creates a view of a three dimensional scene by **projecting points onto the display plane along converging paths**
- The **projection lines (projectors) all intersect at a common point called the projection reference point** or the center of projection
- Under perspective projections objects that are **further away from the display plane are displayed smaller** than objects of the same size nearer the display plane
- Parallel lines in the scene that are **not parallel** to the display plane are **projected into converging lines** (therefore perspective projections are not affine transformations)
- Images formed using perspective projections appear realistic since this is the way our eyes and cameras form images



Perspective projection - world coordinates to viewing coordinates

- We will see in a few moments that when points are defined in the viewing reference frame **simple equations can be derived that determine where three dimensional coordinates will be projected to**
- To define points in this reference frame we must perform **some transformations to the world coordinate representation**
- **Conversion from world to viewing coordinates** is equivalent to a **transformation that superimposes the viewing reference frame on the world reference frame** using translations and rotations
- Remember the equivalent transformation in 2-d?

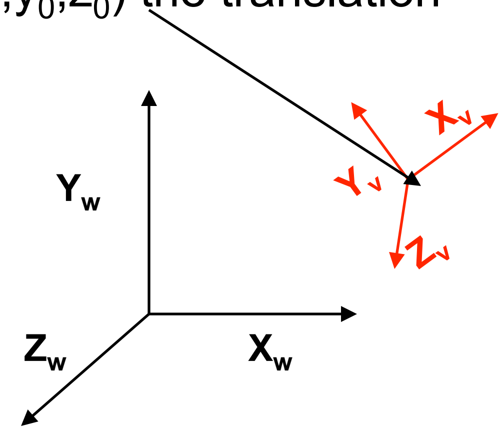
Perspective projection - world coordinates to camera reference frame

- This transformation sequence is as follows:

1. Translate the origin of the viewing reference frame (view reference point) **to the world coordinate origin**

If the view reference point is at world coordinates (x_0,y_0,z_0) the translation matrix is simply:

$$\mathbf{T} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & -x_0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & -y_0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & -z_0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 \end{bmatrix}$$



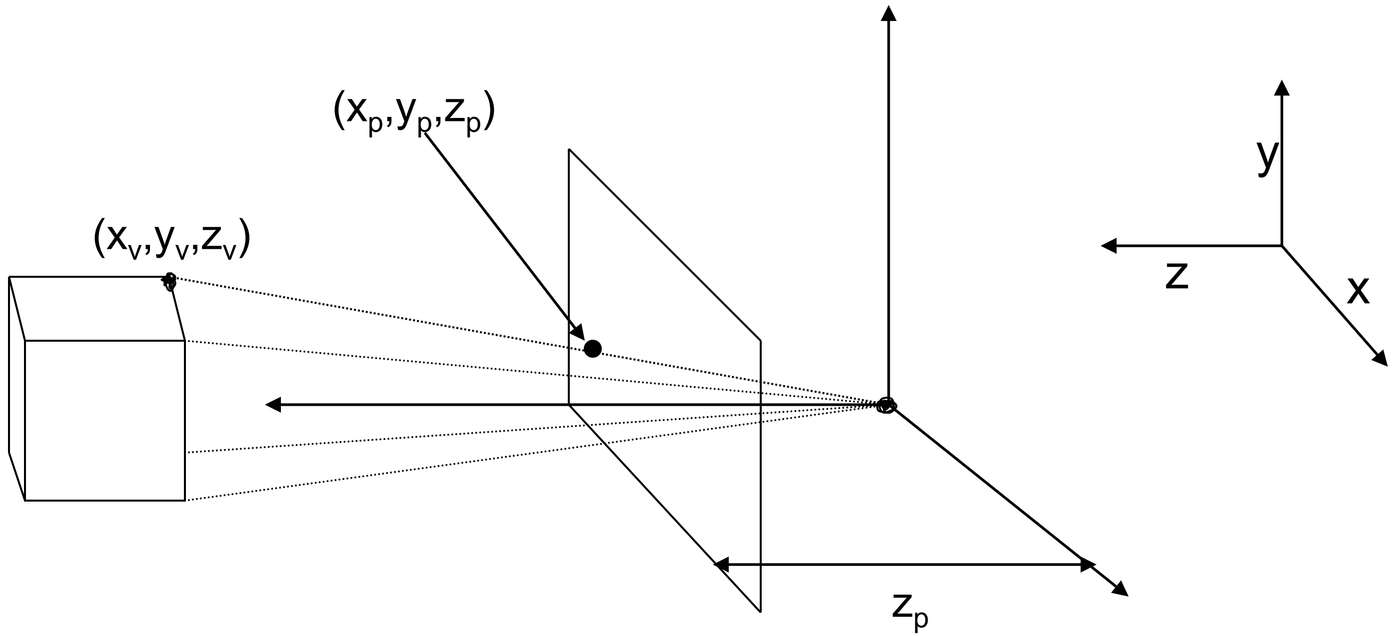
2. Apply rotations to **align the x_v, y_v, z_v** axes with the world coordinate **x_w, y_w, z_w axes**

- **Rotations in three dimensions can be performed about any spatial axis in 3-space** whereas **rotations in two dimensions are performed about the axis perpendicular to the x-y plane**

- So when converting from world to viewing coordinates in 3-d **we may have to do three rotations to get the correct alignment**

Perspective projection - projecting viewing coordinates to display plane

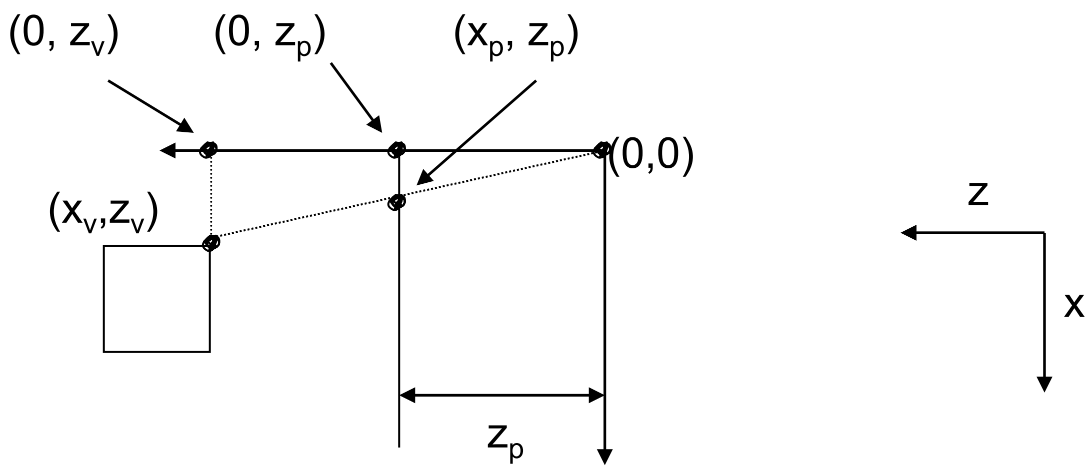
- Now we are in a position to do the projection
- Consider the following diagram



- It shows how the vertices of a cube are projected onto a plane a distance z_p from the viewing origin

Perspective projection - projecting viewing coordinates to display plane

- Considering only the point (x_v,y_v,z_v) and looking along the y axis

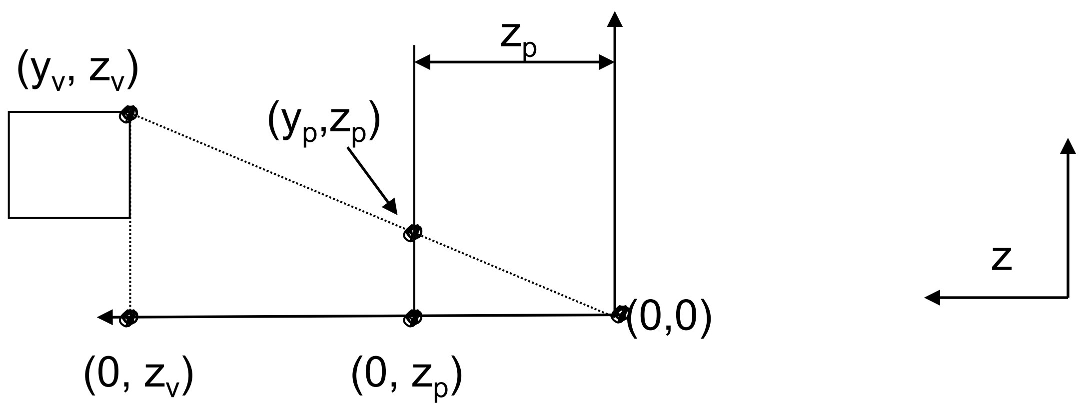


- The triangle $(0,0)$, (x_v,z_v), $(0, z_v)$ is similar (ratio of sides are the same) to the triangle $(0,0)$, (x_p,z_p), $(0, z_p)$ so,

$$\frac{x_p}{z_p} = \frac{x_v}{z_v}$$

Perspective projection - projecting viewing coordinates to display plane

- Considering only the point (x_v,y_v,z_v) and looking along the x axis



- The triangle $(0,0)$, (y_v,z_v), $(0, z_v)$ is similar (ratio of sides are the same) to the triangle $(0,0)$, (y_p,z_p), $(0, z_p)$ so

$$\frac{y_p}{z_p} = \frac{y_v}{z_v}$$

Perspective projection - projecting viewing coordinates to display plane

- Rearranging our projection equations gives

$$x_p = \frac{z_p x_v}{z_v} \qquad y_p = \frac{z_p y_v}{z_v}$$

- **Once we know the three dimensional coordinates of the point we want to project** and the **displacement of the view/display plane z_p** from the projection reference point (the origin in previous example) we can **compute where that point will project on the display plane**

Perspective projection matrix

- These equations can be represented in matrix form using homogenous coordinates as

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/z_p & 0 \end{bmatrix} \cdot \begin{bmatrix} x_v \\ y_v \\ z_v \\ 1 \end{bmatrix}$$

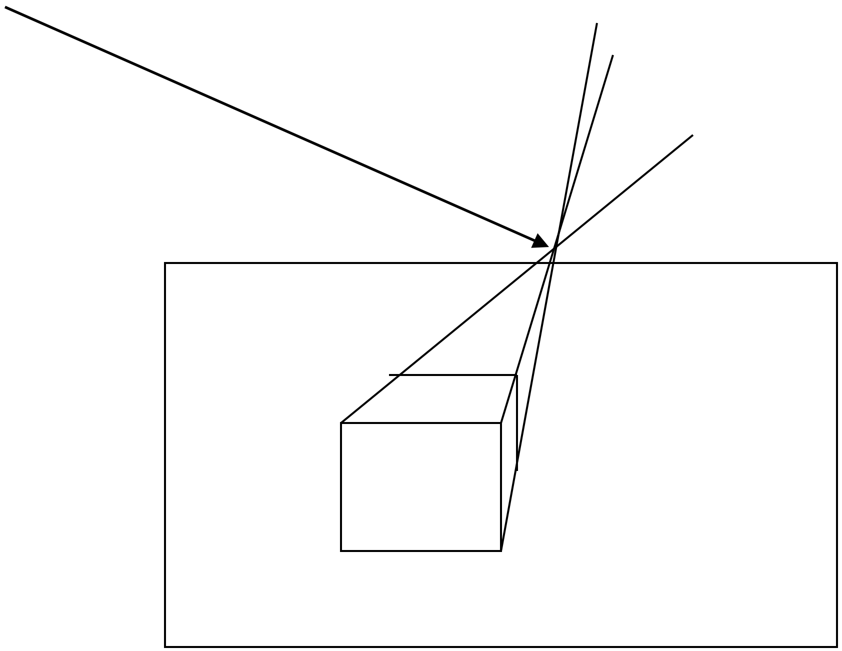
- This is called the **perspective projection matrix P**, and is used excessively in computer graphics and computer vision (this is a special case when the projection reference point is at the origin)
- x_p and y_p can be found in the usual manner using homogenous coordinates

$$x_p = \frac{x_h}{h} \qquad y_p = \frac{y_h}{h}$$

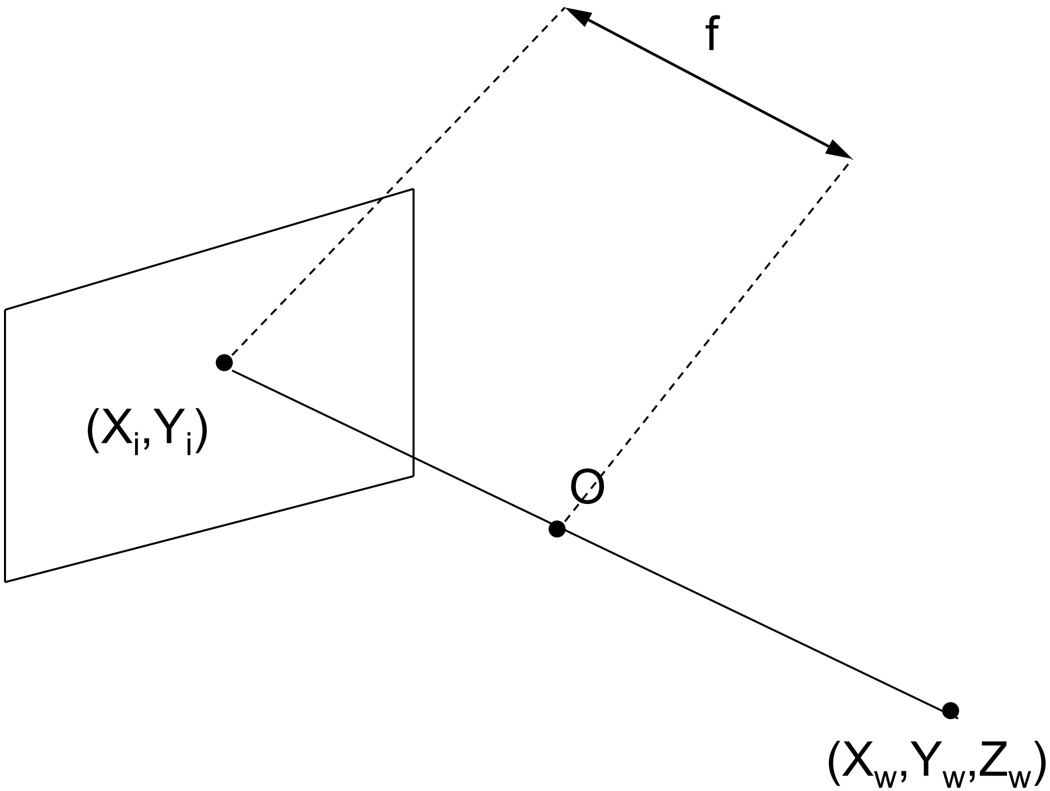
Perspective projection - vanishing points

- As was said already that any parallel lines in the viewing reference frame not parallel to the display plane are **projected into converging lines**

- The point at which a set of projected parallel lines appear to converge is called a **vanishing point**



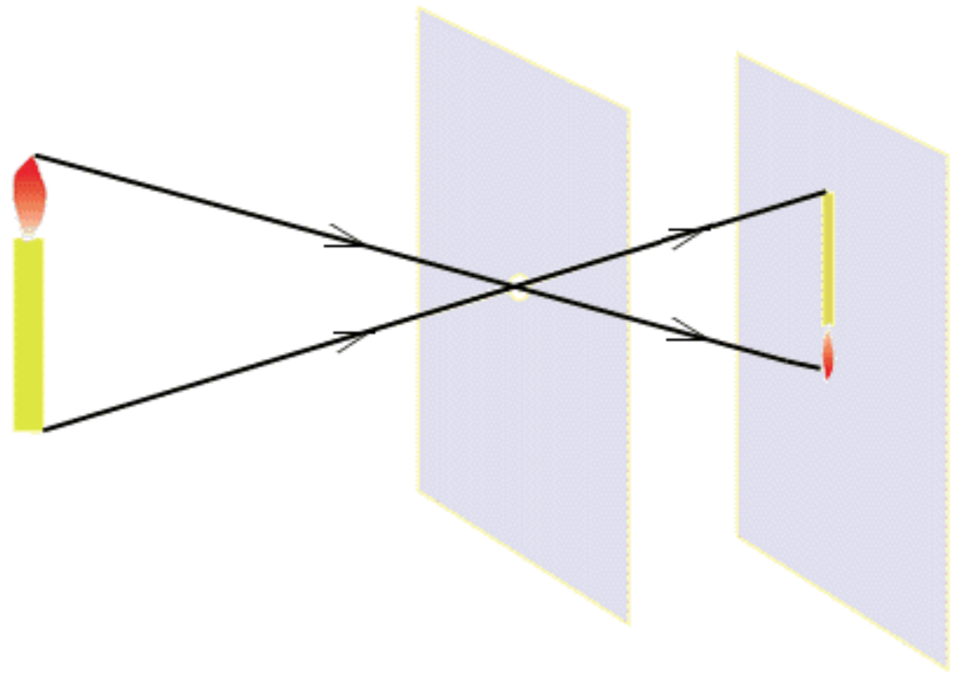
Perspective projection - relationship with Pinhole Camera



$$X_i = fX_w/Z_w$$

$$Y_i = fY_w/Z_w$$

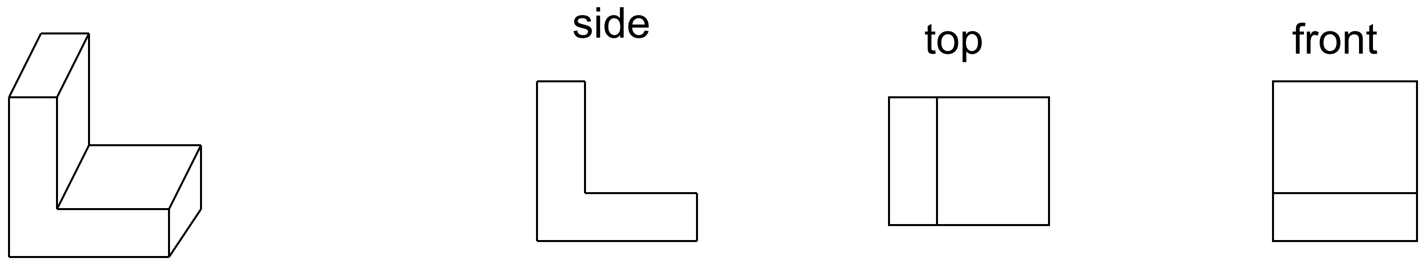
The Pinhole Camera



- Simplest imaging model!

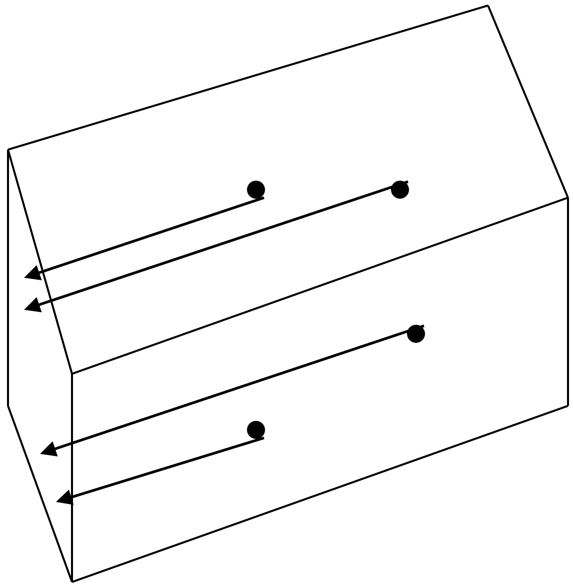
Perspective projection - a special case

- A special case of the perspective projection is when **the projection reference point is at infinity**
- This type of projection is called an **orthographic projection or a parallel projection**
- Object points are **projected along parallel lines onto the display plane**
- **Different two dimensional views** of an object can be recovered by **selecting different orientations and positions** for the view/display plane
- **Parallel lines project to parallel lines under this projection**
- Used in engineering and architectural drawings that require different views of an object whilst maintaining relative proportions



Orthographic projections

- The frustum view volume becomes a parallelepiped in orthographic projections



- shapes of the objects are **preserved** under this projections
- To find where a point projects to under an orthographic projection **draw a line from the point to the display plane**, **the line must be orthogonal to the display plane**
- **If you cannot draw an orthogonal line from the display plane to the point it lies outside the view volume**

Orthographic projection matrix

- The orthographic projection matrix is very simple

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_v \\ y_v \\ z_v \\ 1 \end{bmatrix}$$