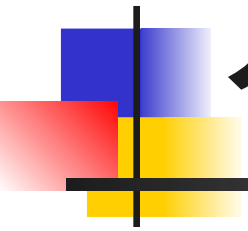


几何变换与投影

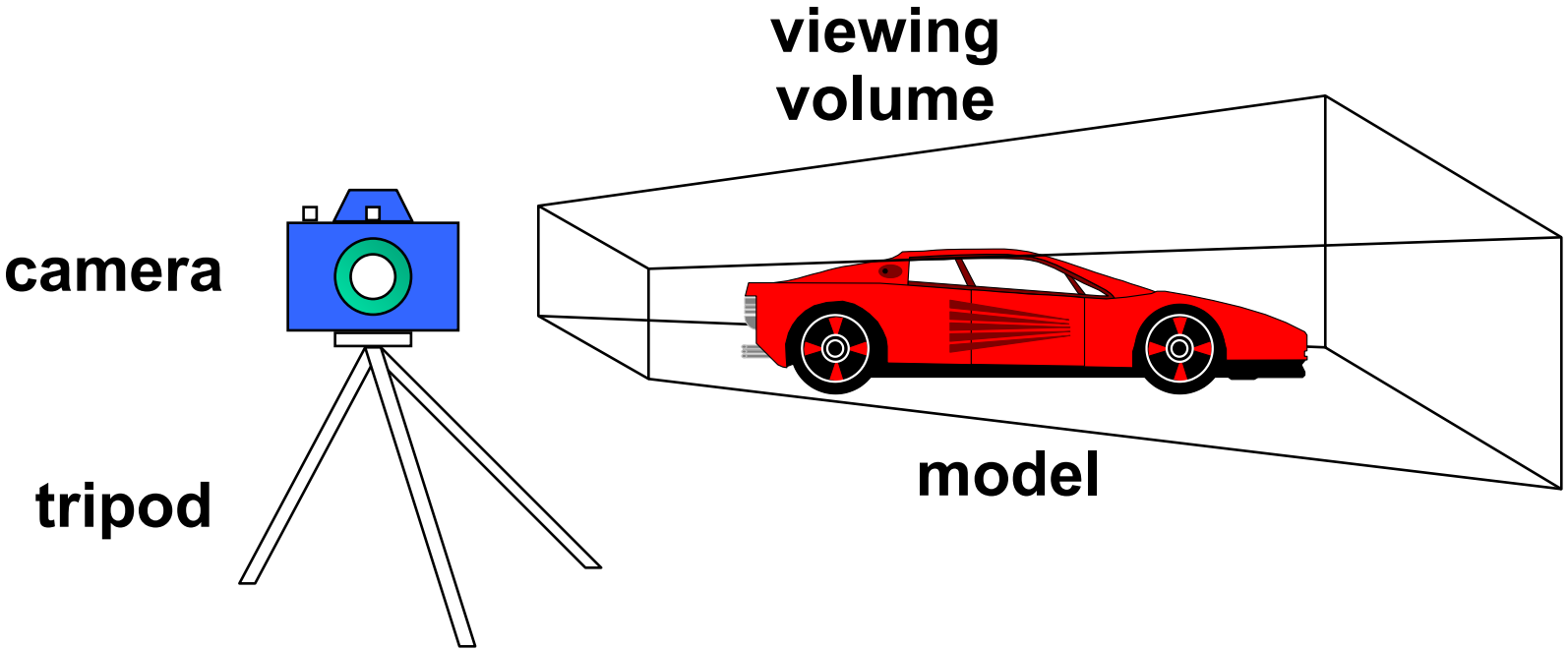
Math.

OpenGL

Programming

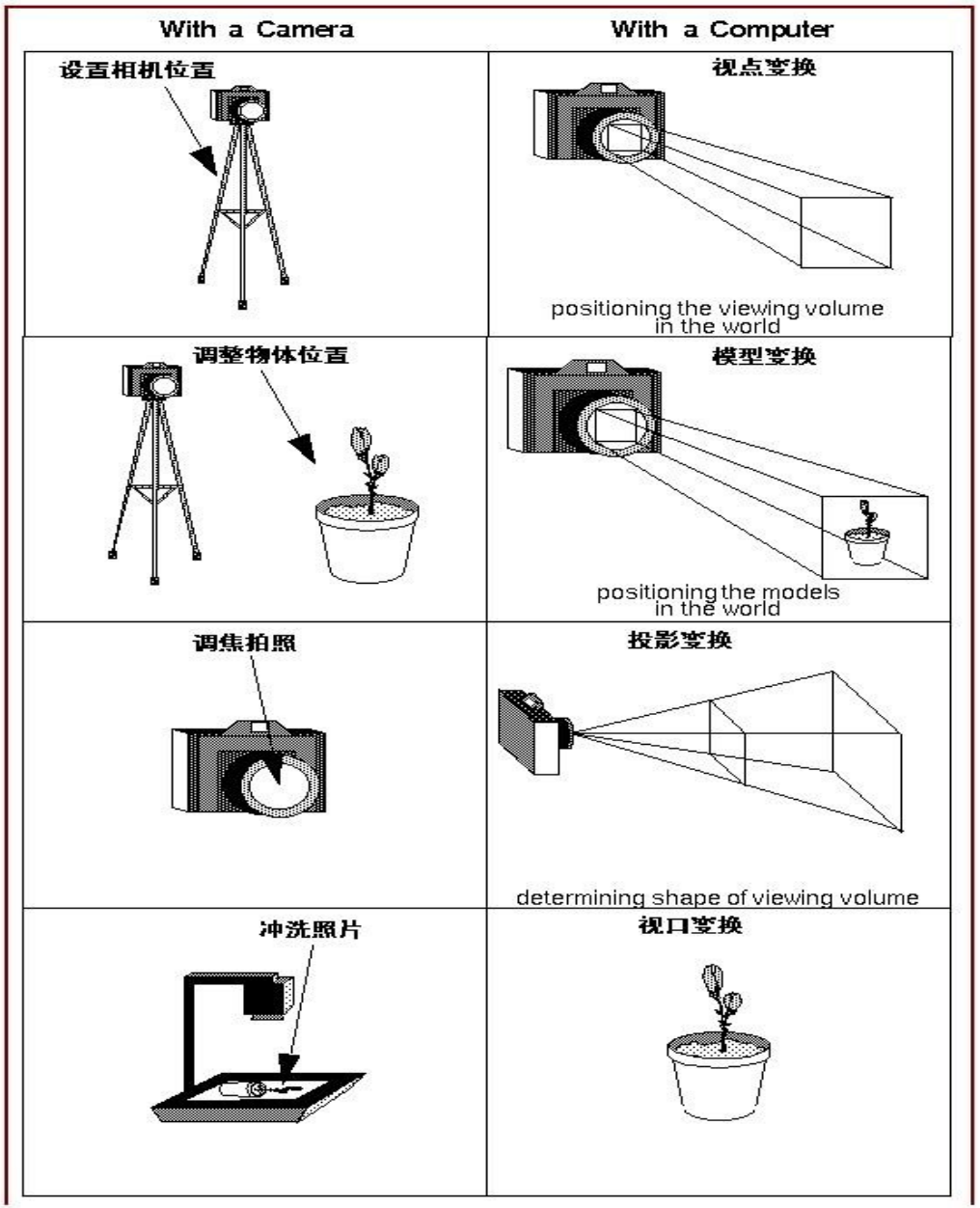
相机

- *3D is just like taking a photograph!*



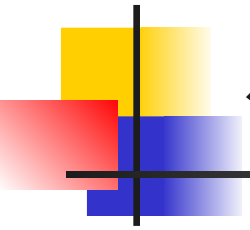
Camera Analogy and Transformations

- *投影变换*
  - 相机的镜头
- *视点变换*
  - 三脚架的位置和方向
- *模型变换*
  - 移动模型
- *视口变换*
  - 照片大小


With a Camera
With a Computer
设置相机位置
视点变换
positioning the viewing volume in the world
调整物体位置
模型变换
positioning the models in the world
调焦拍照
投影变换
determining shape of viewing volume
冲洗照片
视口变换

坐标系统与变换

- *Steps in Forming an Image*
  - specify geometry (world coordinates)
  - specify camera (camera coordinates)
  - project (window coordinates)
  - map to viewport (screen coordinates)
- *Each step uses transformations*
- *Every transformation is equivalent to a change in coordinate systems*



仿射变换 Affine Transformations

- *Want transformations which preserve geometry*
  - lines, polygons, quadrics
- *Affine = line preserving*
  - Rotation, translation, scaling
  - Projection
  - Concatenation (composition)

齐次坐标 Homogeneous Coordinates

- each vertex is a column vector

$$\vec{v} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- w is usually 1.0
- all operations are matrix multiplications
- directions (directed line segments) can be represented with w = 0.0

Homogeneous Coordinates

- *A vertex is transformed by 4 x 4 matrices*
  - all affine operations are matrix multiplications
  - all matrices are stored column-major in OpenGL
  - matrices are always post-multiplied
  - product of matrix and vector is $v' = \mathbf{M}\vec{v}$

$$\mathbf{M} = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$$

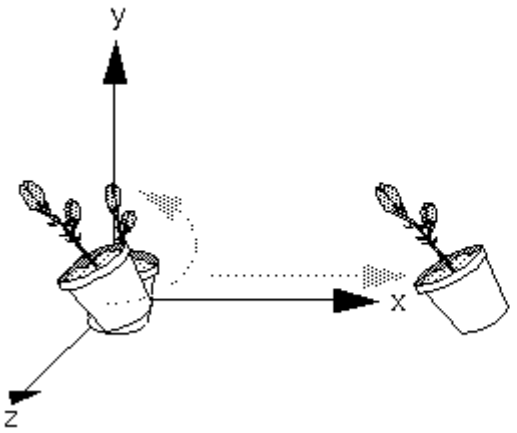
Compositing Modeling Transformations

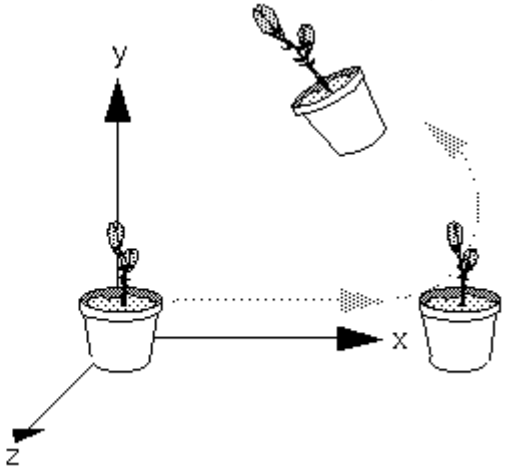
- *Combine transformations*
- *v'=NMLv=N(M(L(v)))*

```
Transform(N);
Transform(M);
Transform(L);
Draw_Point(v);
```

Thinking about Transformations

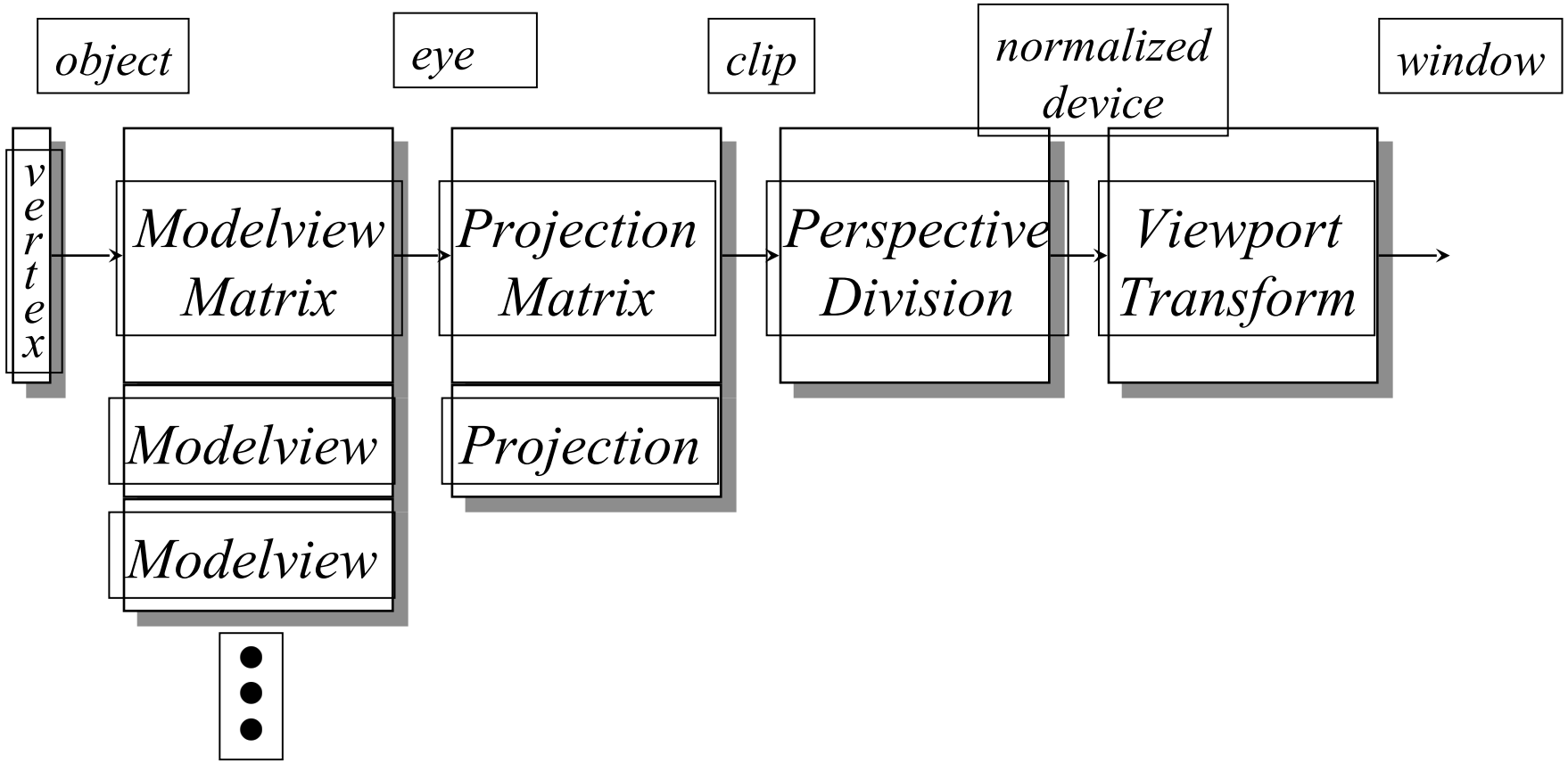
- ***Figure :*** *Rotating First or Translating First*

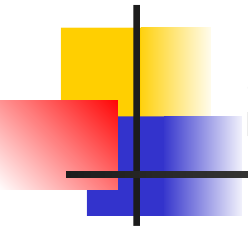





```
glMultMatrixf(T);       /* apply translation T */
glMultMatrixf(R);       /* apply Rotation R */
Draw_Objects(v);        /* draw transformed vertex v */
```

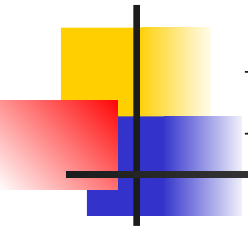
Transformation Pipeline in OpenGL





Specifying Transformations

- *Programmer has two styles of specifying transformations*
  - specify matrices (`glLoadMatrix, glMultMatrix`)
  - specify operation (`glRotate, glOrtho`)
  - Programmer does not have to remember the exact matrices



Matrix Operations

- *Specify Current Matrix*
  - **glMatrixMode(*GL_MODELVIEW* or *GL_PROJECTION*)**
- *Other Matrix*
  - **glLoadIdentity()**

Modeling Transformations

- *Move object*

  glTranslate{fd}(x, y, z)

- *Rotate object around arbitrary axis*

  glRotate{fd}(angle, x, y, z)

  - angle is in degrees

- *Dilate (stretch or shrink) or mirror object*

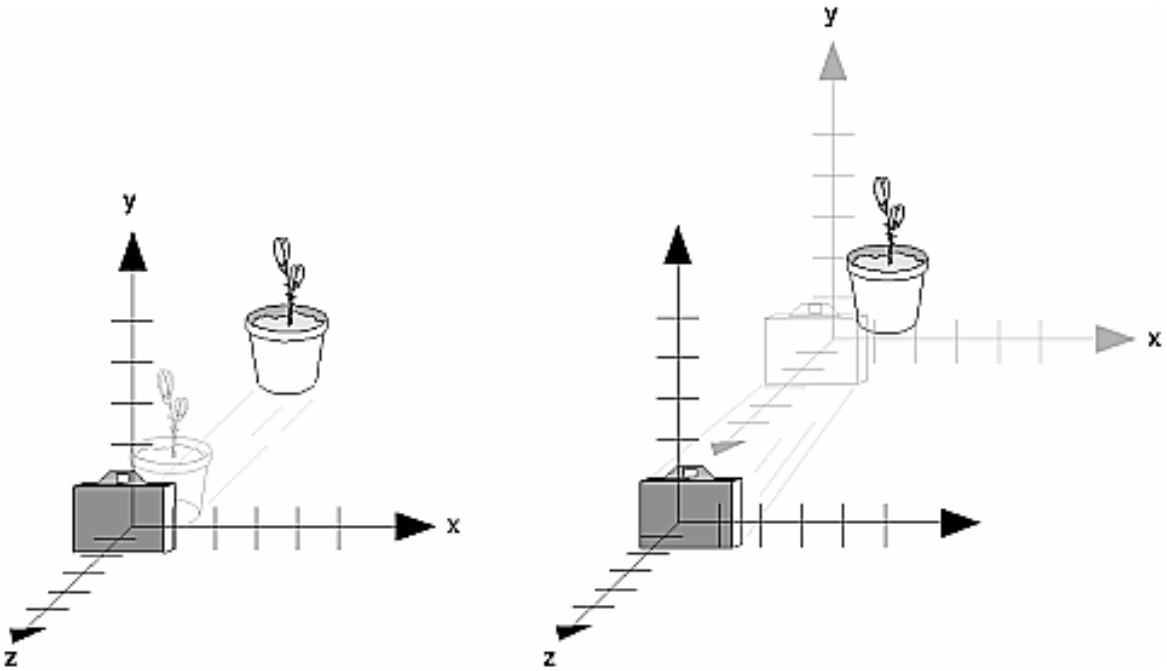
  glScale{fd}(x, y, z)

  [glMultMatrix();]

  See Demo.

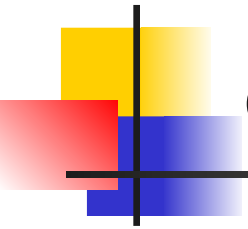
Viewing Transformations

- *glTranslate(); glRotate();*
  - e.g. glTraslatef(0.0,0.0,-5.0);



Viewing Transformations

- *Position the camera/eye in the scene*
  - place the tripod down; aim camera
- *To "fly through" a scene*
  - change viewing transformation and redraw scene
- ***gluLookAt(eye_x, eye_y, eye_z, aim_x, aim_y, aim_z, up_x, up_y, up_z)***
  - up vector determines unique orientation

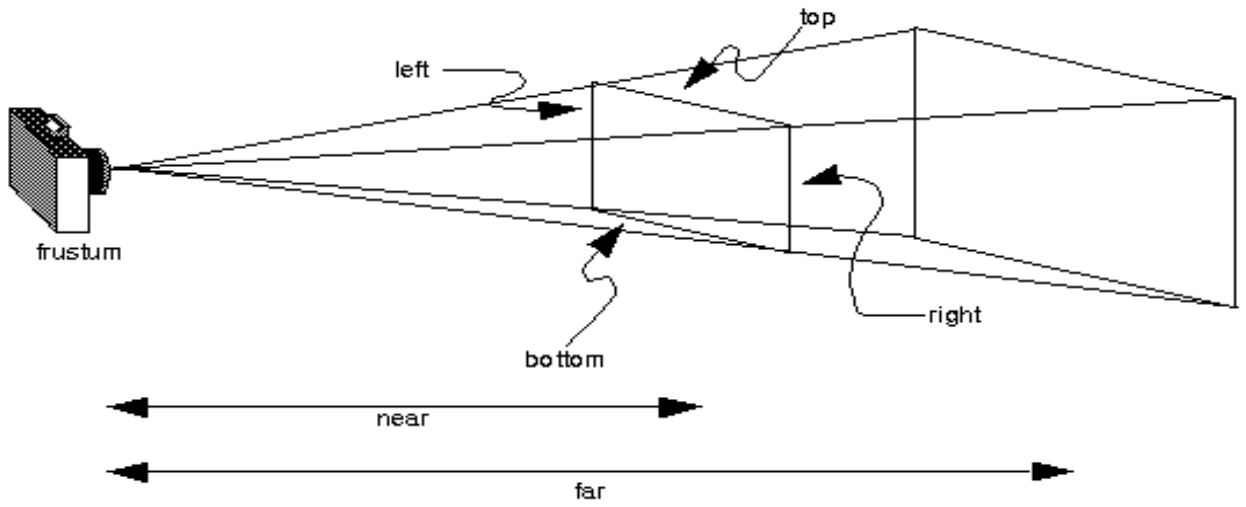


Connection: Viewing and Modeling

- *Moving camera is equivalent to moving every object in the world towards a stationary camera*
- *Viewing transformations are equivalent to several modeling transformations*

Projection Transformation

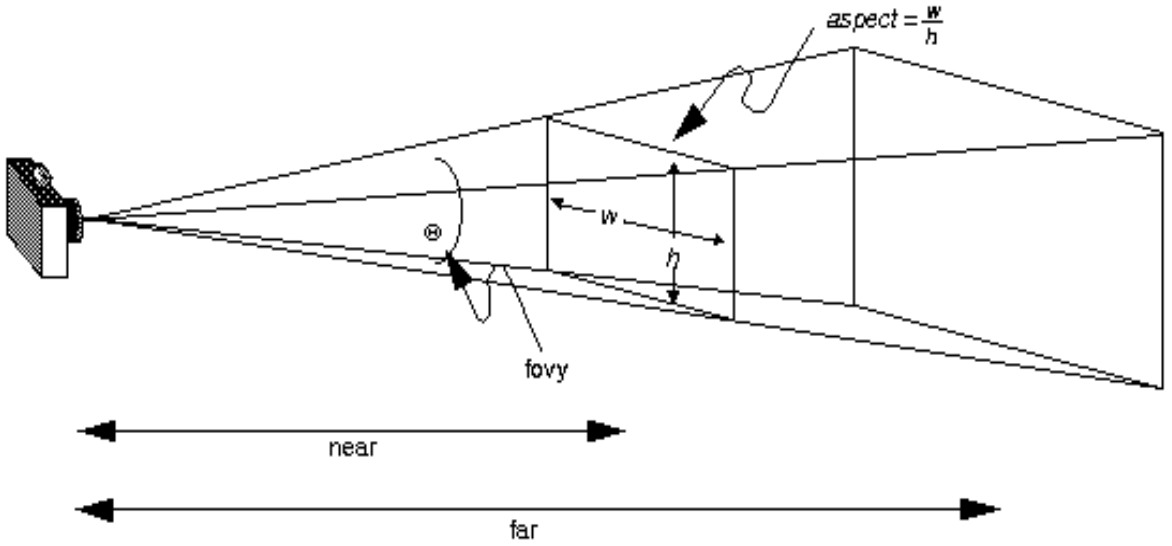
- *Shape of viewing frustum*
- *Perspective projection*
  - **glFrustum(left, right, bottom, top, zNear, zFar)**
  - **(left,bottom,-near), (ringht,top,-near)**



Projection Transformation

- *Shape of viewing frustum*
- *Perspective projection*

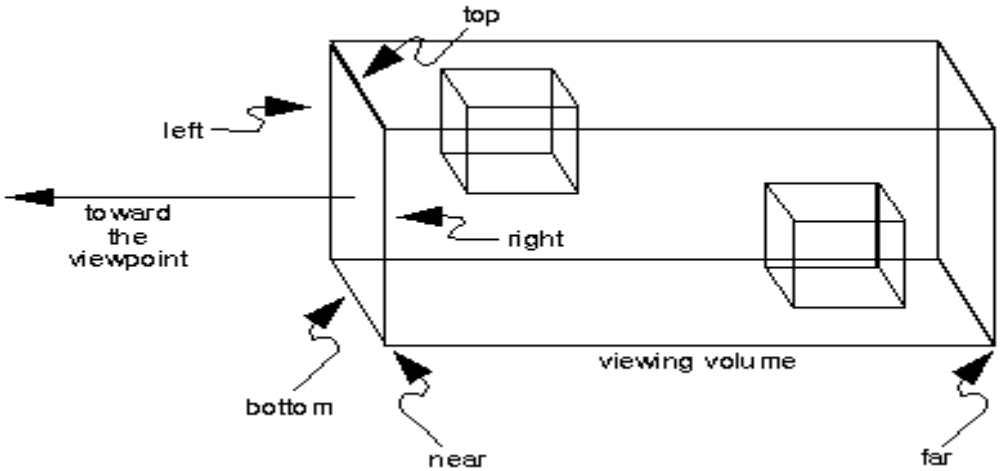
```
gluPerspective( fovy, aspect,
  zNear, zFar )
```



Projection Transformation

- *Orthographic parallel projection*

```
glOrtho( left, right,
  bottom, top, zNear, zFar )
```




```
gluOrtho2D( left, right,
  bottom, top )
```

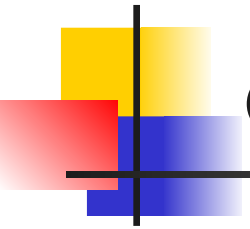
- calls `glOrtho` with z values near zero

使用投影变换

- *Typical use (orthographic projection)*

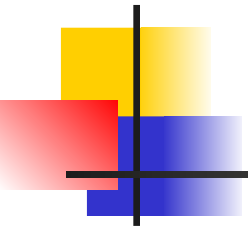
```
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
glOrtho( left, right, bottom, top,
  zNear, zFar );

See Demo.
```



Common Transformation Usage

- *Usually called when window resized*
- *Registered as callback for* `glutReshapeFunc(resize)`



`resize()`: Perspective & LookAt

```
void resize( int w, int h )
{
   glViewport( 0, 0, (GLsizei) w, (GLsizei) h );
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
   gluPerspective( 65.0, (GLfloat) w / h,
                   1.0, 100.0 );
   glMatrixMode( GL_MODELVIEW );
   glLoadIdentity();
   gluLookAt( 0.0, 0.0, 5.0,
              0.0, 0.0, 0.0,
              0.0, 1.0, 0.0 );
}
```

`resize()`: Perspective & Translate

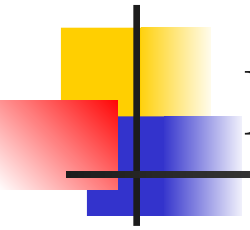
- *Same effect as previous LookAt()*

```
void resize( int w, int h )
{
   glViewport( 0, 0, (GLsizei) w, (GLsizei)
h );
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
   gluPerspective( 65.0, (GLfloat) w/h,
                   1.0, 100.0 );
   glMatrixMode( GL_MODELVIEW );
   glLoadIdentity();
   glTranslatef( 0.0, 0.0, -5.0 );
}
```

Viewport Transformation

`glViewport( x, y, width, height )`

- (*x, y*) specifies the lower left corner of the viewport,
- *(width , height)* are the size of the viewport rectangle.
  - usually same as window size
  - viewport aspect ratio should be same as projection transformation or resulting image may be distorted



Matrix Stacks

- *glPushMatrix();*
  - copies the current matrix and adds the copy to the top of the stack
- *glPopMatrix();*
  - discards the top matrix on the stack
- *glPushMatrix() means "remember where you are" and glPopMatrix() means "go back to where you were."*

Modelview & Projection Matrix Stack

- *glMatrixMode(GL_MODELVIEW);*
  - for constructing hierarchical models
- *glMatrixMode(GL_PROJECTION);*
  - No compose projection
  - only two levels deep
  - e.g. For displaying text

```
glMatrixMode(GL_PROJECTION); glPushMatrix();
/*save the current projection*/
glLoadIdentity();
glOrtho(...);                /*set up for displaying help*/
display_the_help();
glPopMatrix();
```