Introduction to OpenGL
“Viewing”

Reading: Angel Ch.4

OpenGL API for Viewing

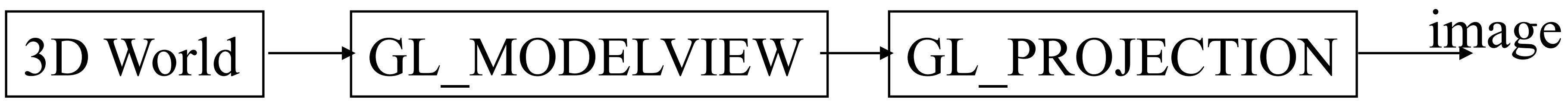
2 Important State Matricies:

GL_MODELVIEW - Transform from world to camera frame

GL_PROJECTION - Projection to image plane coordinates
- defines view frustum

Set the current state using:

glMatrixMode(*) - all subsequent commands relate to the "*" matrix mode



Camera Frame

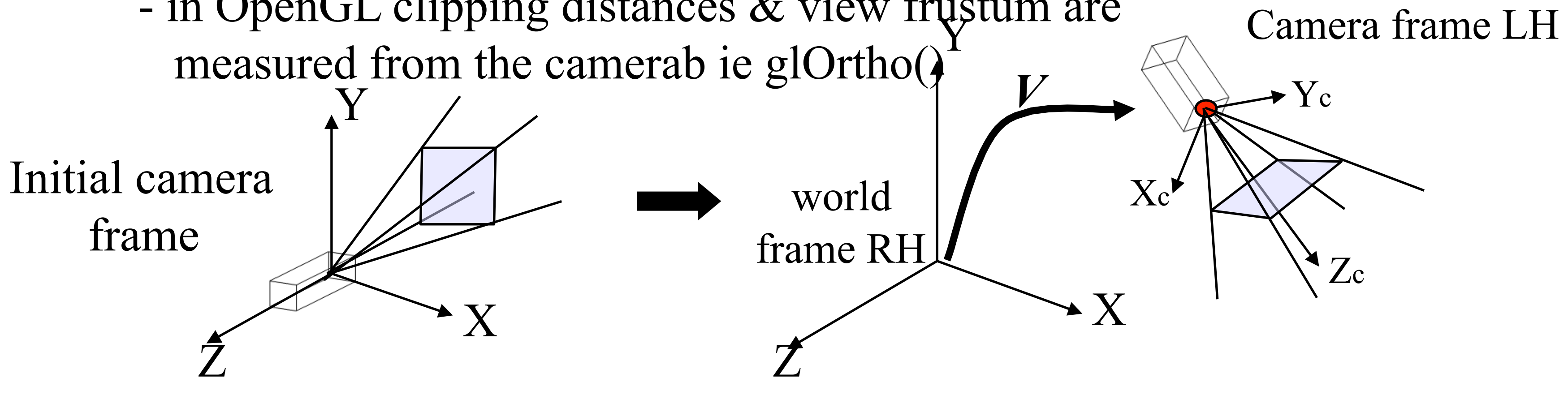
The initial 'default' camera frame is centred at the origin with
the view direction aligned with the negative Z axis of the world frame

Transformation (translation/rotation) are applied to move the camera frame
relative to the world frame "classical viewing"

Distances are measured relative from the viewer to the object
(rather than in physically based systems where the object is moved relative to
the
camera & distance is relative to the object)

Classical viewing results in a left-handed camera frame

- the mirror of the world frame [X,Y,Z] camera frame [X,Y,-Z]
- in OpenGL clipping distances & view frustum are
  measured from the camerab ie glOrtho()



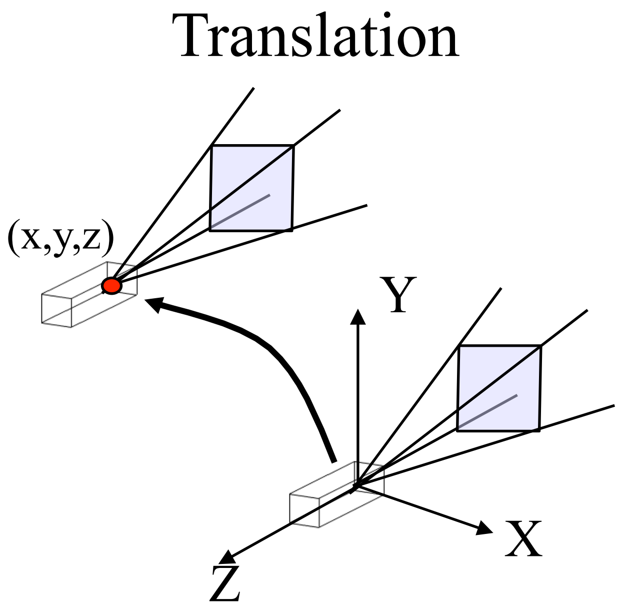
Camera Positioning

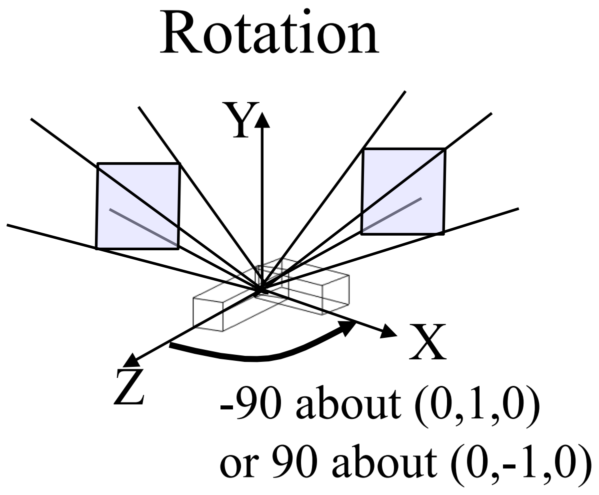
Use GL_MODELVIEW to transform camera to an arbitrary position & orientation (relative to world frame)

2 functions provided:

glTranslatef(-x,-y,-z) - translates camera position to (x,y,z) in world frame

glRotatef(-a, n_x, n_y, n_z) - rotation about axis (n_x, n_y, n_z) by angle a





Example: Camera Positioning

To position the camera at (0,-10,0) with view-direction (-1,0,0) specified in the world frame

(1) Rotate about y-axis 90 degrees to obtain the correct view-direction
(2) Translate by -10 along the camera z-axis to obtain the correct position

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0,0.0,-10);
glRotatef(-90.0,0.0,1.0,0.0);
```

Remember: The operation specified last is performed first

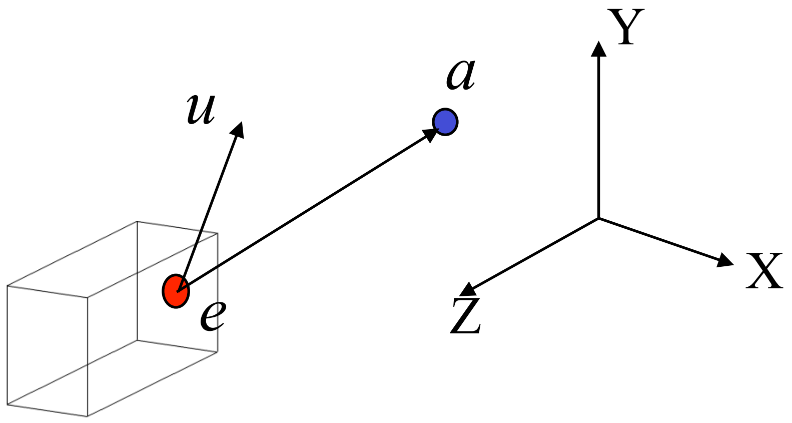
$$x_c = Vx_w = TRx_w$$

GLUT Look at Function

GLUT provides a utility function for easy camera positioning

Specify: camera position *e* + point to lookat *a* + camera up direction *u*

```
gluLookAt(ex,ey,ez,ax,ay,az,ux,uy,uz);
```



The equivalent view plane normal *n=a-e*
from this we derive the 4x4 camera view matrix *V*

Projections in OpenGL

To specify the projection we define the view frustum
2 methods

- direct setting of projection matrix
- specify the view frustum for orthographic or perspective using gl functions

Orthographic projection:

```
glOrtho(xmin,xmax,ymin,ymax,znear,zfar);
```

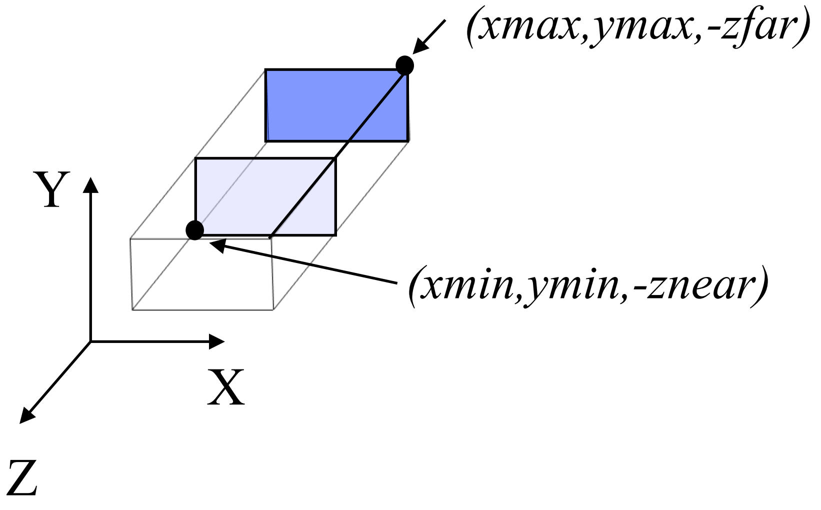
Perspective projection:

```
glFrustum(xmin,xmax,ymin,ymax,znear,zfar);

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

Orthographic view frustum

```
glOrtho(xmin,xmax,ymin,ymax,znear,zfar);
```

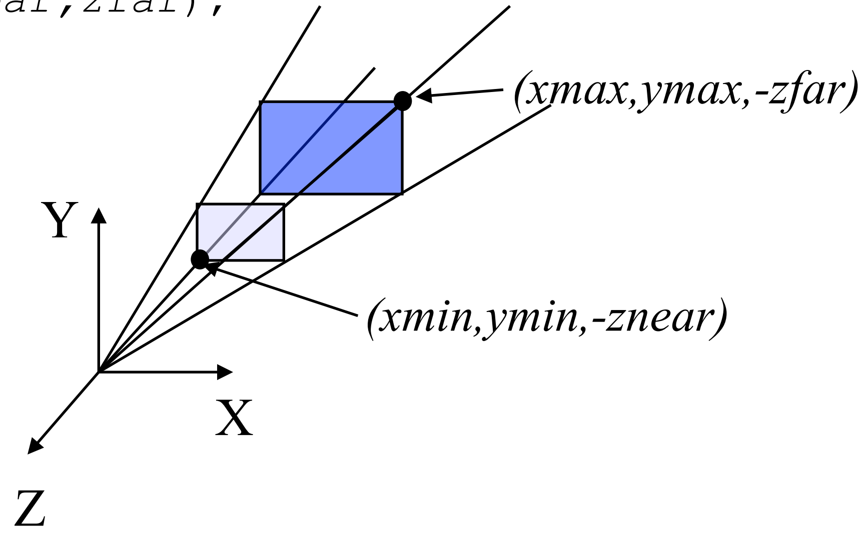


zfar,znear can be negative but *zfar>znear*

Perspective view frustum

`glFrustum(`*`xmin,xmax,ymin,ymax,znear,zfar`*`);`

zfar >0 and znear >0
zfar,znear are distance to plane
from the centre of projection
projection plane is orthogonal to z-axis

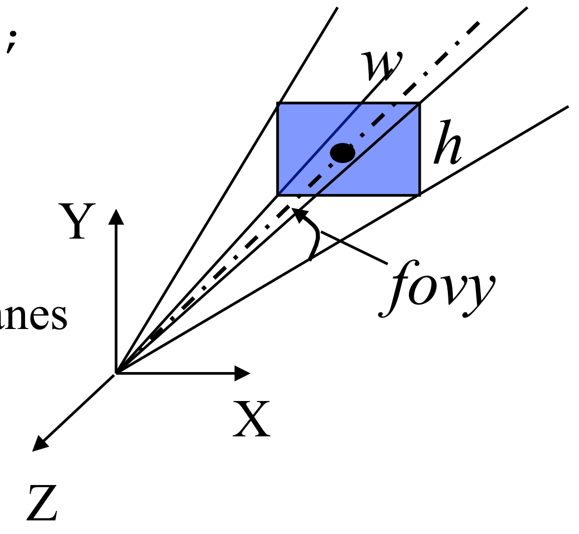


`gluPerspective(`*`fovy,aspect,near,far`*`);`

GLUT utility function
specify view frustum by field-of-view angle
= angle between top and bottom planes
view frustum is symmetric about y=0 & x=0 planes
near/far as in `glFrustum()`

For a plane at distance *d*: $fovy = 2\tan^{-1}\left(\frac{h}{2d}\right)$

$aspect = \frac{w}{h}$



Example: Setting up a Perspective Projection

Set up a perspective projection to view objects within a 90degree field of view at a distance of 1 to 2 units

2 implementations:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glfrustum(-1,-1,1,1,1,2);
```

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(90,1.0,1,2);
```

Hidden-surface removal

When we display a scene we want to render the nearest visible surface for objects inside the view frustum.

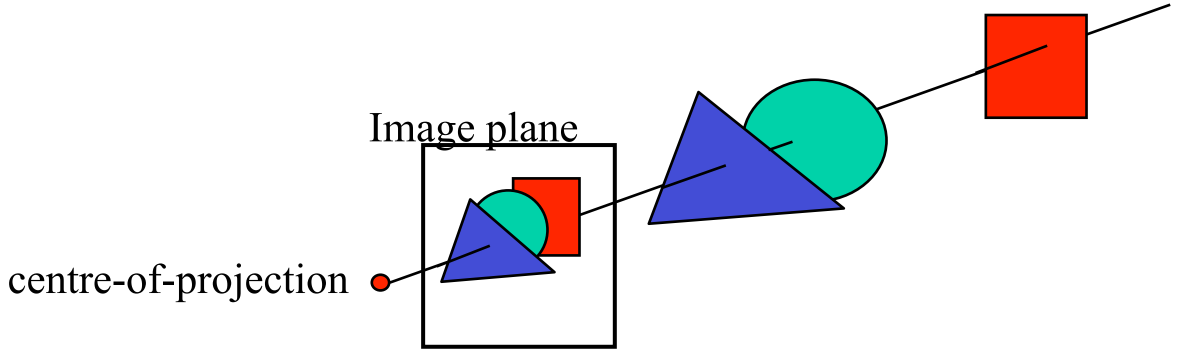
- ‘Hidden-surface-removal’ algorithms

z-buffer algorithm:

- as each polygon in the scene is rasterised (projected to image plane and sampled) we keep track of the distance from the centre-of-projection to the **closest point** on any projected polygon
- if a polygon is closer then keep the distance and atributes for that polygon.

Image-space hidden surface removal algorithm supported by OpenGL
Worst-case complexity is proportional to number of polygons

ie real-time if polygons can be rendered in real-time



Enabling Hidden-surface-removal

To enable z-buffering OpenGL

(1) Initialise display mode with a depth buffer
(2) Enable depth buffering
(3) Clear depth buffer each time scene is rendered

```
init() {
   ...
   glInitDisplayMode(...|...|GLUT_DEPTH);
   glEnable(GL_DEPTH_TEST);
   ...
}

display(){
   ...
   glClear(GL_DEPTH_BUFFER_BIT);
   ...
}
```

Example: Walking through a scene see sec 5.6

Example: Shadows sec5.9