Computer Graphics

View in 2D & 3D

Teacher: A.prof. Chengying Gao(高成英)

E-mail: mcsgcy@mail.sysu.edu.cn

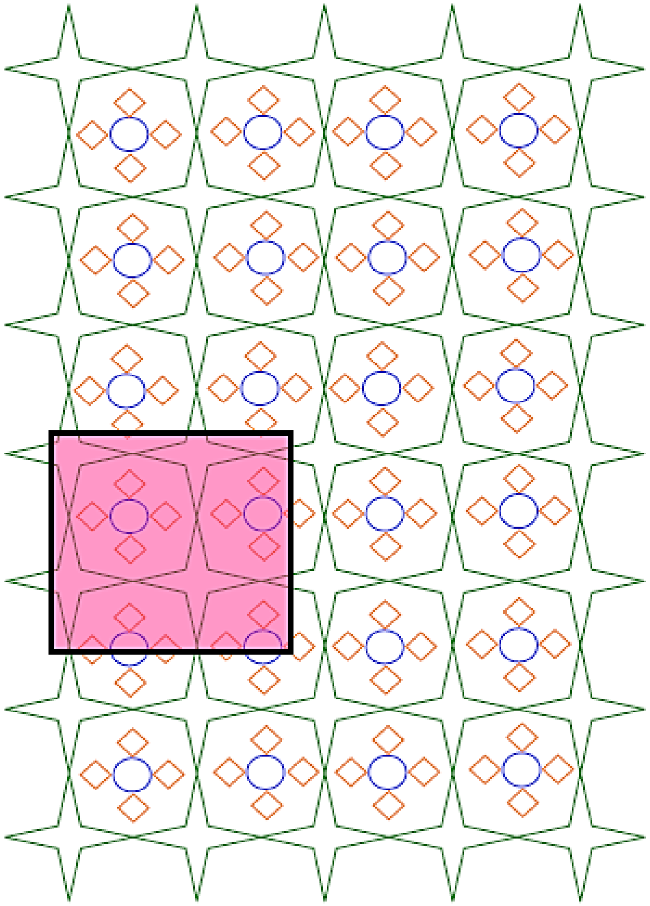
School of Data and Computer Science

Outline

- **2D Viewing Transformation**
- 3D Viewing Transformation
  - Computer view
    - Positioning the camera
    - Projection

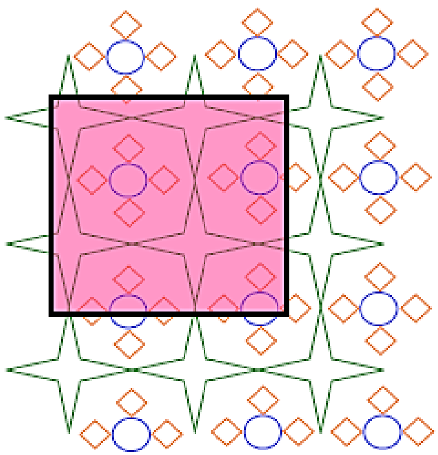
Viewing Transformation

- The world is infinite (2D or 3D) but the screen is finite
- Depending on the details the user wishes to see, he limits his view by specifying a window in this world

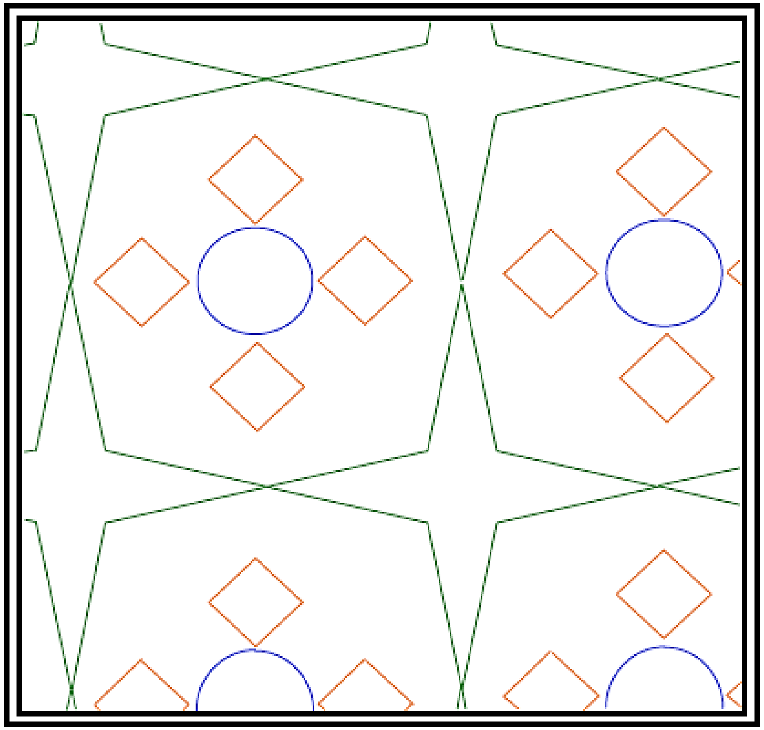


2D Viewing Transformation

- By applying ***appropriate transformations*** we can map the world seen through the window on to the screen



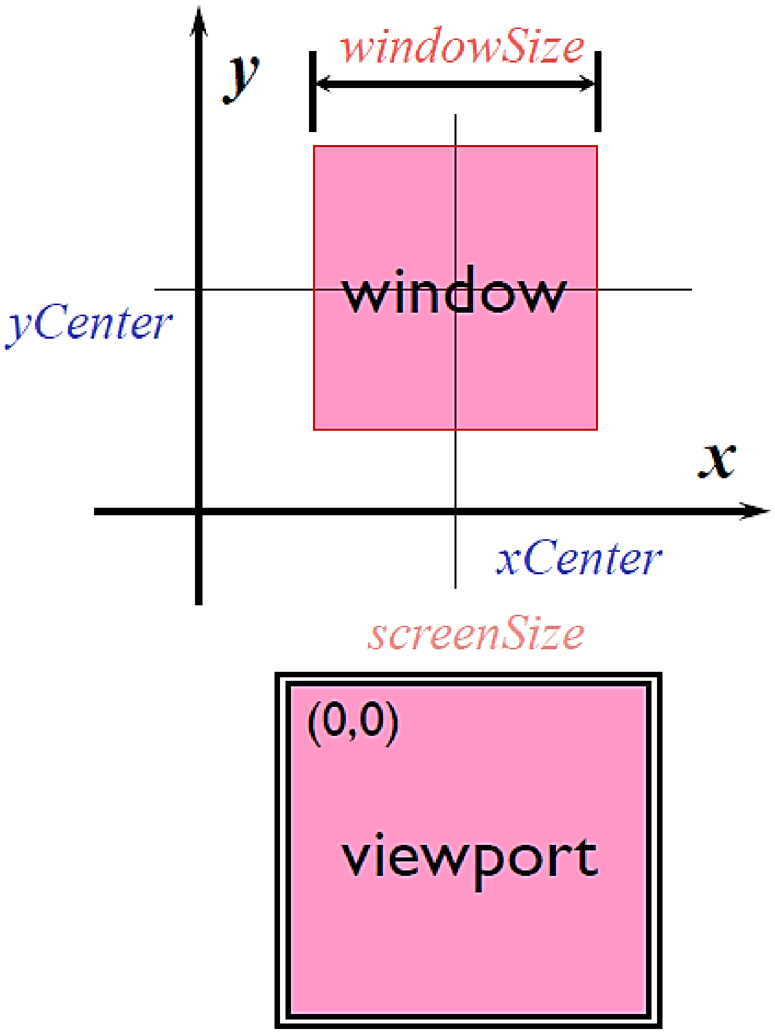
2D World



Screen

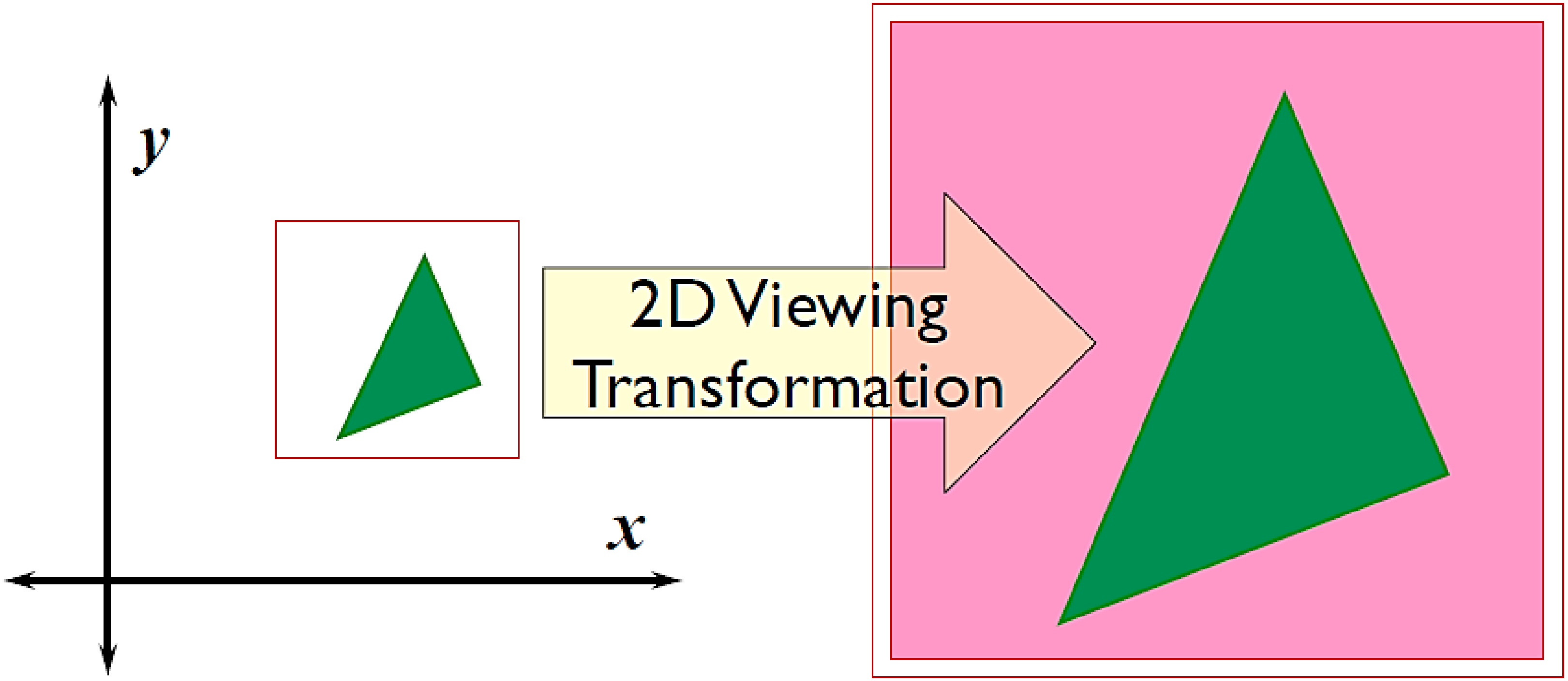
2D Viewing Transformation

- ***Window*** is a rectangular region in the 2D world specified by
  - a ***center*** *(xCenter, yCenter)* and
  - ***size*** *windowSize*
- Screen referred to as ***Viewport*** is a discrete matrix of pixels specified by
  - ***size*** *screenSize* (in pixels)

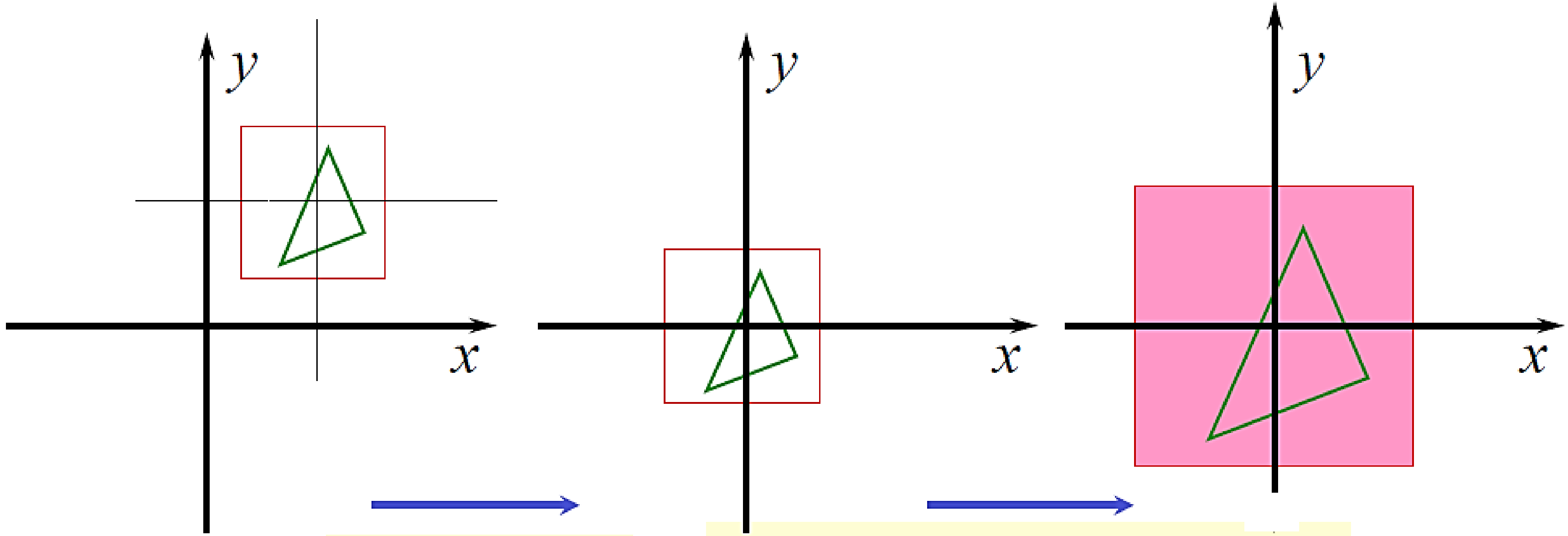


2D Viewing Transformation

- Mapping the 2D world seen in the ***window*** on to the ***viewport*** is 2***D viewing transformation***
  - also called ***window to viewport transformation***



Deriving Viewport Transformation

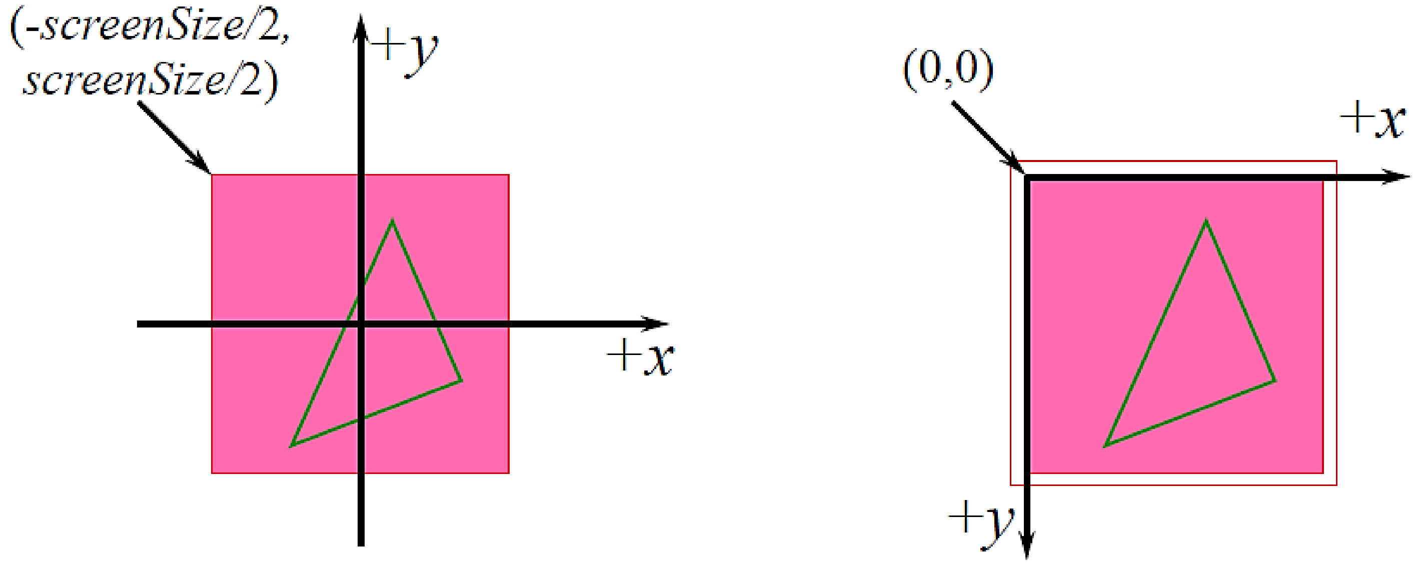


$$x - xCenter$$
$$y - yCenter$$

$$(x - xCenter) \bullet scaleFactor$$
$$(y - yCenter) \bullet scaleFactor$$

$$where, \quad scaleFactor = \frac{screenSize}{windowSize}$$

Deriving Viewport Transformation



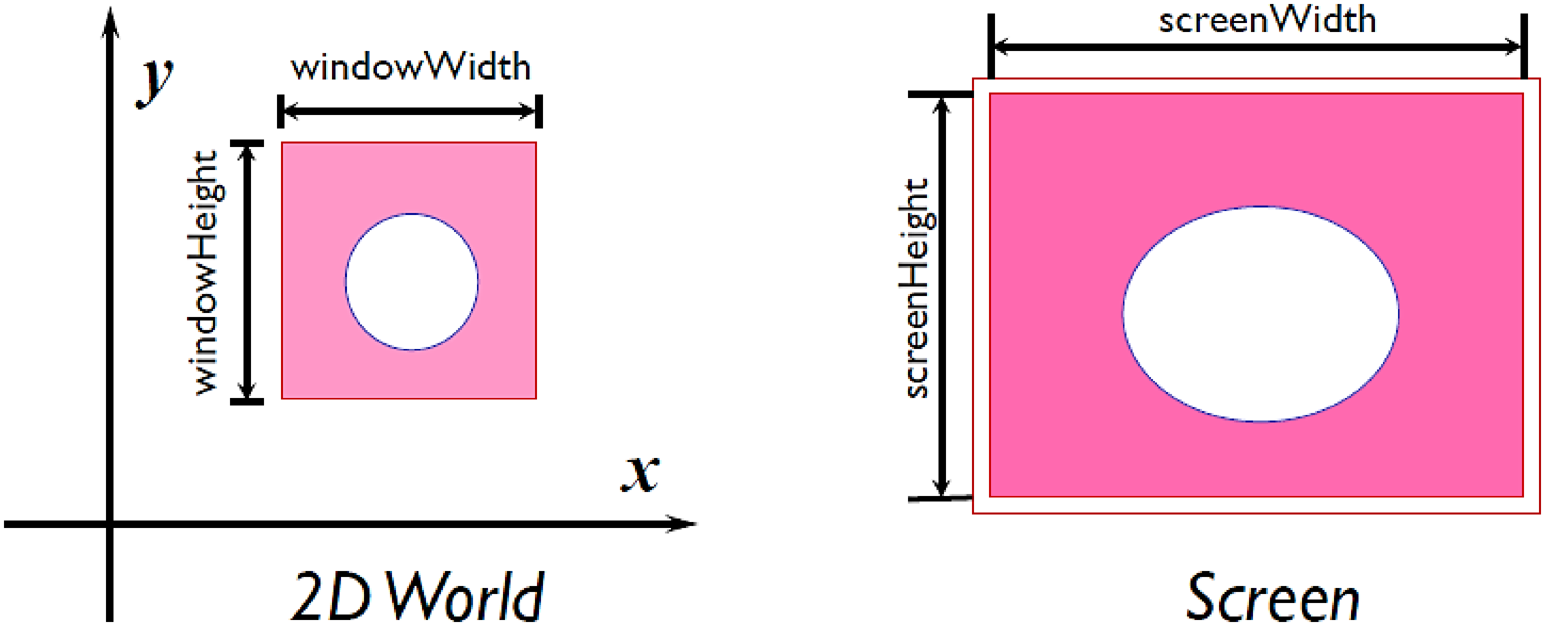
$$\frac{screenSize}{2} + (x - xCenter) \bullet scaleFactor$$

$$\frac{screenSize}{2} - (y - yCenter) \bullet scaleFactor$$

- Given any point in the 2D world, the above transformations maps that point on to the screen

The Aspect Ratio (纵横比)

- In 2D viewing transformation the ***aspect ratio*** is maintained when the scaling is uniform
- ***scaleFactor*** is same for both x and y directions



OpenGL Commands

gluOrtho2D(left, right, bottom, top)

Creates a matrix for projecting 2D coordinates onto the screen and multiplies the current matrix by it.

glViewport(x, y, width, height)

Define a pixel rectangle into which the final image is mapped.

(x, y) specifies the lower-left corner of the viewport.

(width, height) specifies the size of the viewport rectangle.

glOrtho2D是窗口变换，设置窗口的。二维绘图来说窗口由gluOrth2D()设定;
glViewport是视口变换，设置视口的。它设置的视口的左下角，以及宽度和高度。
它负责把视景体截取的图像按照怎样的高和宽显示到屏幕上。

2D Rendering

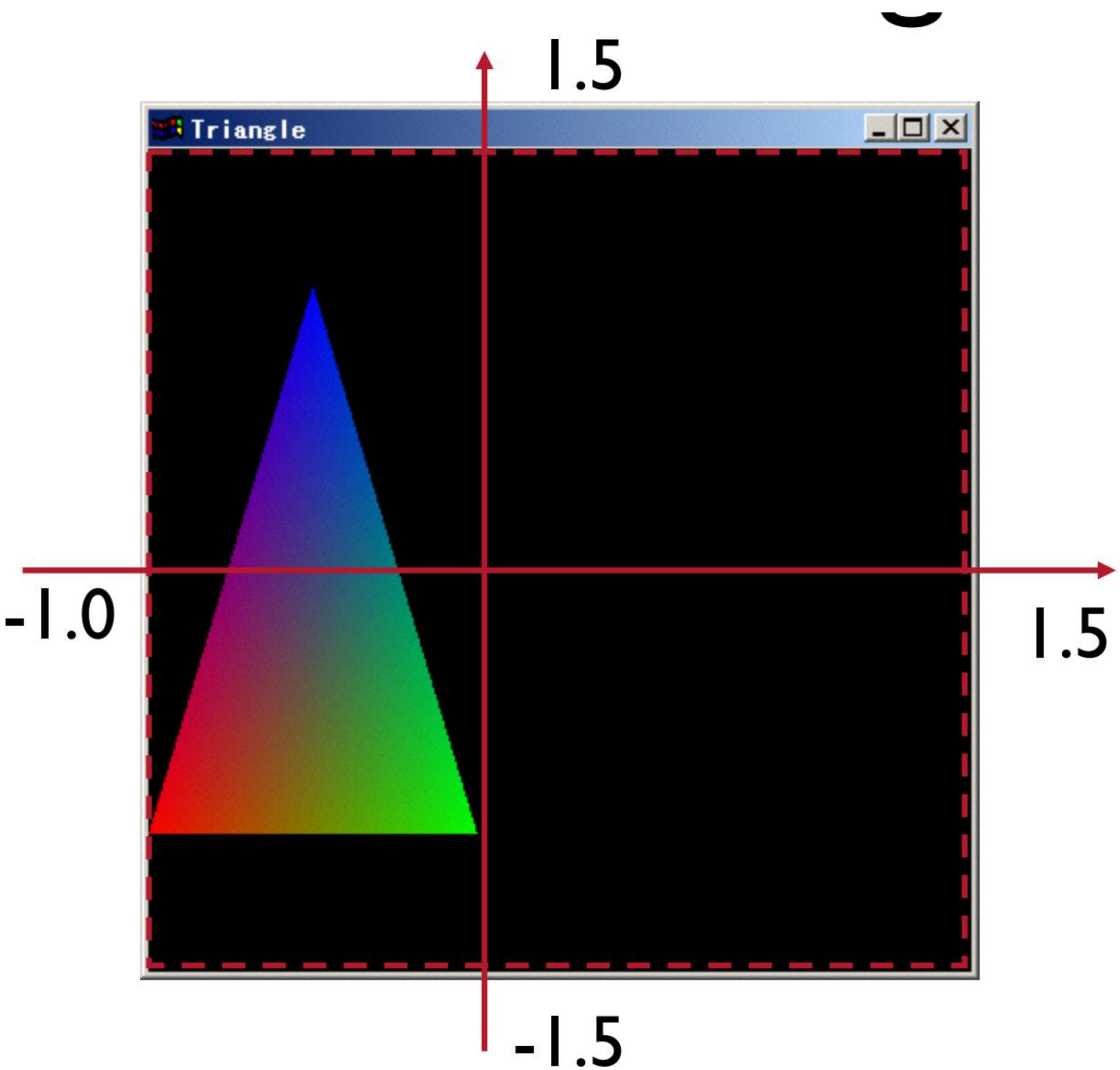
```
void myReshape(GLsizei w,GLsizei h)
{
    glViewport(0,0,w,h);//设置视口
    glMatrixMode(GL_PROJECTION);//指明当前矩阵为GL_PROJECTION
    glLoadIdentity();//将当前矩阵置换为单位阵

    //定义二维正视投影矩阵
    if(w <= h)
     gluOrtho2D(-1.0,1.5,-1.5,1.5*(GLfloat)h/(GLfloat)w);
    else
     gluOrtho2D(-1.0,1.5*(GLfloat)w/(GLfloat)h,-1.5,1.5);
    glMatrixMode(GL_MODELVIEW); //指明当前矩阵为GL_MODELVIEW
}
```

2D Rendering

```
void myDisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT); //刷新颜色buffer
    glShadeModel(GL_SMOOTH); //设置为光滑明暗模式
    glBegin(GL_TRIANGLES);//开始画三角形
        glColor3f(1.0,0.0,0.0);//设置第一个顶点为红色
        glVertex2f(-1.0,-1.0);//设置第一个顶点的坐标为（-1.0，-1.0）
        glColor3f(0.0,1.0,0.0);//设置第二个顶点为绿色
        glVertex2f(0.0,-1.0);//设置第二个顶点的坐标为（.0，-1.0）
        glColor3f(0.0,0.0,1.0);//设置第三个顶点为蓝色
        glVertex2f(-0.5,1.0);//设置第三个顶点的坐标为（-0.5，1.0）
    glEnd(); //三角形结束
    glFlush(); //强制OpenGL函数在有限时间内运行
}
```

2D Rendering



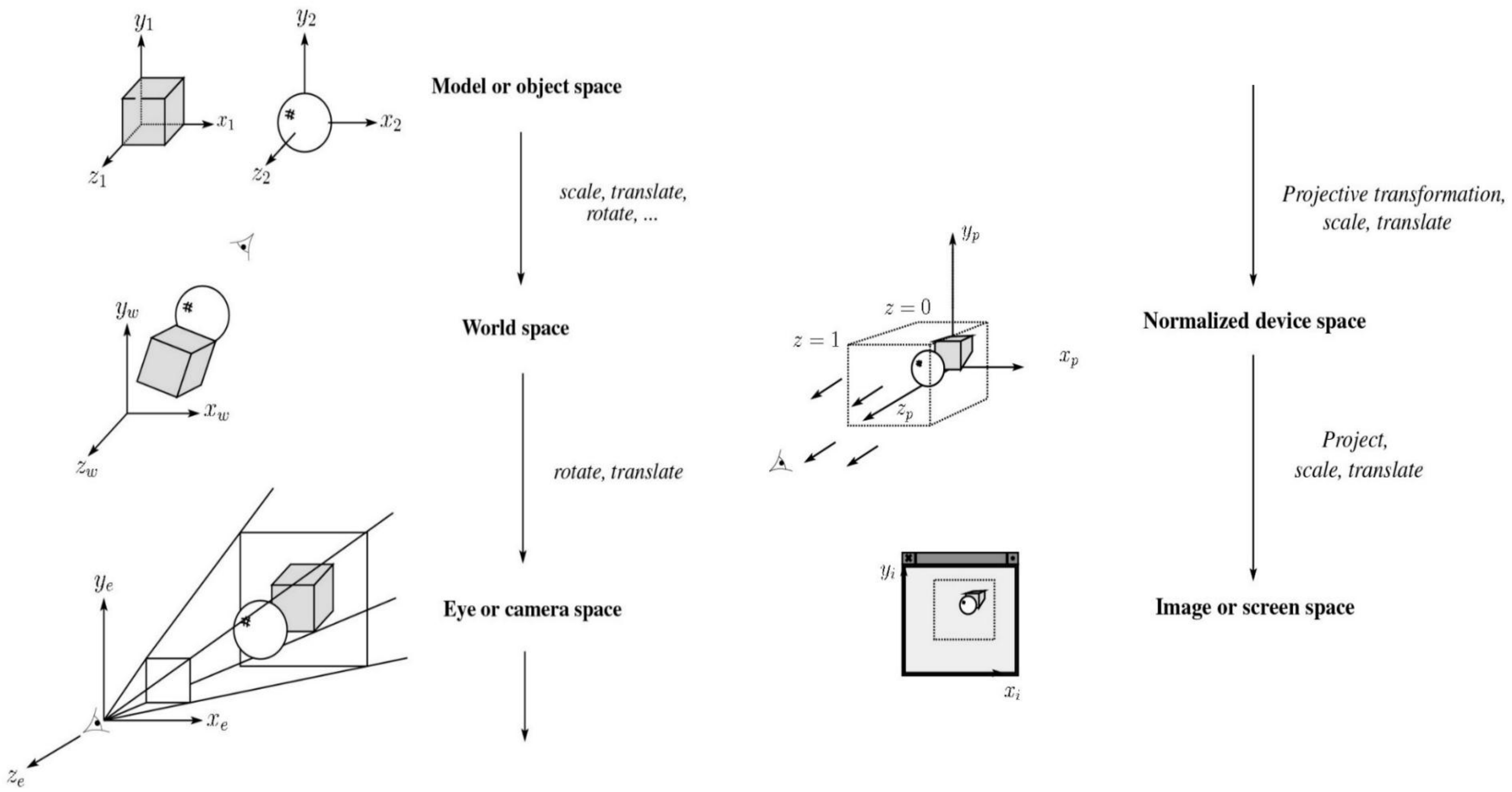
Outline

- 2D Viewing Transformation
- 3D Viewing Transfromation
    - Positioning the camera
    - Projection

3D Viewing Transformation

- To display a 3D world onto a 2D screen
  - Specification becomes complicated because there are many parameters to control
  - Additional task of reducing dimensions from 3D to 2D (projection)
  - 3D viewing is analogous to taking a picture with a camera

3D Geometry pipeline

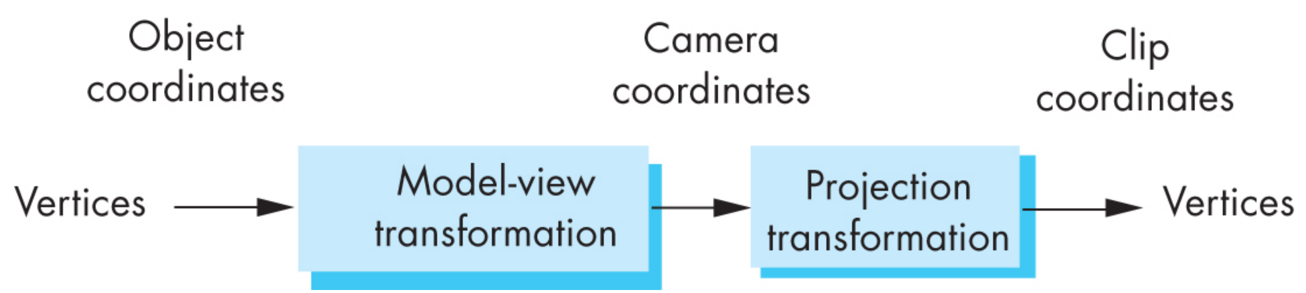


Transformation and Camera Analogy

- **Modeling transformation**
  - Shaping, positioning and moving the objects in the world scene
- **Viewing transformation**
  - Positioning and pointing camera onto the scene, selecting the region of interest
- **Projection transformation**
  - Adjusting the distance of the eye
- **Viewport transformation**
  - Enlarging or reducing the physical photograph

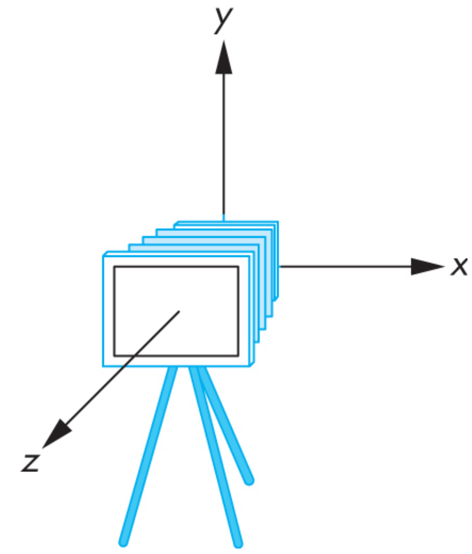
Computer view

- The view has three functions, are implemented in pipeline system
  - Positioning the camera
    - Setup the model-view matrix
  - Set the lens
    - Projection matrix
  - Clipping
    - view frustum



Camera in OpenGL

- In OpenGL, the initial world frame and camera frame are the same
- A camera located at the origin, and point to the negative direction of Z axis
- OpenGL also specifies the view frustum default, it is a center at the origin of the side length of 2 cube



Outline

- 2D Viewing Transformation
- 3D Viewing Transformation
    - **Positioning the camera**
    - Projection

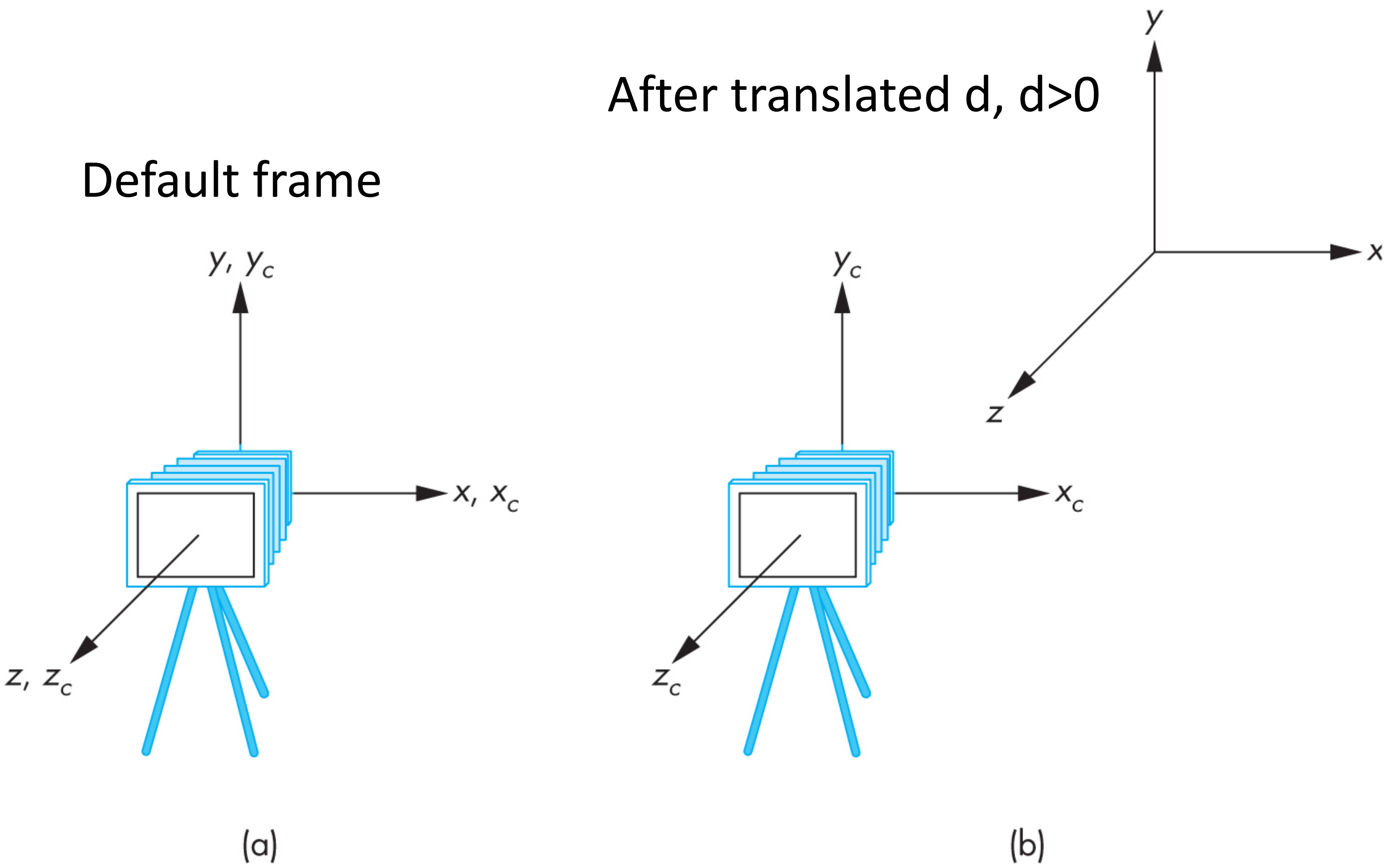
Moving the camera frame

- If you want to see objects with positive Z coordinate more, we can
  - Moves The camera along the positive Z axis
  - Moves the object along the negative Z axis
- They are equivalent, is determined by the model-view matrix
  - Need a translation: glTranslated(0.0, 0.0, d);
  - Here, $d > 0$

Moving the camera frame

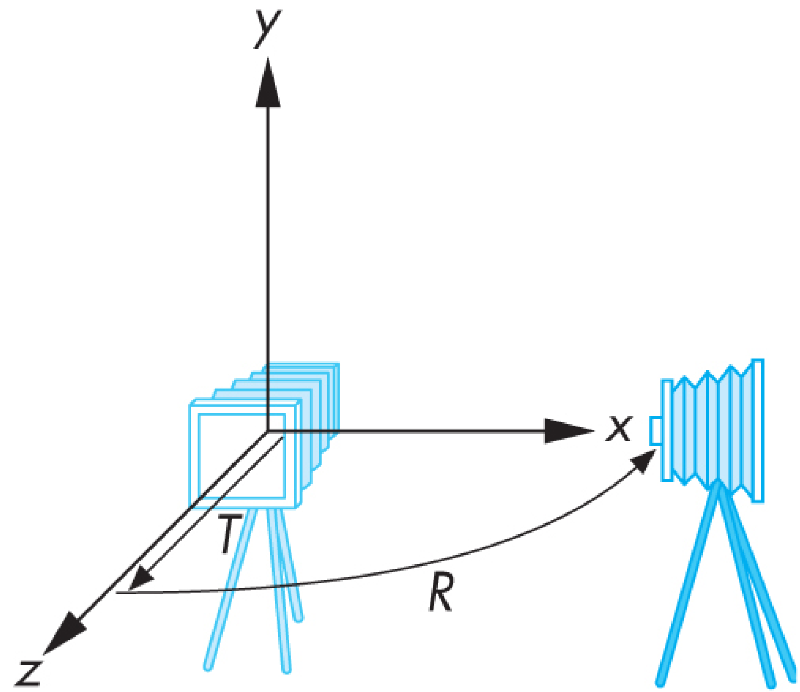
After translated d, d>0

Default frame



Moving the camera frame

- Can use a series of translation and rotation to the camera position to any position
- For example, in order to get the side view
  - Rotate the camera: R
  - Move the camera from the origin: T
  - C = TR



Viewing Specification

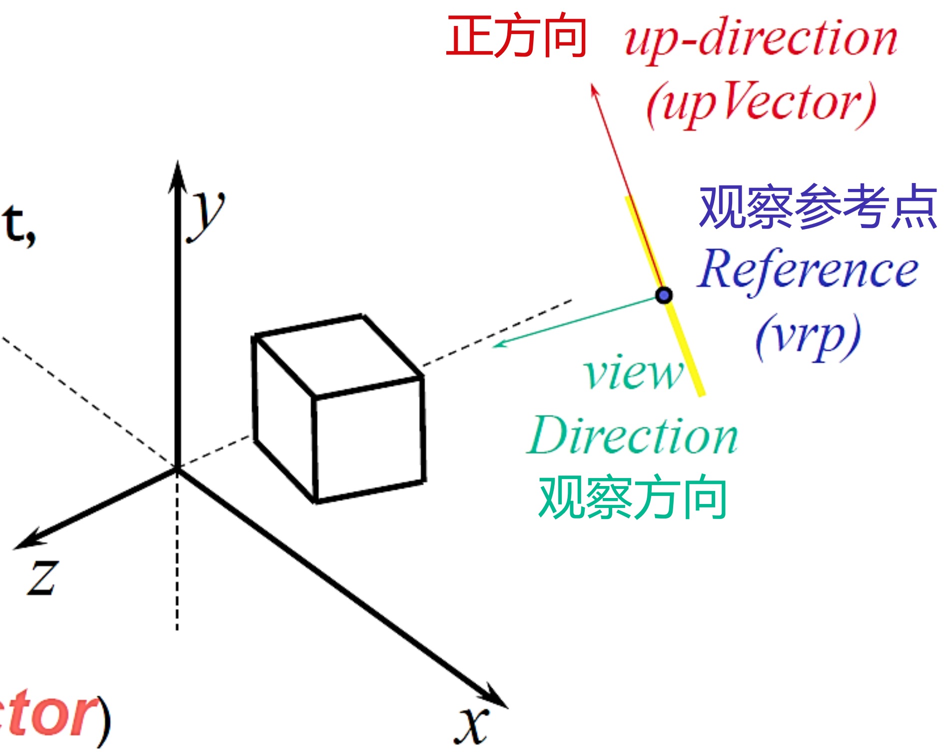
Specify

Focus point or reference point, typically on the object

(***view reference point***)

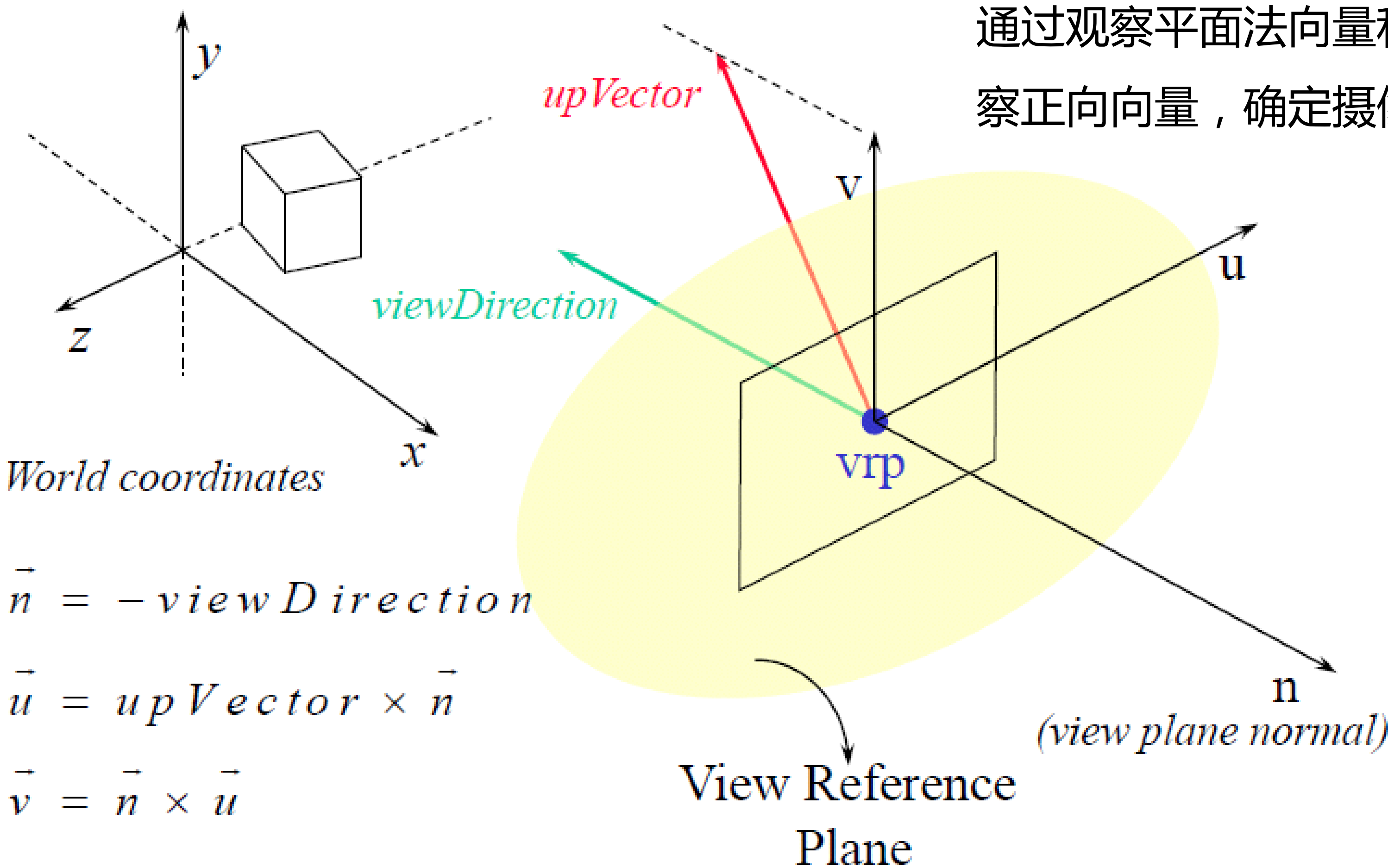
direction of viewing (***viewDirection***)

picture's up-direction (***upVector***)



All the specifications are in ***world coordinates***

View Reference Coordinate System



$$\vec{n} = -viewDirection$$

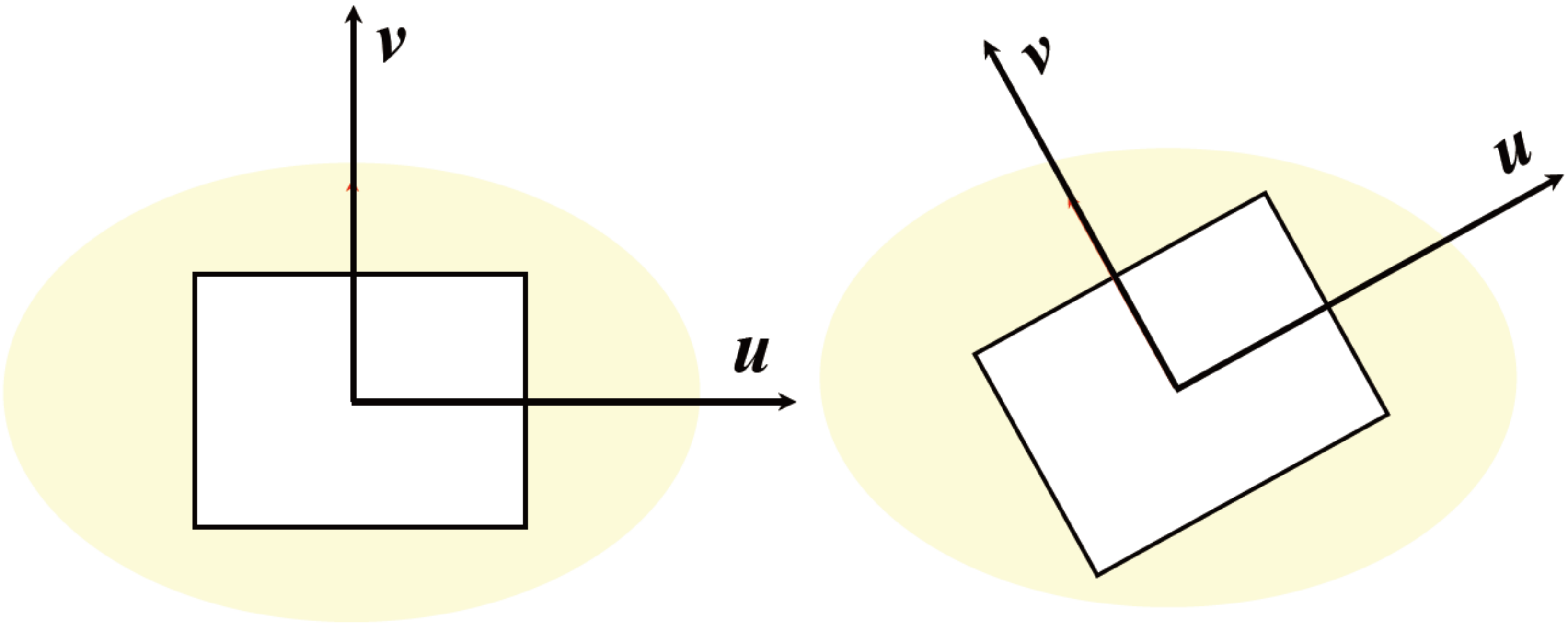
$$\vec{u} = upVector \times \vec{n}$$

$$\vec{v} = \vec{n} \times \vec{u}$$

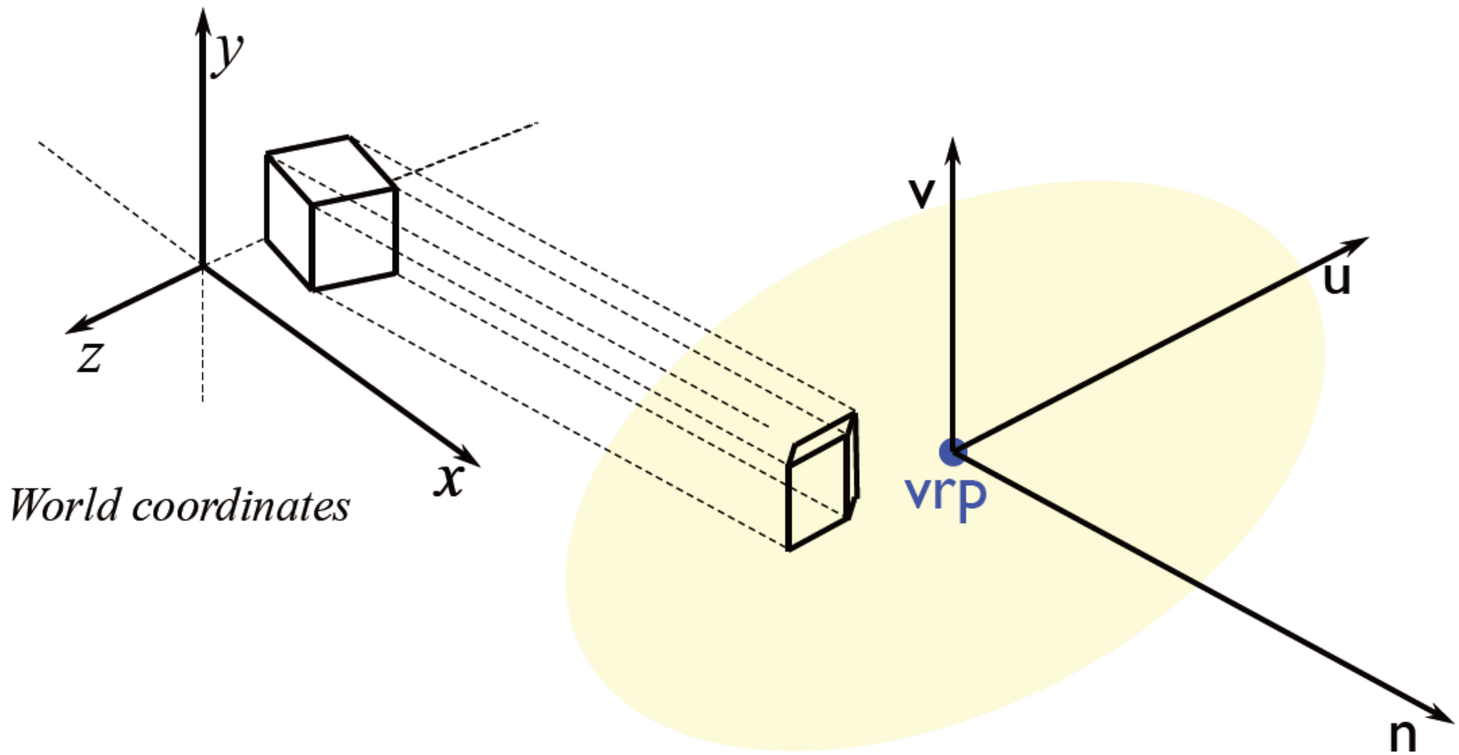
通过观察平面法向量和观察正向向量，确定摄像机。

View Up Vector

- ***upVector*** decides the orientation of the *view window* on the *view reference plane*



View Reference Coordinate System

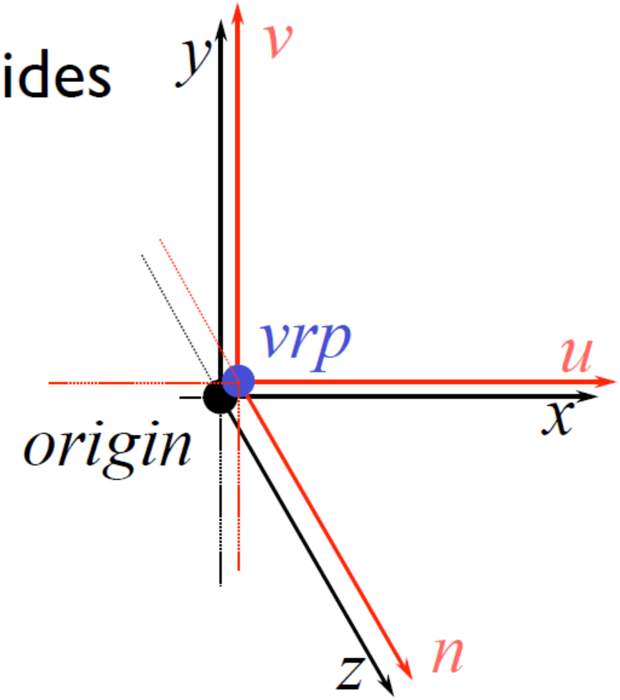


- Once the *view reference coordinate system* is defined, the next step is to project the *3D world* on to the *view reference plane*

Simplest Camera Position

- Projecting on to an arbitrary view plane looks tedious
- One of the simplest camera positions is one where ***vrp*** coincides with the ***world origin*** and

  u,v,n matches ***x,y,z***

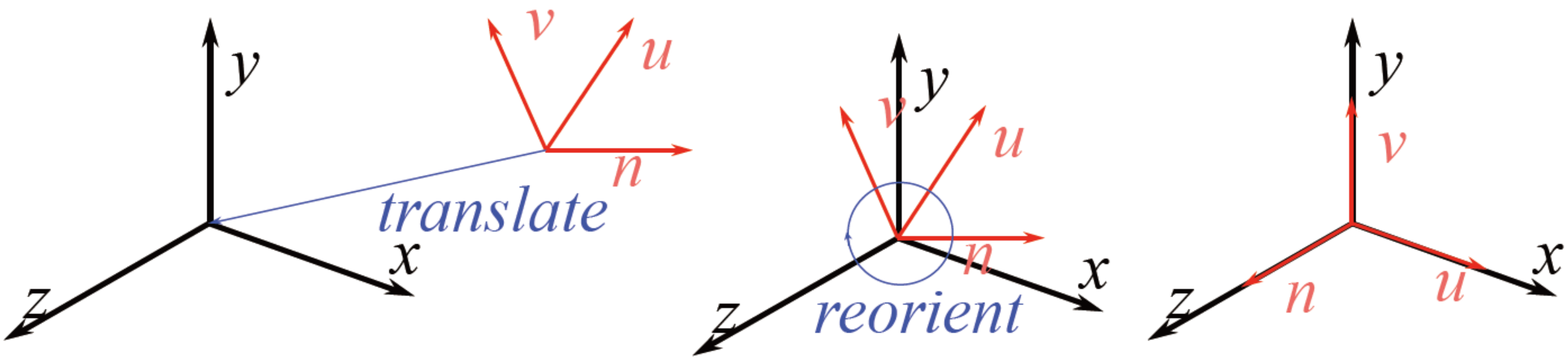


- Projection could be as simple as ignoring the z-coordinate

World to Viewing coordinate Transformation

- The world could be transformed so that the view reference coordinate system coincides with the world coordinate system
- Such a transformation is called world to viewing coordinate transformation
- The transformation matrix is also called view orientation matrix

Deriving View Orientation Matrix



- The ***view orientation matrix*** *transforms* a point from *world coordinates* to *view coordinates*

$$\begin{bmatrix} u_x & u_y & u_z & -\vec{u} \bullet vrp \\ v_x & v_y & v_z & -\vec{v} \bullet vrp \\ n_x & n_y & n_z & -\vec{n} \bullet vrp \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A More Intuitive Approach Offered by GLU

- OpenGL provides a very helpful utility function that implements the look-at viewing specification:

```
gluLookAt ( eyex, eyey, eyez,  // eye point
            atx,  aty,  atz,   // lookat point
            upx,  upy,  upz ); // up vector
```

- These parameters are expressed in world coordinates

OpenGL Viewing Transformation

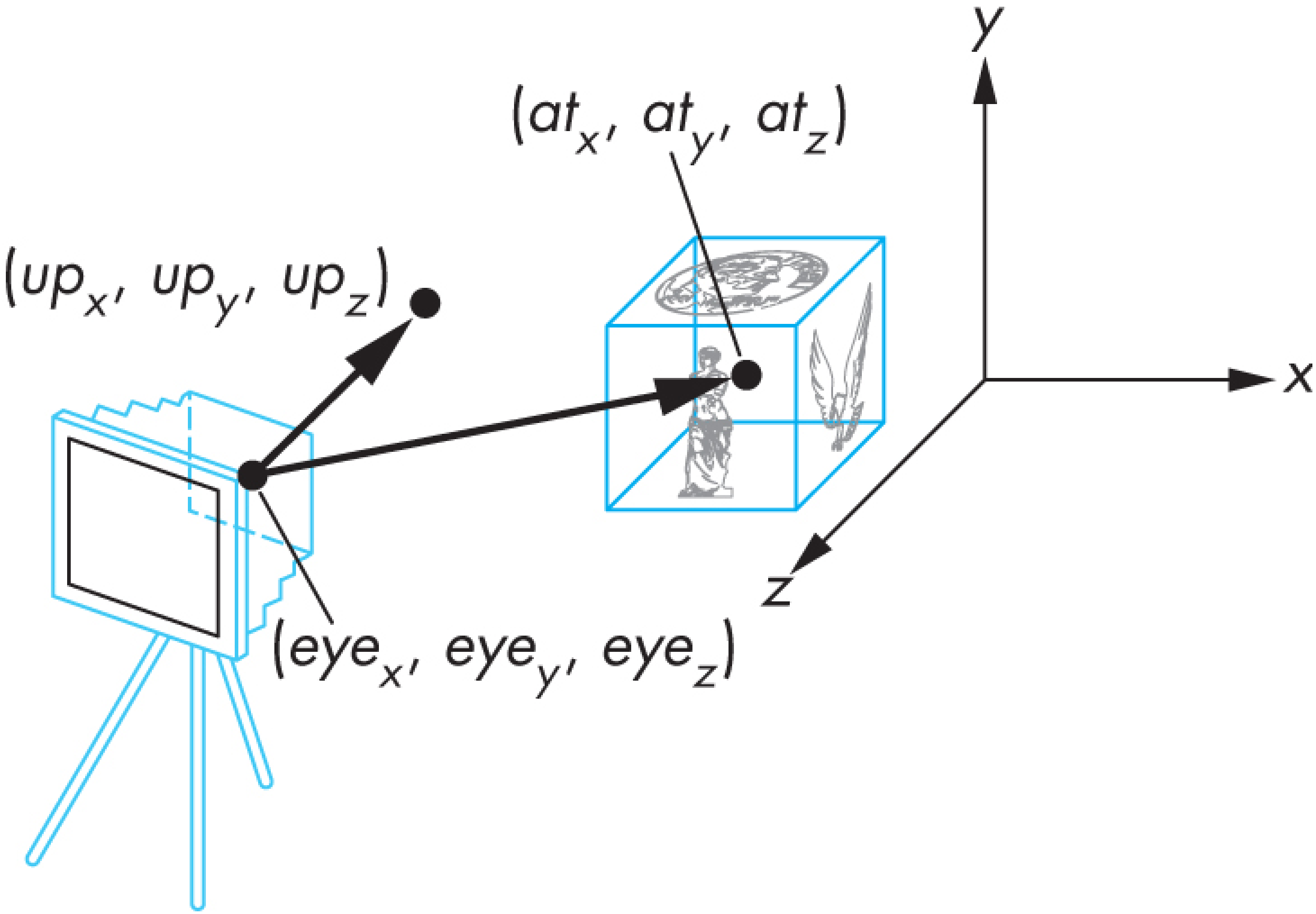
```
gluLookAt(ex,ey,ez,lx,ly,lz,ux,uy,uz)
```

- postmultiplies current matrix, so to be safe:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(ex,ey,ez,lx,ly,lz,ux,uy,uz)
// now ok to do model transformations
```

它封装了世界坐标系到观察坐标系的转换。调用之后，我们就把坐标系变换的矩阵放入了矩阵栈，后续对物体的位置描述，会通过此矩阵栈进行转换到我们的观察坐标系了。

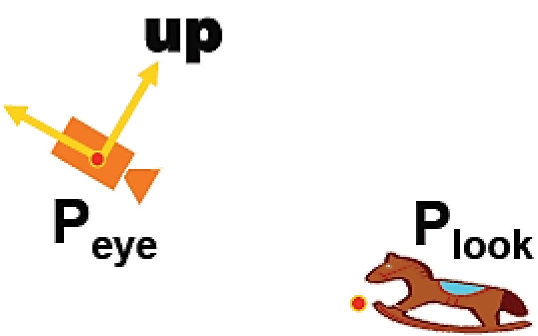
gluLookAt Illustration



Look-At Positioning

- We specify the view frame using the look-at vector **a** and the camera up vector **up**
- The vector **a** points in the negative viewing direction

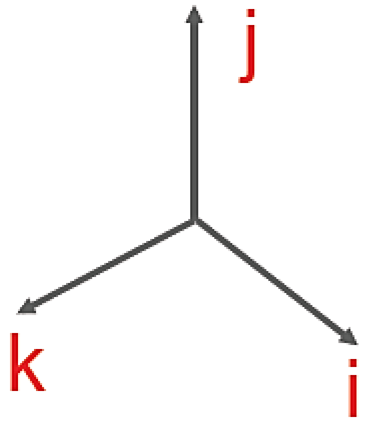
$$\mathbf{a} = P_{eye} - P_{look}$$



- In 3D, we need a third vector that is perpendicular to both **up** and **a** to specify the view frame

Where does it point to?

- The result of the cross product is a vector, not a scalar, as for the dot product
- In OpenGL, the cross product $\mathbf{a} \times \mathbf{b}$ yields a RHS vector. $\mathbf{a}$ and $\mathbf{b}$ are the thumb and index fingers, respectively

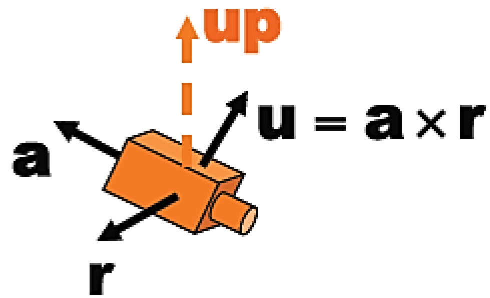


Constructing a Coordinates

- The cross product between the up and the look-at vector will get a vector that points to the right.

$$\mathbf{r} = \mathbf{up} \times \mathbf{a}$$

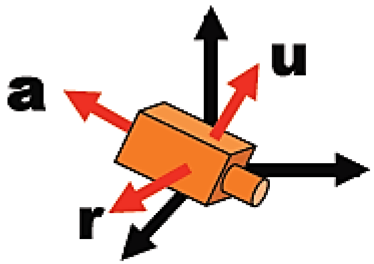
- Finally, using the vector **a** and the vector **r** we can synthesize a new vector **u** in the up direction:



Rotation

- Rotation takes the unit world frame to our desired **view reference frame:**

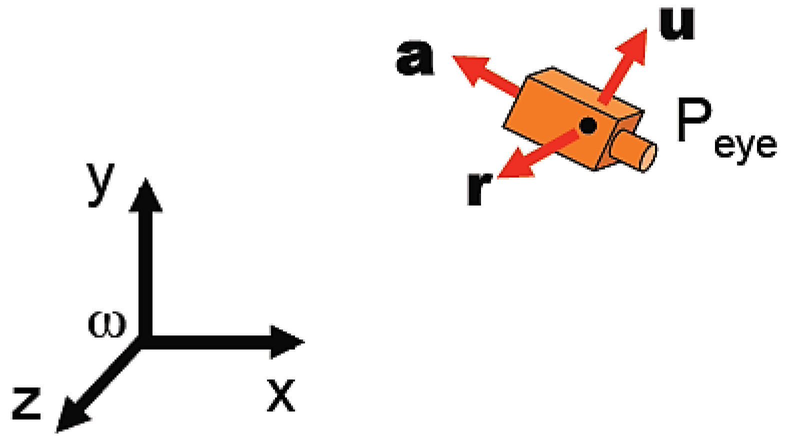
$$\begin{bmatrix} r_x & u_x & a_x & 0 \\ r_y & u_y & a_y & 0 \\ r_z & u_z & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{R}$$



Translation

- Translation to the eye point:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & eye_x \\ 0 & 1 & 0 & eye_y \\ 0 & 0 & 1 & eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Composing the Result

- The final **viewing coordinate transformation is:**

$$\mathbf{E} = \mathbf{TR} = \begin{bmatrix} 1 & 0 & 0 & eye_x \\ 0 & 1 & 0 & eye_y \\ 0 & 0 & 1 & eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_x & u_x & a_x & 0 \\ r_y & u_y & a_y & 0 \\ r_z & u_z & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The Viewing Transformation

- Transforming all points P in the world with $\mathbf{E}^{-1}$:

$$\mathbf{V} = \mathbf{R}^{-1}\mathbf{T}^{-1} = \begin{bmatrix} r_x & r_y & r_z & 0 \\ u_x & u_y & u_z & 0 \\ a_x & a_y & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Where these are normalized vectors:

$$\mathbf{a} = P_{eye} - P_{look}$$

$$\mathbf{r} = \mathbf{up} \times \mathbf{a}$$

$$\mathbf{u} = \mathbf{a} \times \mathbf{r}$$

Looking At a cube

- Setting up the OpenGL look-at viewing transformation:

```
void display(void){
        glClear(GL_COLOR_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        // Setting up the view
        gluLookAt(
                0.0, 0.0, 5.0,     // Eye is at (0,0,5)
                0.0, 0.0, 0.0,     // Center is at (0,0,0)
                0.0, 1.0, 0.);     // Up is in positive Y direction
        // Now we are using the world frame
        // Draw Object
        glColor3f (1.0, 1.0, 1.0);
        glutWireCube(1.0);
        glutSwapBuffers();
}
```

Model/View Transformation

- Combine modeling and viewing transform
  - Combine into single matrix
  - Saves computation time
    - if many points are to be transformed
  - Possible because viewing transformation directly follows modeling transformation without intermediate operations

gluLookAt() and other transformations

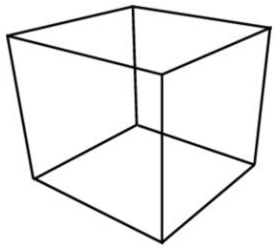
- The user can define the model-view matrix to achieve the same function
- But from the concept of the gluLookAt () as the camera position, while the other follow-up transformation as object position
- gluLookAt in the OpenGL () function is **the only specialized** for positioning the camera function

Outline

- 2D Viewing Transformation
- 3D Viewing Transformation
  - Computer view
    - Positioning the camera
    - **Projection**

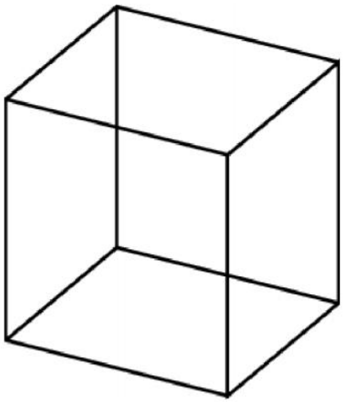
Perspective Projection

- Size varies inversely with respect to the distance form the center of projection
- Tends to look more realistic：Cannot generally measure
  - Shape
  - Object distances
  - Angles(except front faces)
  - Parallel lines appear no longer parallel



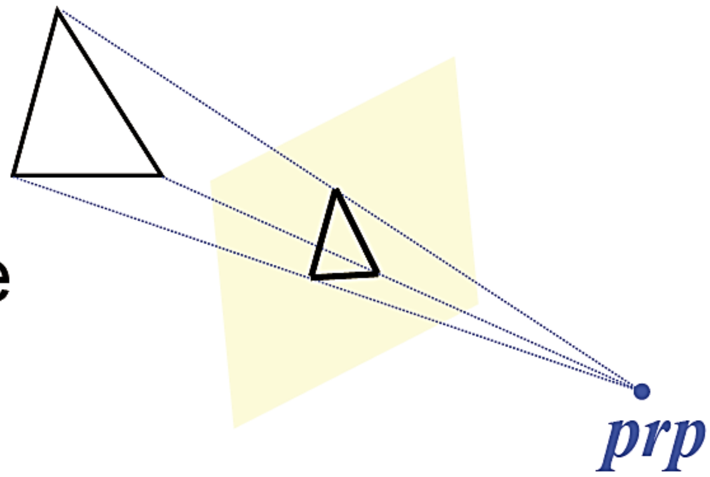
Perspective Projection

- Less realistic because perspective foreshortening is lacking
- Can however, use for exact measurements
  - Angles still only preserved for front faces
  - Parallel lines remain parallel



Perspective Projection

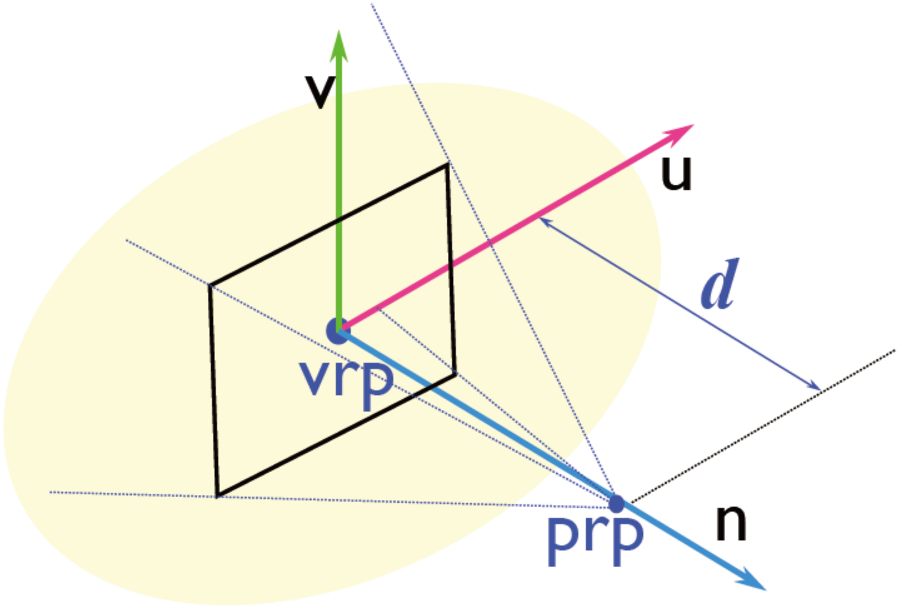
- The points are transformed to the view plane along lines that converge to a point called
  - ***projection reference point*** (*prp*) or
  - ***center of projection*** (*cop*)



- *prp* is specified in terms of the viewing coordinate system

Transformation Matrix for Perspective Projection

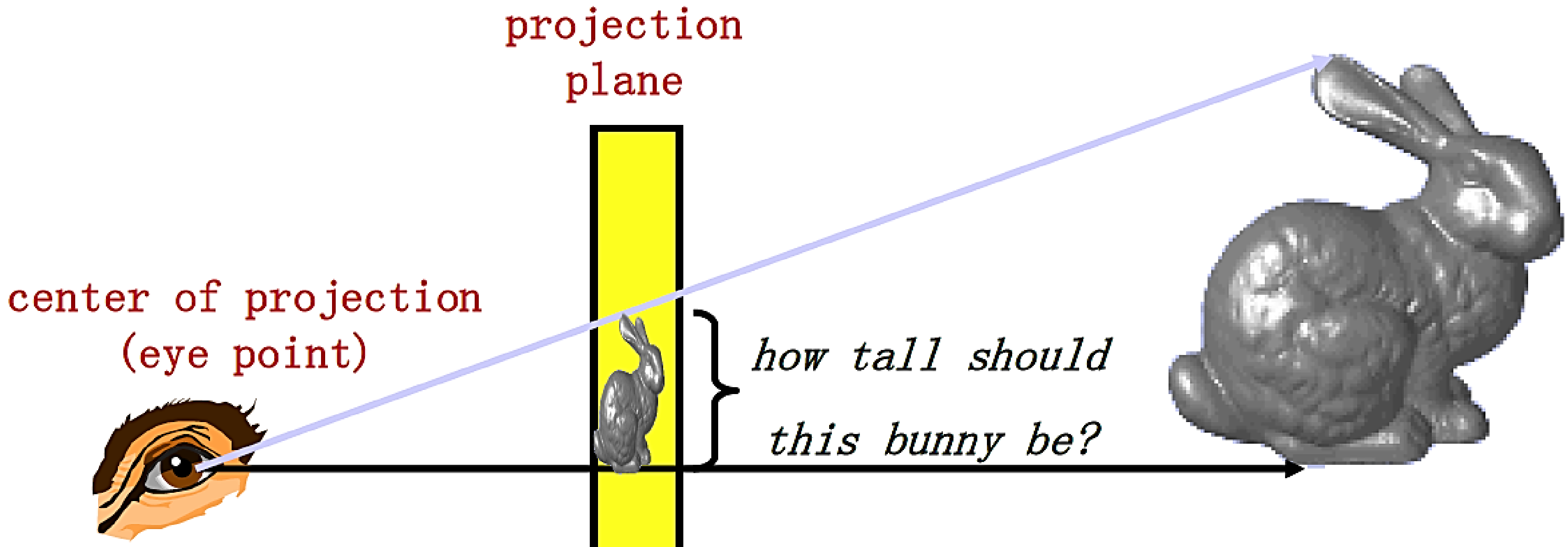
- ***prp*** is usually specified as perpendicular distance ***d*** behind the view plane



transformation matrix
for *perspective projection*

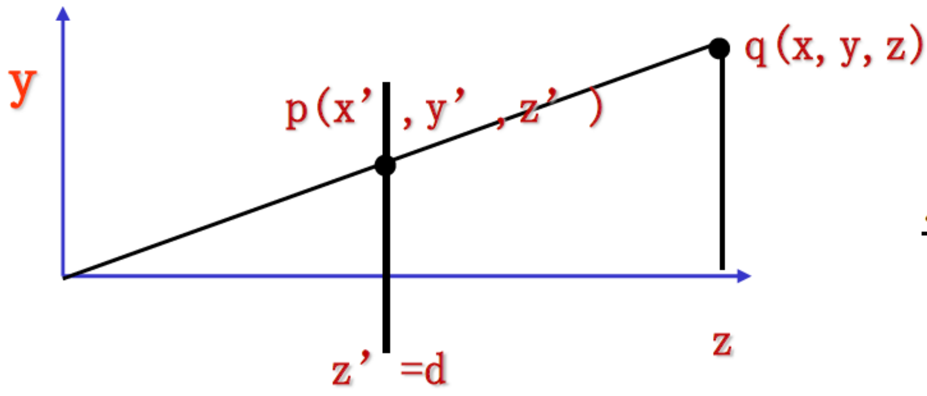
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}$$

Perspective Projection



Basic Perspective Projection

similar triangles



$$\frac{y'}{d} = \frac{y}{z} \rightarrow y' = \frac{y \cdot d}{z}$$

$$\frac{x'}{d} = \frac{x}{z} \rightarrow x' = \frac{x \cdot d}{z}$$

but $z' = d$

Given p = Mq, write out the Projection Matrix M.

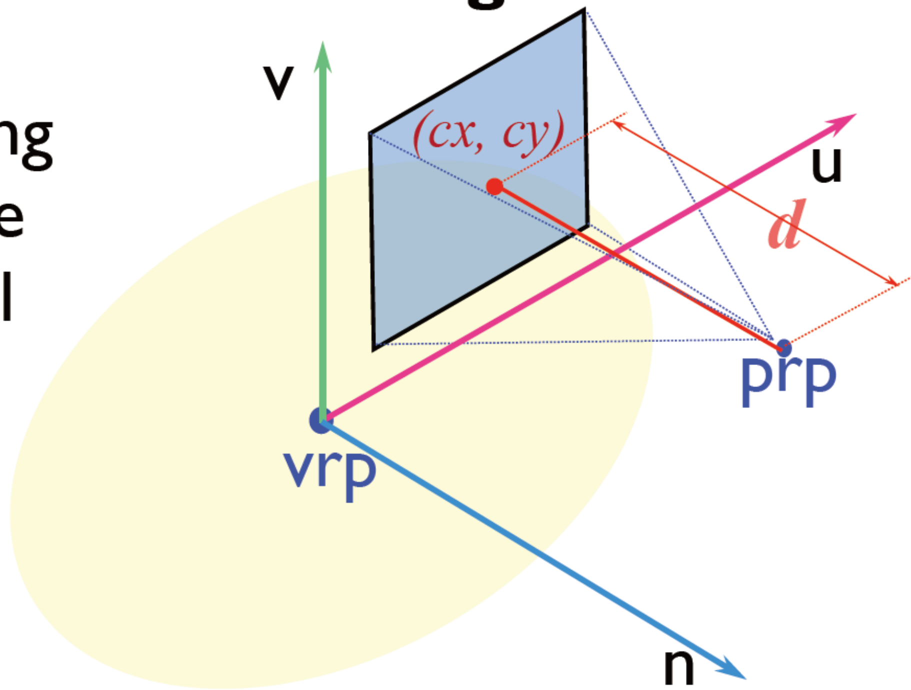
Homogeneous Coordinates

$$\mathbf{p} = \mathbf{Mq}$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \qquad \mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

View Window

- ***View window*** is a rectangle in the *view plane* specified in terms of view coordinates.
- Specify ***center (cx, cy), width*** and ***height***
- *prp* lies on the axis passing through the center of the view window and parallel to the n-axis



Perspective Projection

1. Apply the view orientation transformation
2. Apply translation, such that the center of the view window coincide with the origin
3. Apply the perspective projection matrix to project the 3D world onto the view plane

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -cx \\ 0 & 1 & 0 & -cy \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} u_x & u_y & u_z & -\vec{u} \bullet vrp \\ v_x & v_y & v_z & -\vec{v} \bullet vrp \\ n_x & n_y & n_z & -\vec{n} \bullet vrp \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. Apply 2D viewing transformations to map the view window (centered at the origin) on to the screen

Orthogonal Projection Matrix: Homogeneous coordinates

$$\mathbf{P}_\mathrm{p} = \mathbf{Mp}$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$x_\mathrm{p} = x$$

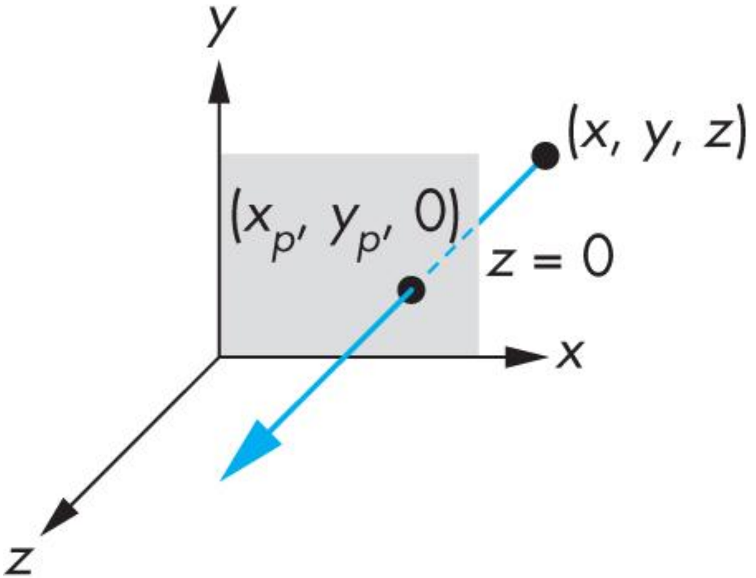
$$y_\mathrm{p} = y$$

$$z_\mathrm{p} = 0$$

$$w_\mathrm{p} = 1$$

在实际应用中可以令$\mathrm{M} = \mathrm{I}$, 然后把对角线第三个元素置为零。

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



Orthogonal Projection

1. Apply the world to view transformation
2. Apply the parallel projection matrix to project the 3D world onto the view plane

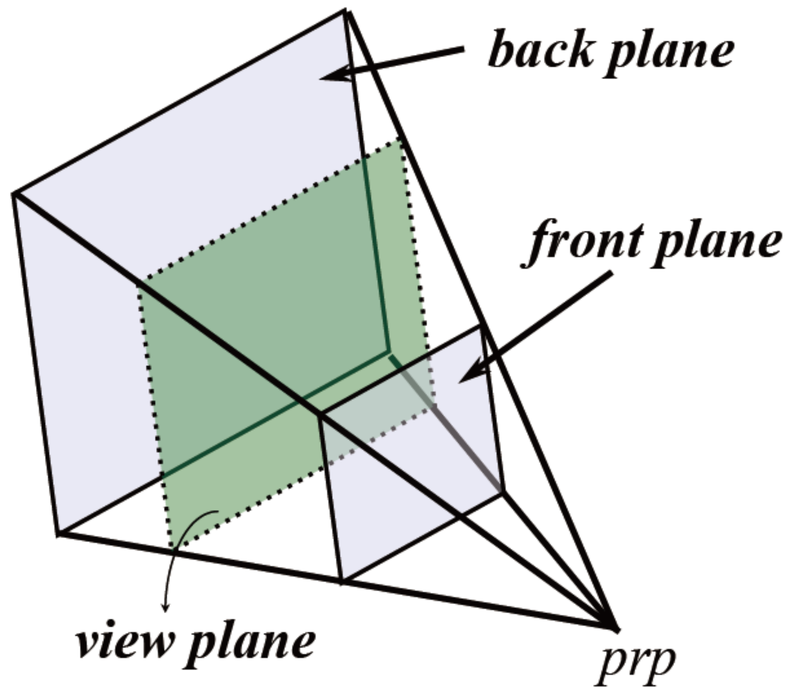
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} u_x & u_y & u_z & -\vec{u} \bullet vrp \\ v_x & v_y & v_z & -\vec{v} \bullet vrp \\ n_x & n_y & n_z & -\vec{n} \bullet vrp \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Apply 2D viewing transformations to map the view window on to the screen

View Volume & Clipping

- For perspective projection the ***view volume*** is a semi-infinite pyramid with apex (顶点) at ***prp*** and edges passing through the corners of the view window

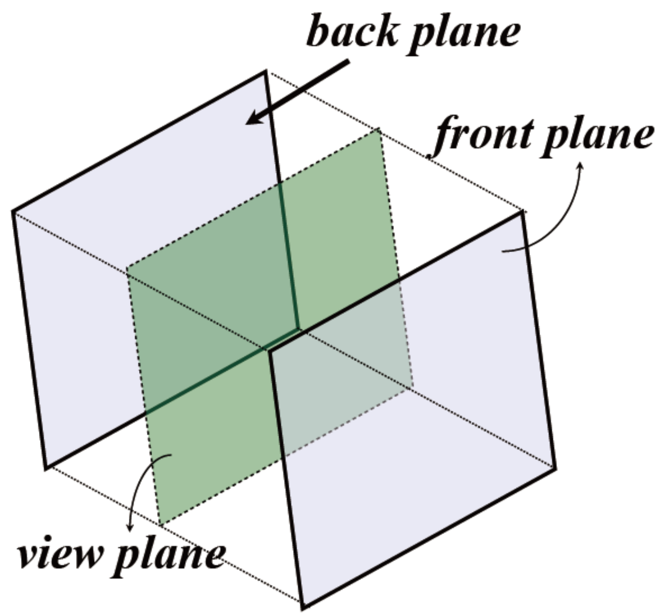
- For efficiency, view volume is made finite by specifying the front and back clipping plane specified as distance from the view plane



View Volume & Clipping

- For parallel projection the ***view volume*** is an infinite parallelepiped (平行六面体) with sides parallel to the direction of projection

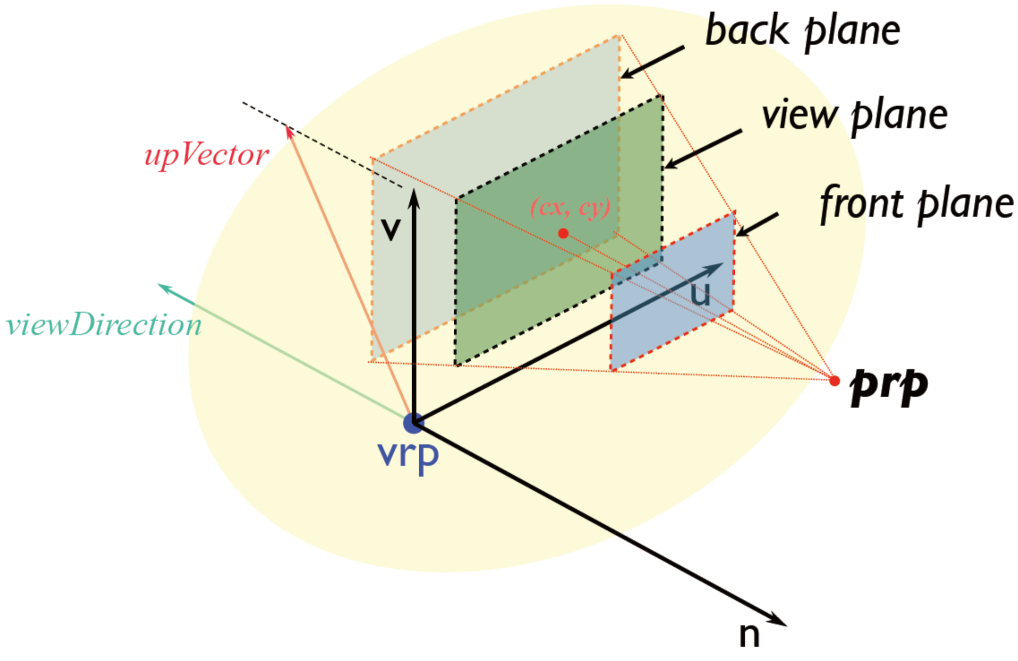
- View volume is made finite by specifying the front and back clipping plane specified as distance from the view plane



- Clipping is done in 3D by clipping the world against the front clip plane, back clip plane and the four side planes

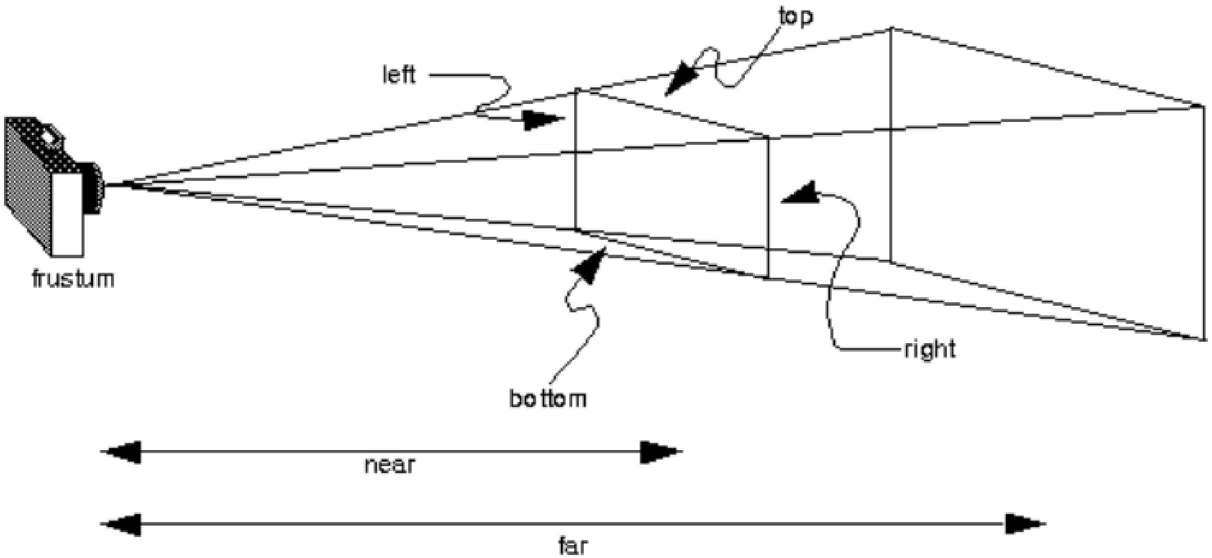
The Complete View Specification

- **Specification in world coordinates**
  - position of viewing (***vrp***), direction of viewing(***-n***),
  - up direction for viewing (***upVector***)
- **Specification in view coordinates**
  - view window : center (***cx, cy***), ***width*** and ***height***,
  - ***prp*** : distance from the view plane,
  - front clipping plane : distance from view plane
  - back clipping plane : distance from view plane



Perspective in OpenGL

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



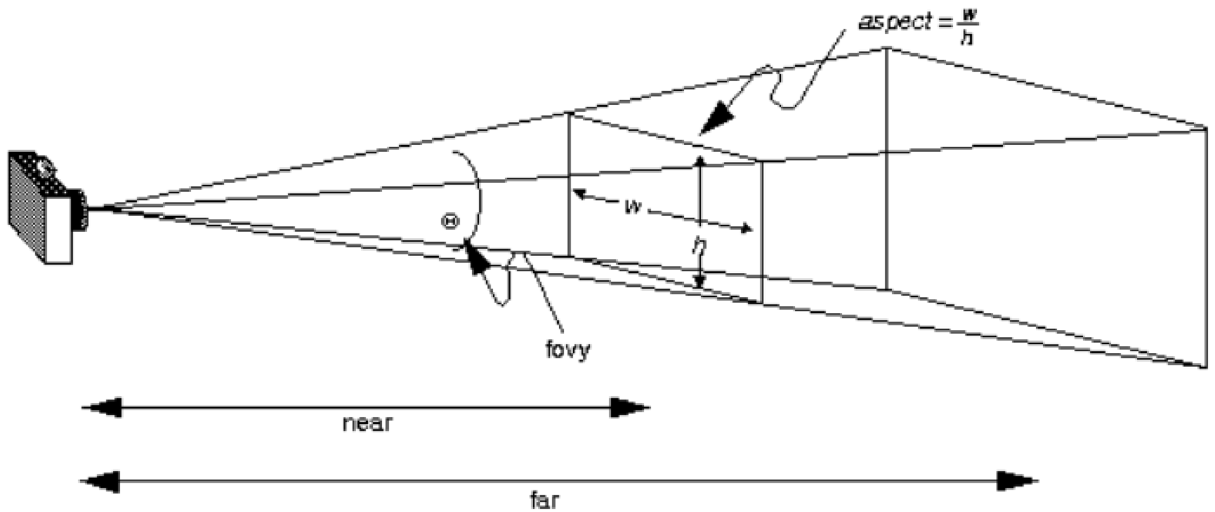
glMatrixMode(GL_PROJECTION);

glLoadIdentity( );

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

Perspective in OpenGL

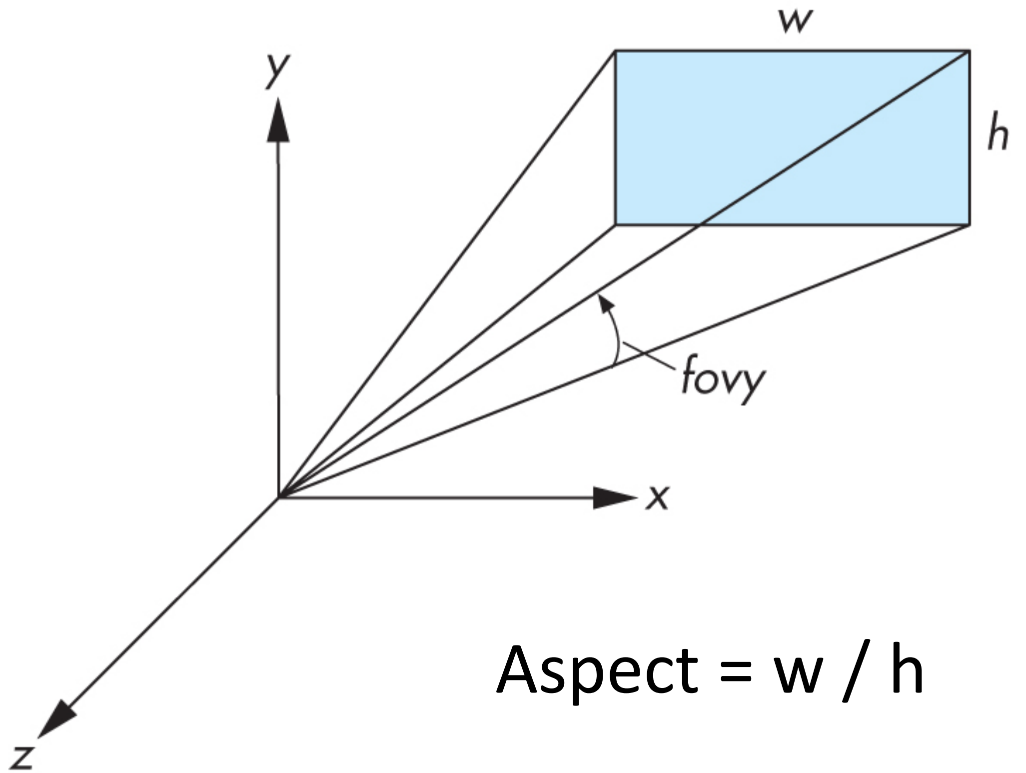
gluPerspective(fovy, aspect, near, far);



FOV is the angle between the top and bottom planes

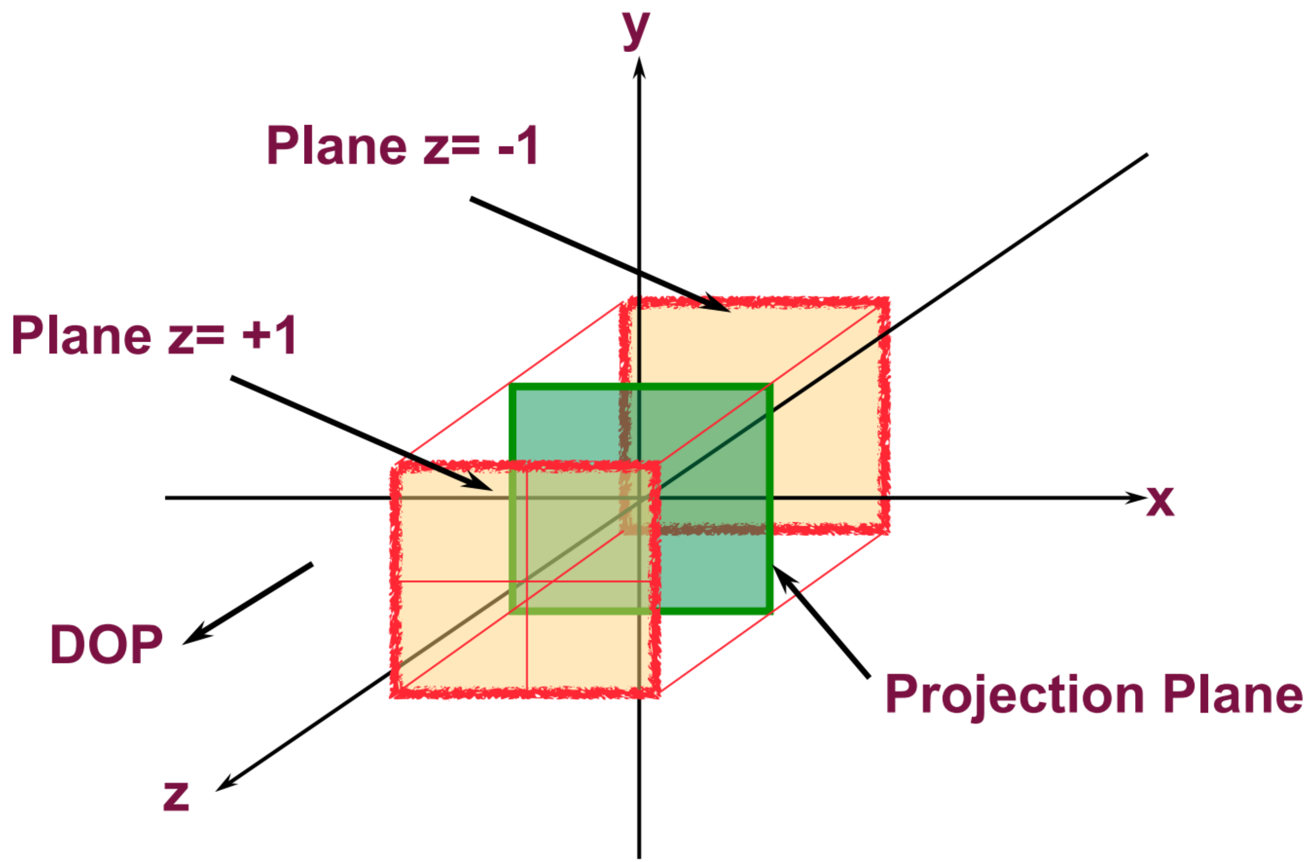
Field of application

- Application of glFrustum sometimes difficult to get the desired results
- GluPerspective (fovy, aspect, near, far) can provide a better results



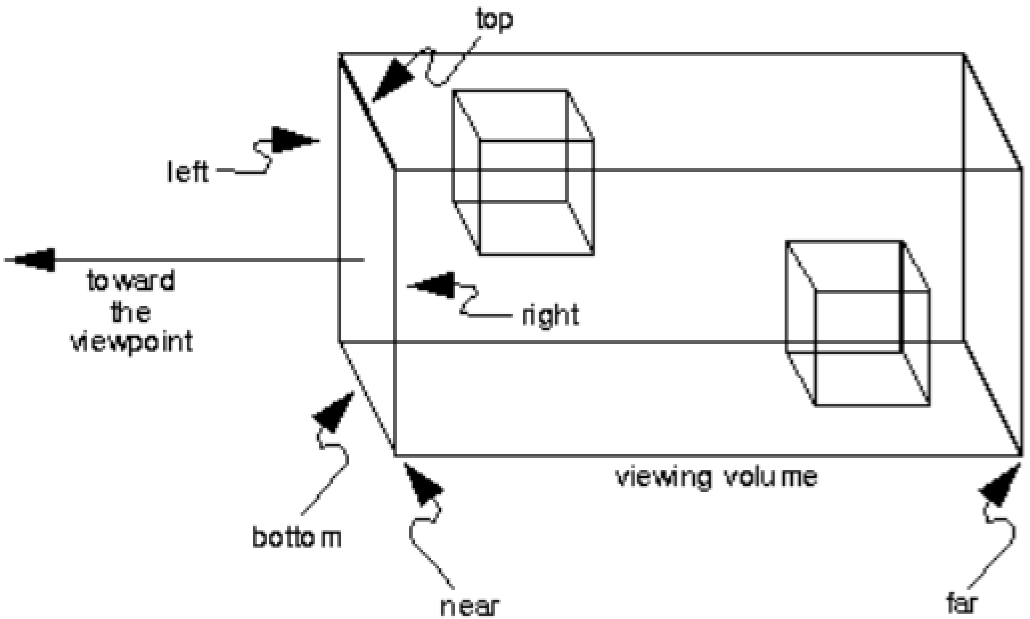
Aspect = w / h

Orthographic Default View Volume



Orthogonal view in OpenGL

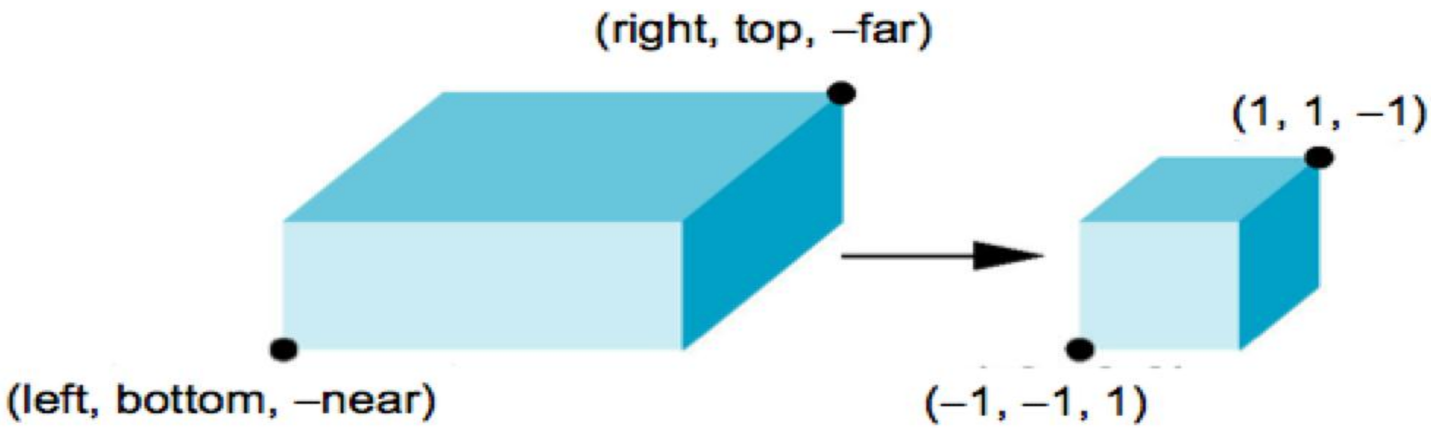
glOrtho(left, right, bottom, top, near, far);

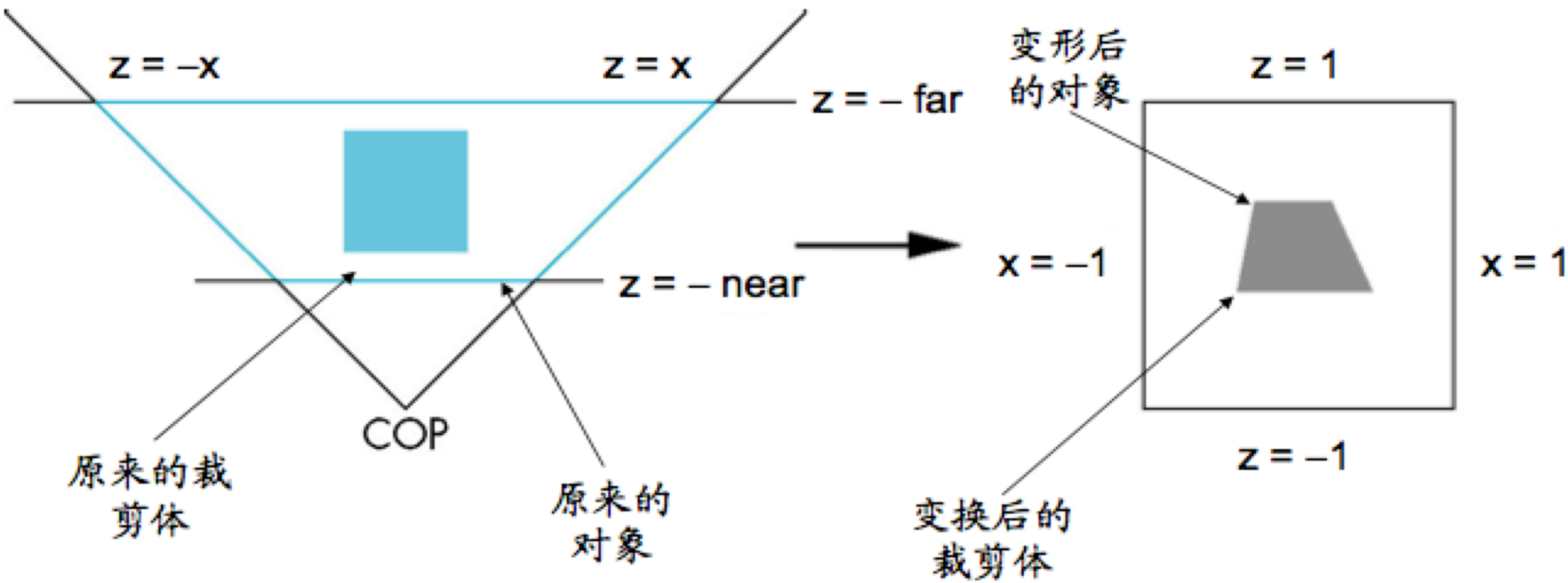


Normalization

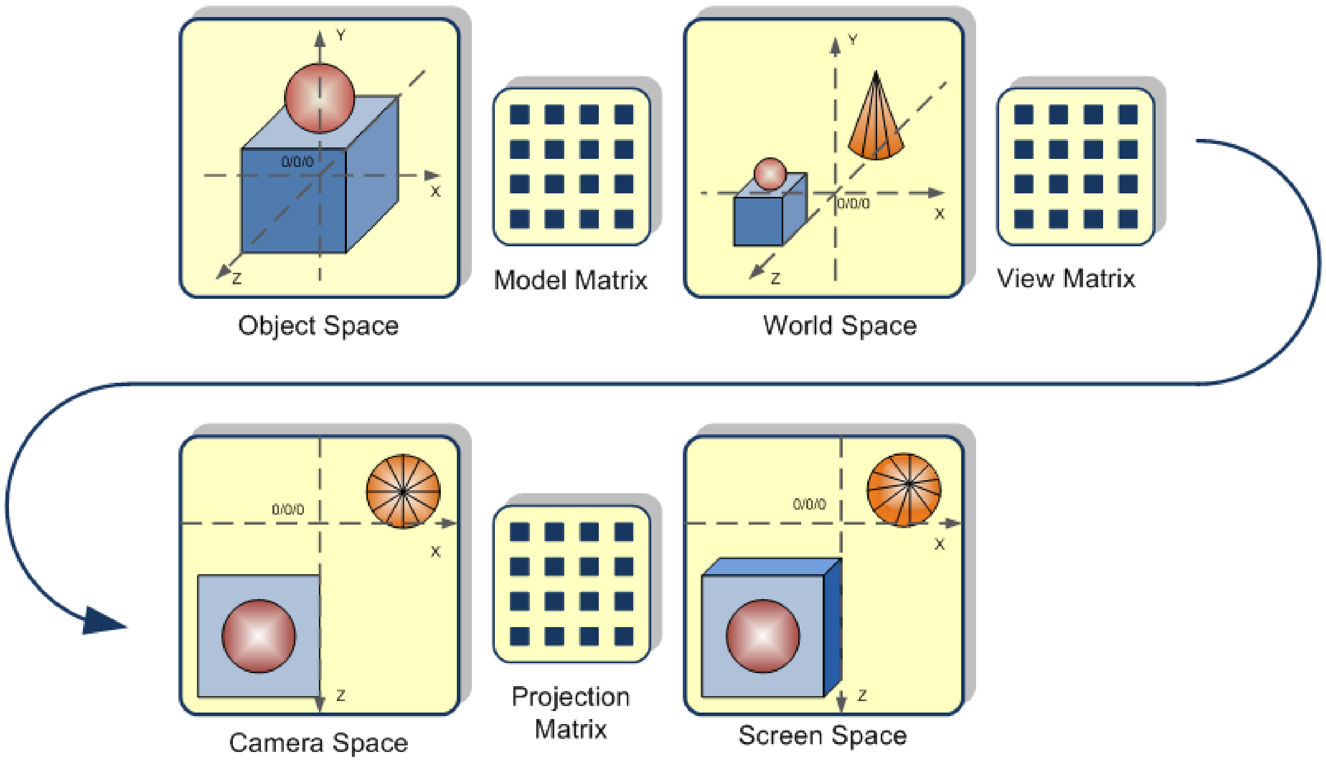
- Normalization allows for a single pipeline for both perspective and orthogonal viewing.
- It simplifies clipping.
- Projection to the image plane is simple (discard z).
- z is retained for z-buffering (visible surface determination)

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





Transformation Pipeline



1. Vertices of the Object to draw are in **Object space** (as modelled in your 3D Modeller)

2. … get transformed into World space by multiplying it with the **Model Matrix**

3. Vertices are now in **World space** (used to position the all the objects in your scene)

4. … get transformed into Camera space by multiplying it with the **View Matrix**

5. Vertices are now in **View Space** – think of it as if you were looking at the scene throught “the camera”

6. … get transformed into Screen space by multiplying it with the **Projection Matrix**

7. Vertex is now in **Screen Space** – This is actually what you see on your Display.