LECTURE 5

Axis Transformation

1

WHAT HAVE WE SEEN SO FAR?

Basic representations (point, vector)

Basic operations on points and vectors (dot product, cross products, etc.)

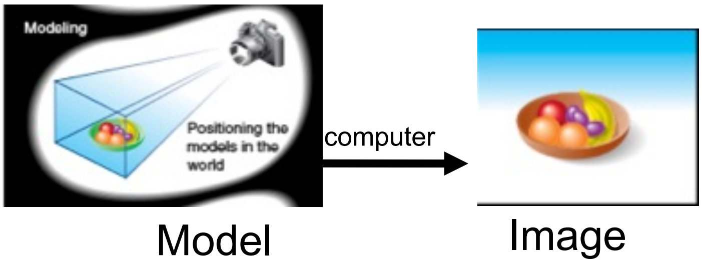
Transformation – manipulative operators on the basic representation (translate, rotate, deformations) – 4x4 matrices to "encode" all these.

2

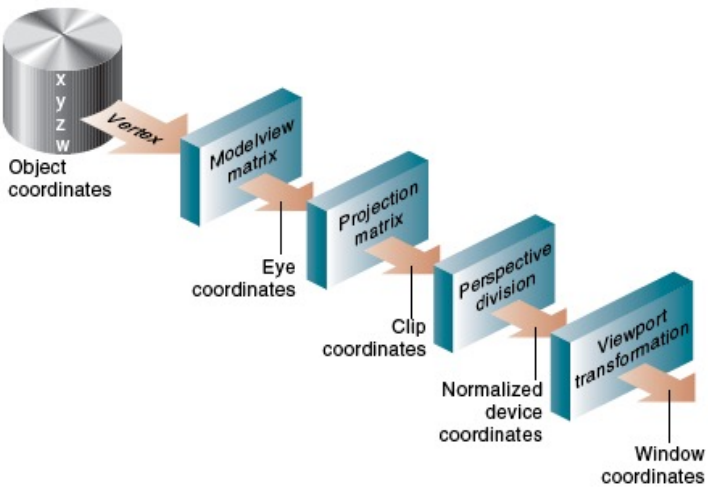
WHY DO WE NEED THIS?

In order to generate a picture from a model, we need to be able to not only specify a model but also manipulate the model in order to create more interesting images.

From a model, how do we generate an image



OVERVIEW

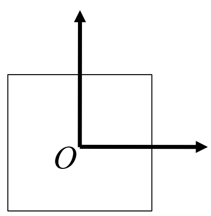


COORDINATE SYSTEMS

- Object coordinates
- World coordinates
- Camera coordinates
- Normalized device coordinates
- Window coordinates

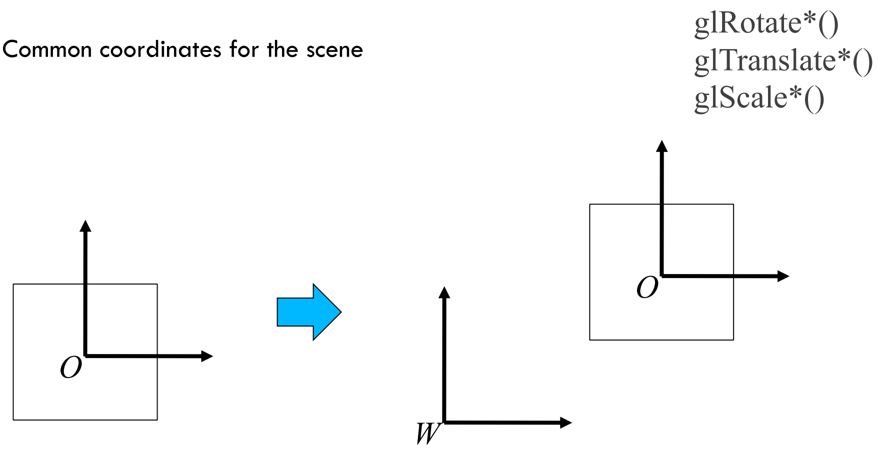
OBJECT COORDINATES

Convenient place to model the object



WORLD COORDINATES

Common coordinates for the scene

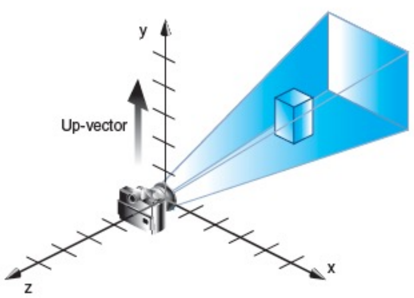


Object coordinates

World coordinates

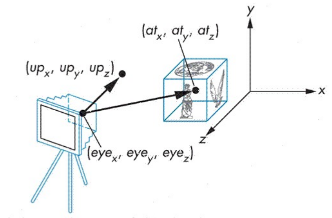
CAMERA COORDINATES

Coordinate system with the camera in a convenient pose



CAMERA COORDINATE

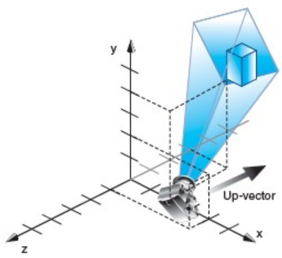
- **Viewing with gluLookAt()**
  - **looking at a scene from an arbitrary point of view.**
  - Takes 3 sets of arguments
    - specify the location of the viewpoint
    - define a reference point toward which the camera is aimed
    - indicate which direction is up.

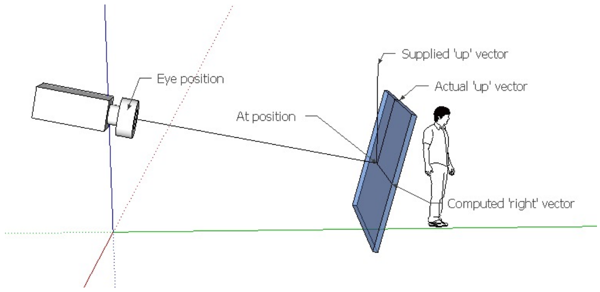


gluLookAt(GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ,
GLdouble atX, GLdouble atY, GLdouble atZ,
GLdouble upX, GLdouble upY, GLdouble upZ);

CAMERA COORDINATES

- The effect of a gluLookAt()
  - gluLookAt(4.0, 2.0, 1.0, 2.0, 4.0, -3.0, 2.0, 2.0, -1.0)
  - The camera position (eyex, eyey, eyez) is at (4, 2, 1).
  - It looking at the model, so the reference point is at (2, 4, -3)
  - An orientation vector of (2, 2, -1) is chosen to rotate the viewpoint to this 45-degree angle.





The blue window can be thought of as the 'near-plane' that your imagery is drawn on (your monitor).

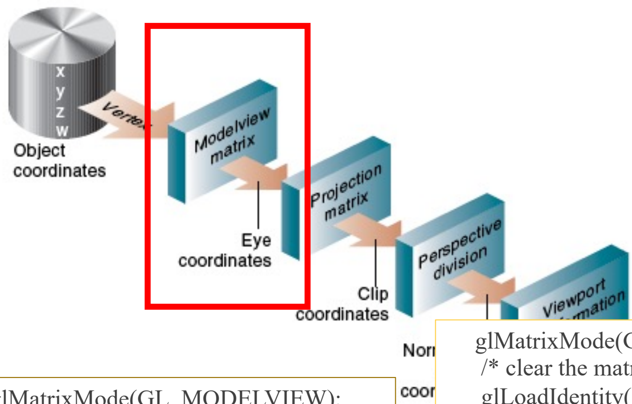
-If all you supply is the eye-point and the at-point, that window is free to spin around.

- You need to give an extra 'up' direction to pin it down.

-OpenGL will project the 'up' vector down, so that it forms a 90 degree angle with the 'z' vector defined by *eye* and *at*

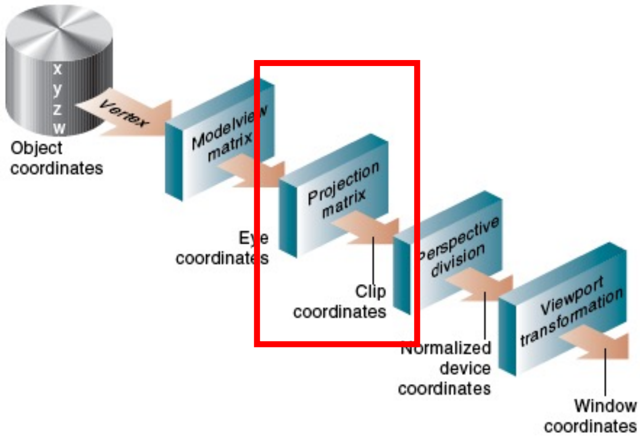
-Once 'in' (z) and 'up' (y) directions are defined, it's easy to calculate the 'right' or (x) direction from those two

In this figure, the 'supplied' up vector is (0,1,0) if the blue axis is in the y direction. If you were to give (1,1,1), it would most likely rotate the image by 45 degrees because that's saying that the top of the blue window should be pointed toward that direction.



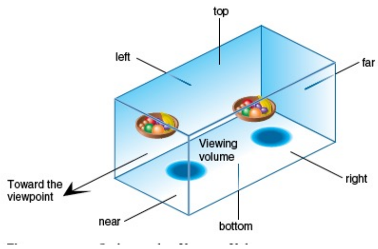

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
/* apply transformation */
glMultMatrixf(N);
glMultMatrixf(M);
glMultMatrixf(L);
glBegin(GL_POINTS);
glVertex3f(v);
glEnd();
```

```
glMatrixMode(GL_MODELVIEW);
 /* clear the matrix */
 glLoadIdentity();
 /* viewing transformation */
 gluLookAt(0.0, 0.0, 5.0,
           0.0, 0.0, 0.0,
           0.0, 1.0, 0.0);
 /* modeling transformation */
 glScalef(1.0, 2.0, 1.0);
 glRotatef(45.0, 0.0,0.0,1.0);
 glutWireCube(1.0);
```



ORTHOGRAPHIC PROJECTION

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

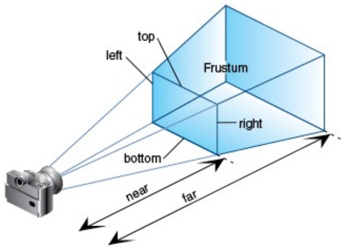


- Both near and far may be positive, negative, or even set to zero.
  - These distances are negative if the plane is to be behind the viewer
- near and far should not be the same value.

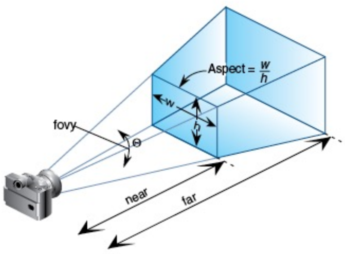
PERSPECTIVE PROJECTION

- Creates a matrix for a perspective-view frustum and multiplies the current matrix by it
- near and far:
  - the distances from the viewpoint to the near and far clipping planes.
  - They should always be positive.

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



PERSPECTIVE PROJECTION




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

gluPerspective(60.0, 1.0, 1.5, 20.0)

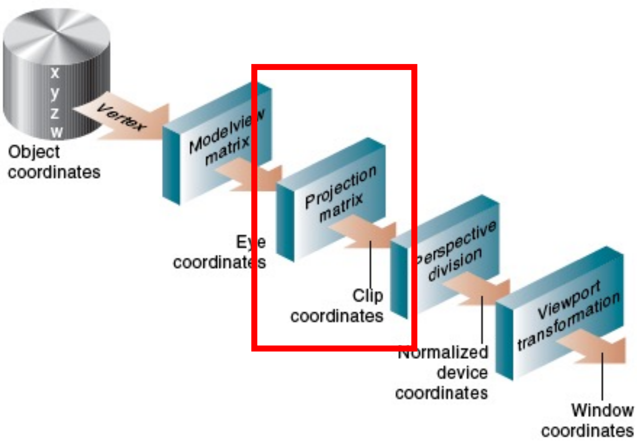
- the angle of the field of view (θ) in y-direction (y-z plane)
  - Range: (0.0~180.0)
- the aspect ratio of the width to the height (w/h)
- the distance between the viewpoint and the near and far clipping planes

→ creates a viewing volume of the same shape as glFrustum() does

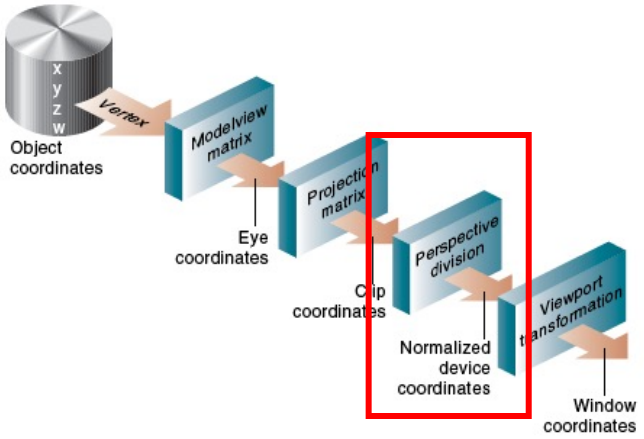
PERSPECTIVE PROJECTION

Projection Matrix

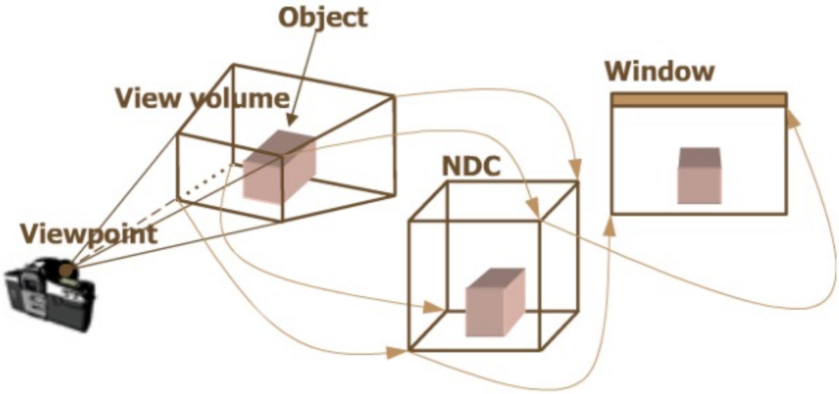
$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} \frac{2near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2\,far\cdot near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$




```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1.0, 1.0,   // left, right
          -1.0, 1.0,   // bottom, top
           1.5, 20.0); // near, far
```

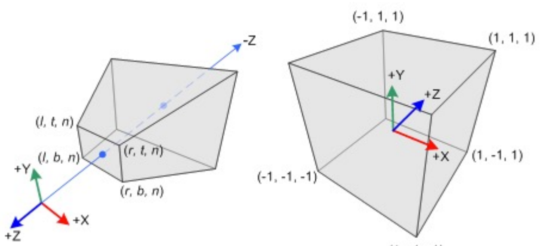

x
y
z
w
Vertex
Object coordinates
Modelview matrix
Eye coordinates
Projection matrix
Clip coordinates
Perspective division
Normalized device coordinates
Viewport transformation
Window coordinates

OVERVIEW


Object
View volume
Window
NDC
Viewpoint

PERSPECTIVE PROJECTION

- In perspective projection, a 3D point in a truncated pyramid frustum (eye coordinates) is mapped to a cube (NDC)
  - the range of x-coordinate from [left, right] to [-1, 1]
  - the y-coordinate from [bottom, top] to [-1, 1]
  - the z-coordinate from [-near, -far] to [-1, 1]

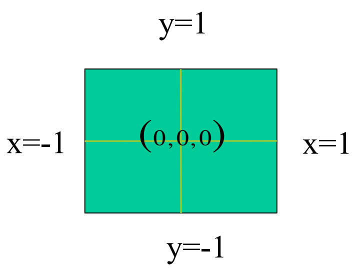


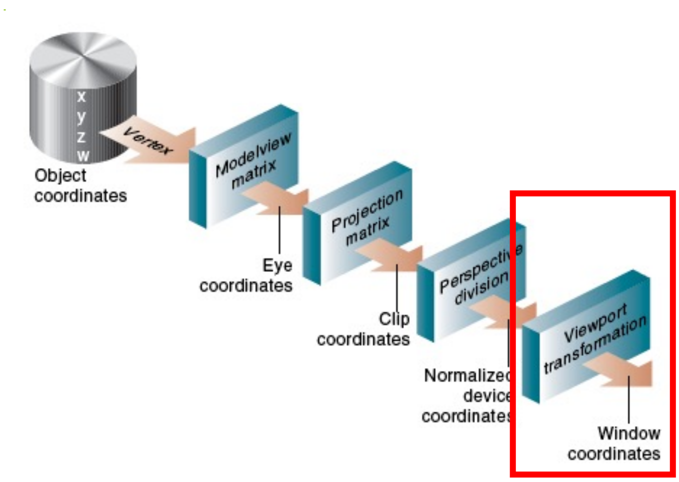
NORMALIZED DEVICE COORDINATES

Device independent coordinates

Visible coordinate usually range from:

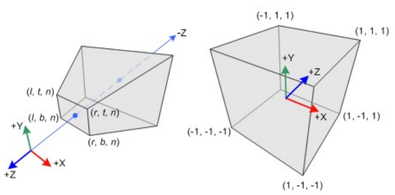
$$-1 \le x \le 1$$
$$-1 \le y \le 1$$
$$-1 \le z \le 1$$

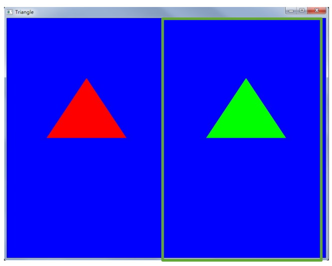




//gluPerspective(fovy, 1.0, near, far);
glViewport(0, 0, 400, 200);

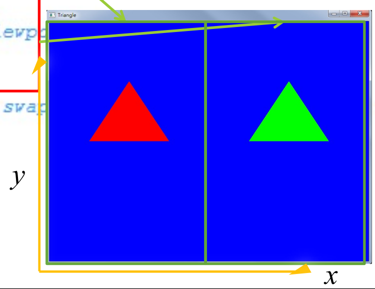
WINDOW COORDINATE





SAMPLE VIEW—MULTIVIEWPORT

```
void RenderScene(void)
{// Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);
    glLoadIdentity();
    glViewport(0,0,width/2,height);//viewport1
    glColor3fv(vColor[0]);
    DrawTriangle();
    glLoadIdentity();
    glViewport(width/2,0,width/2,height);//viewp
    glColor3fv(vColor[1]);
    DrawTriangle();
    glutSwapBuffers(); // Perform the buffer swa
}
```



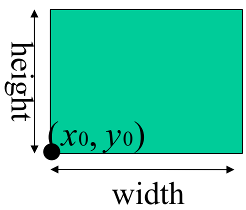
WINDOW COORDINATES

Adjusting the NDC to fit the window:

(x_0, y_0) is the lower left of the window

$$x_w = (x_{nd} + 1)\left(\frac{width}{2}\right) + x_0$$

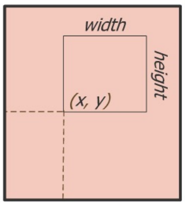
$$y_w = (y_{nd} + 1)\left(\frac{height}{2}\right) + y_0$$

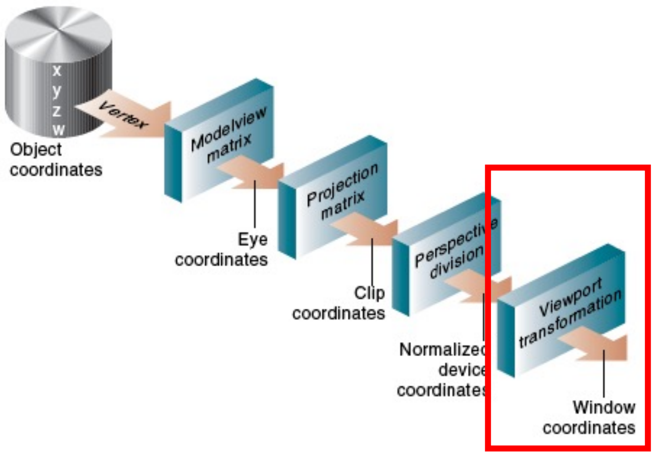


X_w: window coordinate,
X_{nd}: normalized device coordinate, $-1 \le x_{nd} \le 1$

VIEWPORT TRANSFORMATIONS

- 將投影轉換後得到的二維影像圖對應到螢幕上呈現的某個視窗中的位置(視窗坐標軸)。
- 此轉換為轉換到視窗上的最後一次轉換。
- glViewport(x, y, w, h);





//gluPerspective(fovy, 1.0, near, far);
glViewport(0, 0, 400, 200);

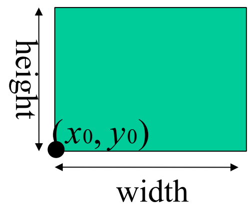
WINDOW COORDINATES

Adjusting the NDC to fit the window:

(x_0, y_0) is the lower left of the window

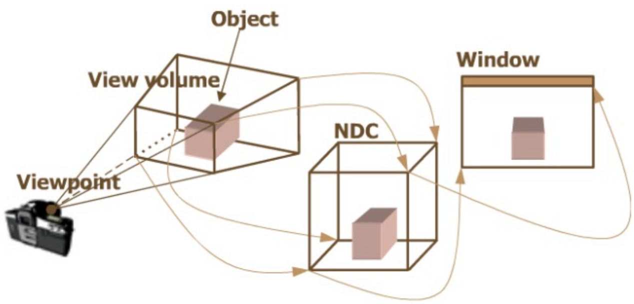
$$x_w = (x_{nd} + 1)\left(\frac{width}{2}\right) + x_0$$

$$y_w = (y_{nd} + 1)\left(\frac{height}{2}\right) + y_0$$



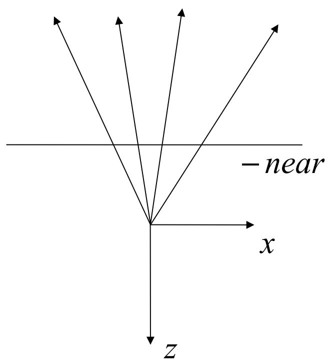
X_w: window coordinate,
X_{nd}: normalized device coordinate, $-1 \le x_{nd} \le 1$

Back to Projection…



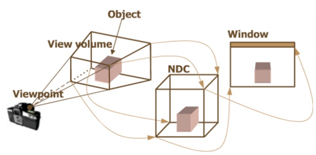
PERSPECTIVE PROJECTION

Taking the camera coordinates to NDC



PERSPECTIVE PROJECTION

Taking the camera coordinates to **NDC**

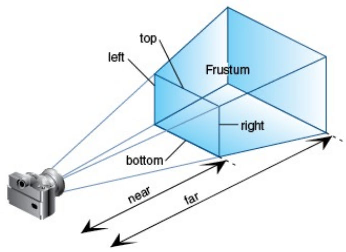


$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} \frac{2near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2\,far\cdot near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

PERSPECTIVE PROJECTION

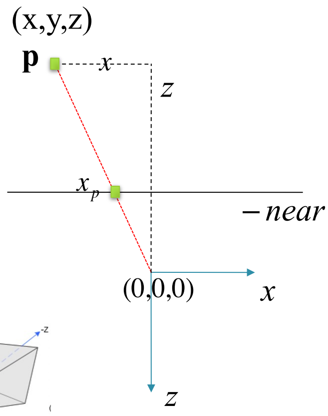
- Creates a matrix for a perspective-view frustum and multiplies the current matrix by it
- near and far:
  - the distances from the viewpoint to the near and far clipping planes.
  - They should always be positive.

void glFrustum(GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top,
GLdouble near, GLdouble far);



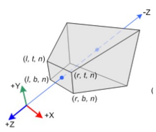
void glFrustum(GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top,
GLdouble near, GLdouble far);

PERSPECTIVE PROJECTION



$$\frac{x_p}{-near} = \frac{x}{z}$$

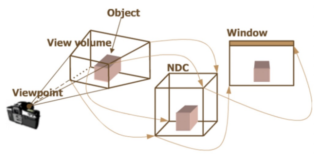
$$x_p = -near\frac{x}{z}$$



near is positive
→ -near is the coordinate

x_p：P點投影至-near平面之座標 (未轉換至NDC座標)

PERSPECTIVE PROJECTION

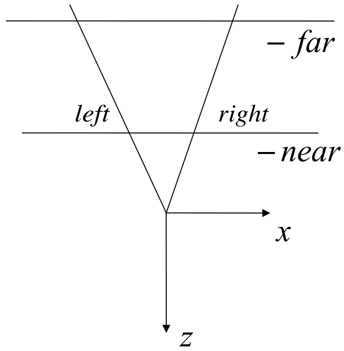


Method1:

$$x_p = -near\frac{x}{z}$$ ← x投影至-near平面之座標 (未轉換至NDC座標)

轉換至NDC 座標:

Map (left,right) to (-1,1) when z = -near



$$-near\frac{x}{z} - left$$

$$\frac{2}{right-left}\left(-near\frac{x}{z} - left\right)$$

$$\frac{2}{right-left}\left(-near\frac{x}{z} - left\right) - 1$$

$$\Rightarrow \quad \frac{-2near}{right-left}\frac{x}{z} - \frac{right+left}{right-left}$$ ← x投影至-near平面,並轉換至NDC座標

- A simple mapping example:

$$\frac{2}{right-left}\left(-near\frac{x}{z} - left\right) - 1$$

left=3 right =9

Left=-1 right =1

x'=5

x'$_{ndc}$=?

$$(5-3)\frac{1-(-1)}{9-3} + (-1)$$

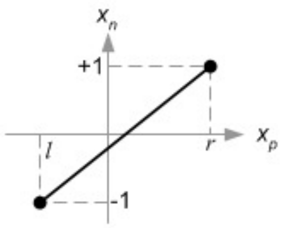
$$= \frac{2}{9-3}(5-3) - 1$$

Method2:

PERSPECTIVE PROJECTION

Taking the camera coordinates to NDC

Map (left,right) to (-1,1)



Mapping from x_p to x_n

$$x_p = -near\frac{x}{z}$$

$$x_n = \frac{2}{r-l}x_p - \frac{r+l}{r-l}$$

$$\begin{aligned} x_n &= \frac{2}{r-l}(-near\frac{x}{z}) - \frac{r+l}{r-l} \\ &= \frac{-2near}{r-l}(\frac{x}{z}) - \frac{r+l}{r-l} \\ &= (\frac{2near}{r-l}x + \frac{r+l}{r-l}z)/(-z) \end{aligned}$$

x_p : x_e is projected to x_p

x_n : linearly mapped x_p to NDC