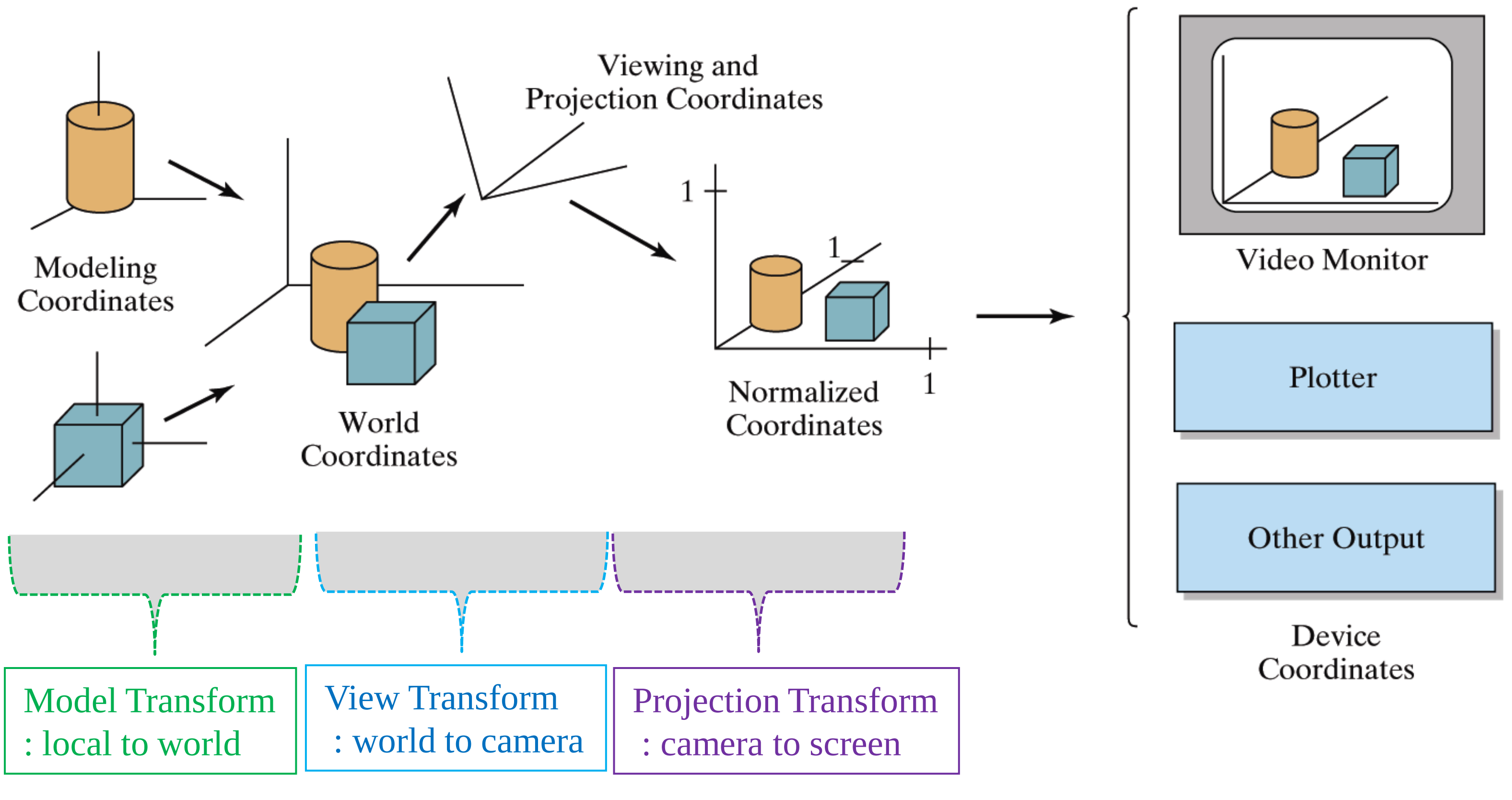
Model View Matrix and its implementation

Sang Il Park

Department of Software

OpenGL Geometric Transformations

- **Consecutive Transformations in OpenGL Pipeline**



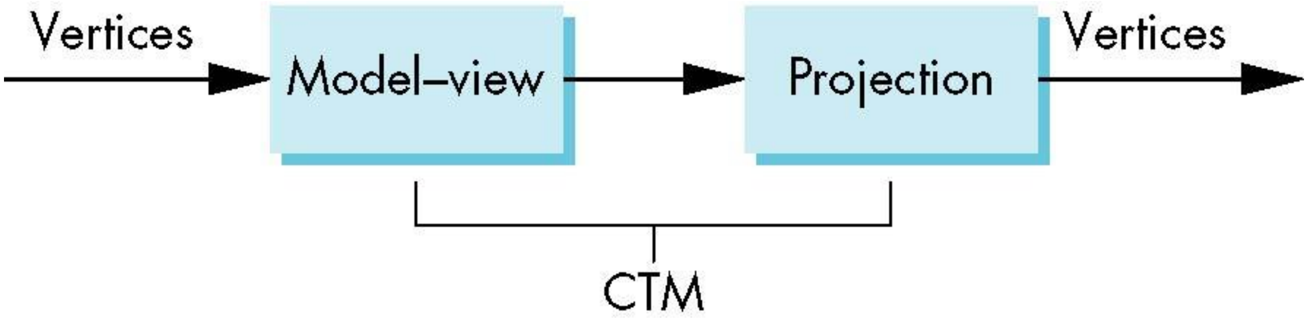
OLD OpenGL Matrices

- Two types of predefined transformations (matrices)
  - Model-View (**`GL_MODELVIEW`**) : model+view
  - Projection (**`GL_PROJECTION`**)

- Single set of functions for manipulation
- Select which to manipulated by
  - **`glMatrixMode(GL_MODELVIEW);`**
  - **`glMatrixMode(GL_PROJECTION);`**

Current Transform Matrix (CTM) in OLD OpenGL

- OpenGL had a model-view and a projection matrix in the pipeline which were concatenated together to form the CTM
- We will emulate this process

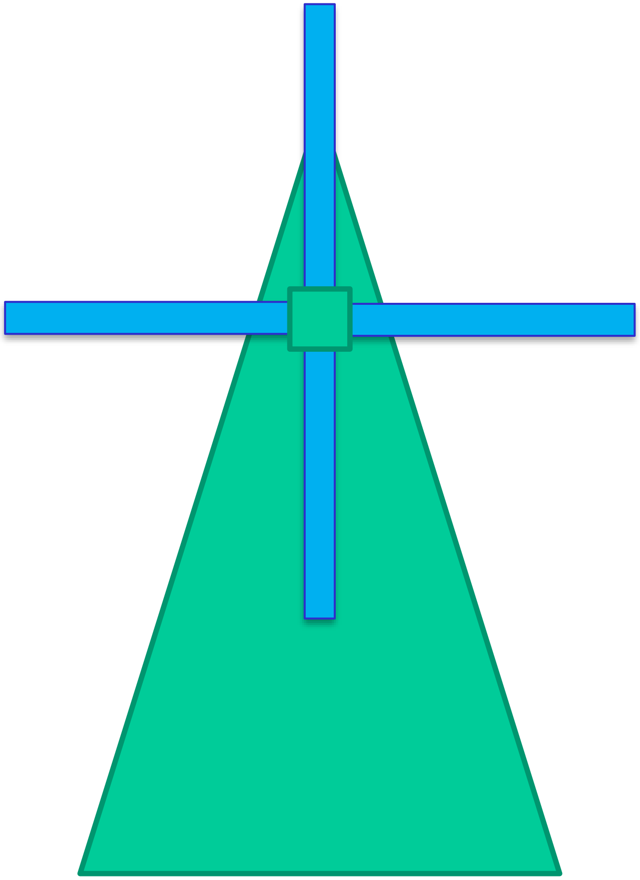


CTM: Current Transform Matrix

OLD OpenGL
Geometric Transformation functions

- Basic Transpormation:
  - `glLoadIdentity();`
  - `glTranslatef(tx, ty, tz);`
  - `glRotatef(theta, vx, vy, vz);` **`angle-axis`**
    - `(vx, vy, vz)` is automatically normalized
  - `glScalef(sx, sy, sz);`
  - `glLoadMatrixf(Glfloat elems[16]);`
- Multiplication
  - `glMultMatrixf(Glfloat elems[16]);`
  - The current matrix is **postmultiplied** by the matrix
  - **Column major**

연습: 바람개비(풍차)만들기

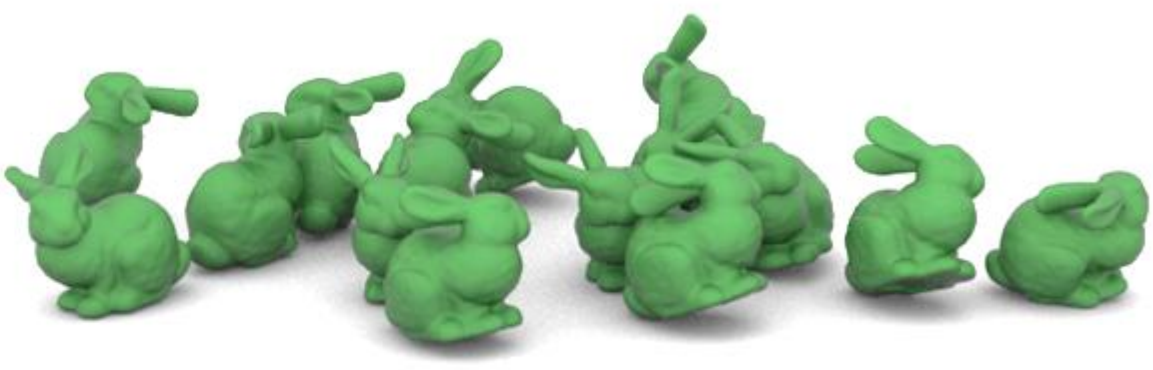


Instance Transformation

- Often we need several instances of an object
  - Wheels of a car
  - Arms or legs of a figure
  - Chess pieces

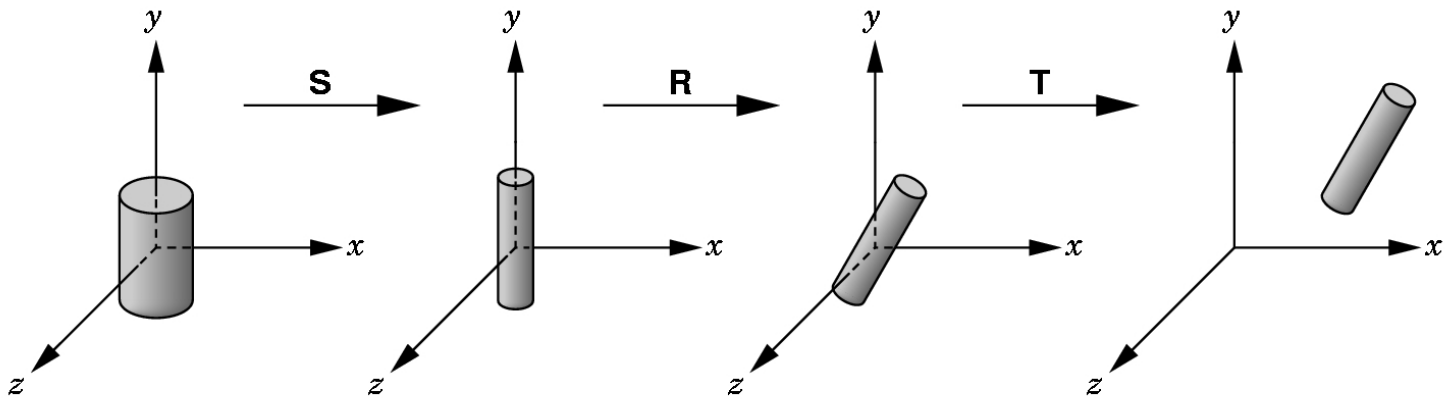
Instance Transformation

- Instances can be shared across space or time
- Write a function that renders the object in “standard” configuration
- Apply transformations to different instances
- Typical order: *scaling → rotation → translation*

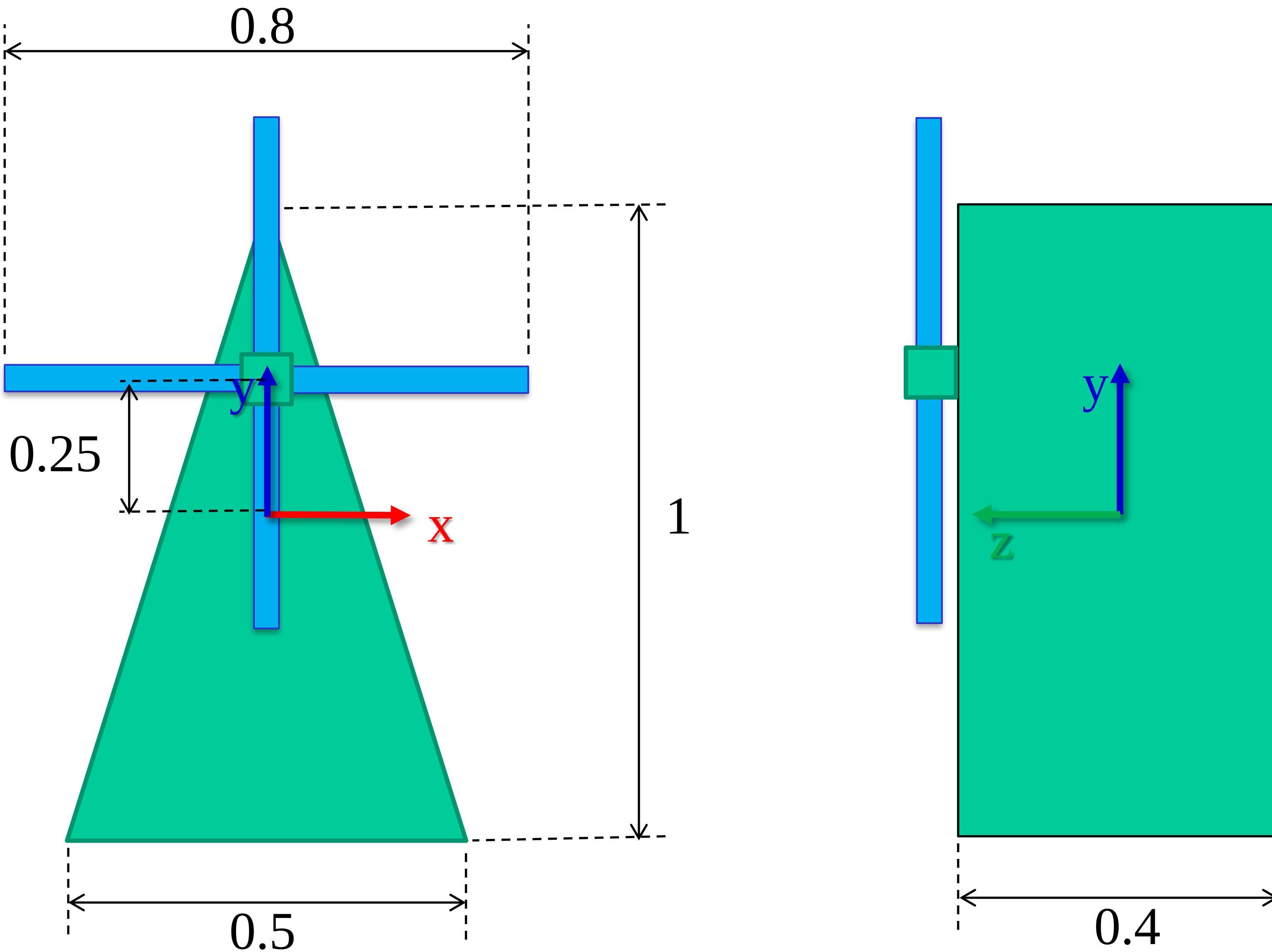


Instance Transformation

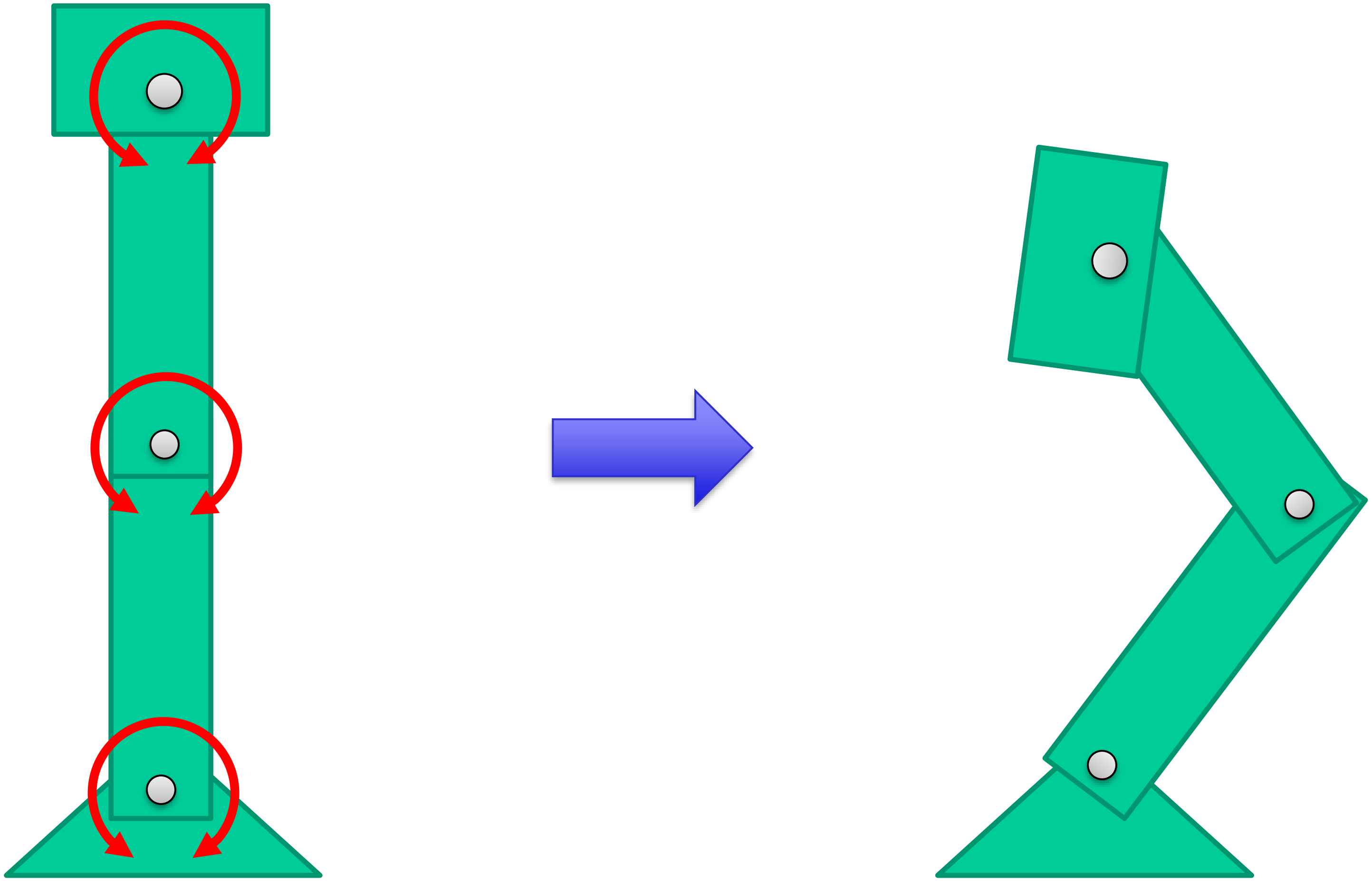
- Instances can be shared across space or time
- Write a function that renders the object in “standard” configuration
- Apply transformations to different instances
- Typical order: *scaling → rotation → translation*



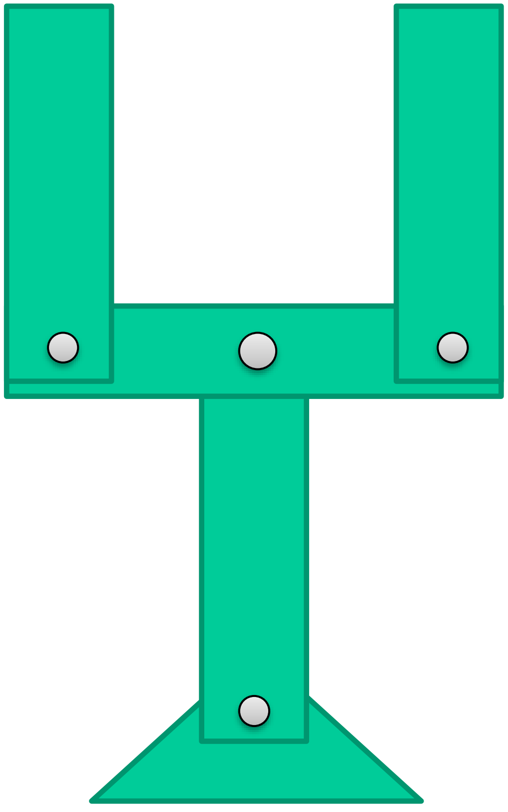
구체적인 계획



로보트 팔 만들기

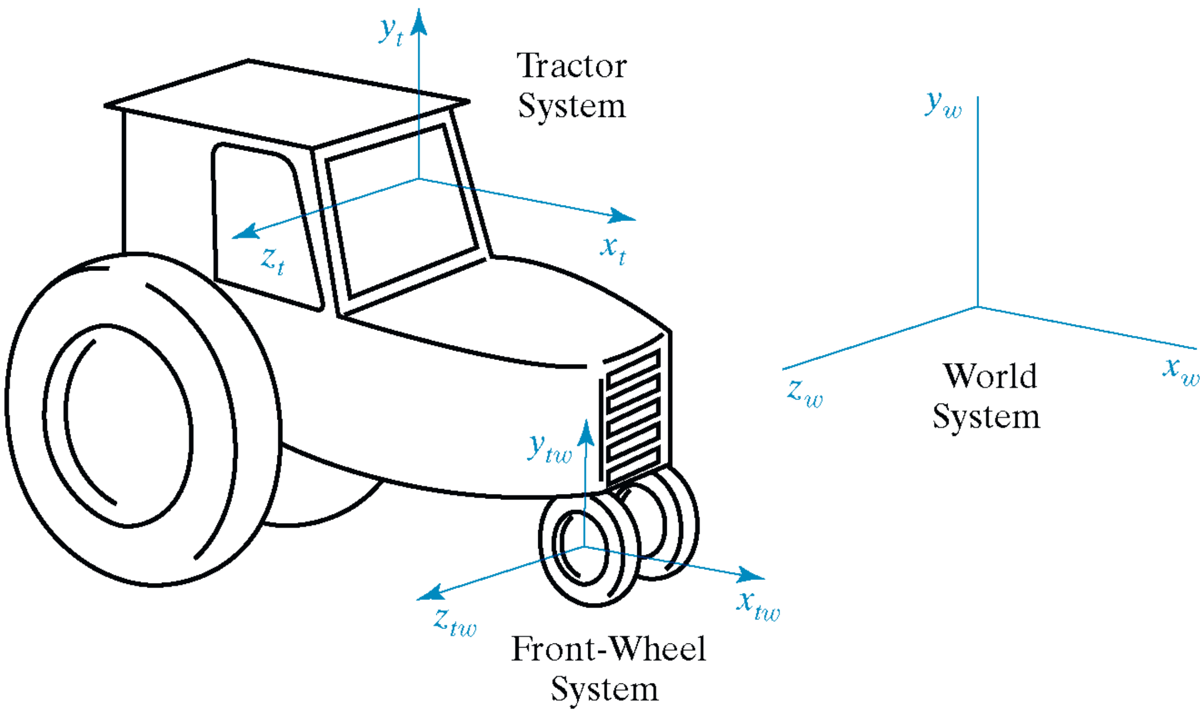


로보트 팔 만들기2



Hierarchical Modeling

- A hierarchical model is created by nesting the descriptions of subparts into one another to form a tree organization



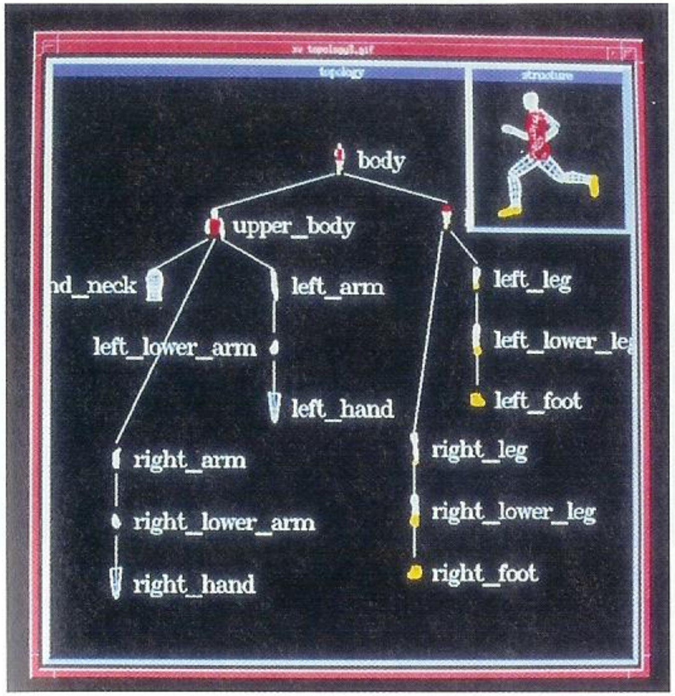


FIGURE 14-4 An object hierarchy generated using the PHIGS Toolkit package developed at the University of Manchester. The displayed object tree is itself a PHIGS structure. (*Courtesy of T. L. J. Howard, J. G. Williams, and W. T. Hewitt, Department of Computer Science, University of Manchester, United Kingdom.*)

OpenGL Matrix Stacks (OLD)

- Stack processing
  - The top of the stack is the "current" matrix
  - **glPushMatrix();** *//* Duplicate the current matrix at the top
  - **glPopMatrix();** *//* Remove the matrix at the top

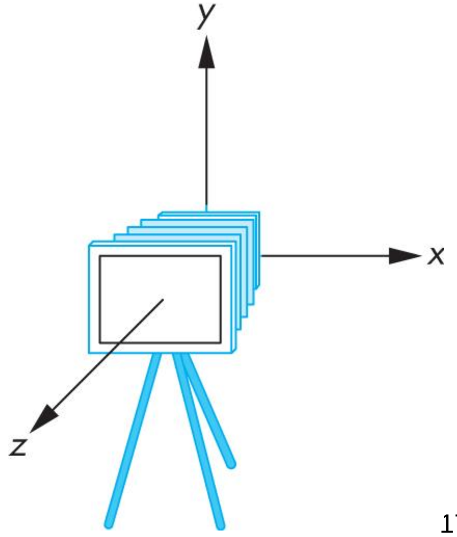
Matrix Stacks by your own

- We emulate Matrix Stacks by using:
  - Linked List such as ***std::list*** or ***std::deque***
  - Or a tree structure for more generality.

Model View Matrix II: Camera Positioning

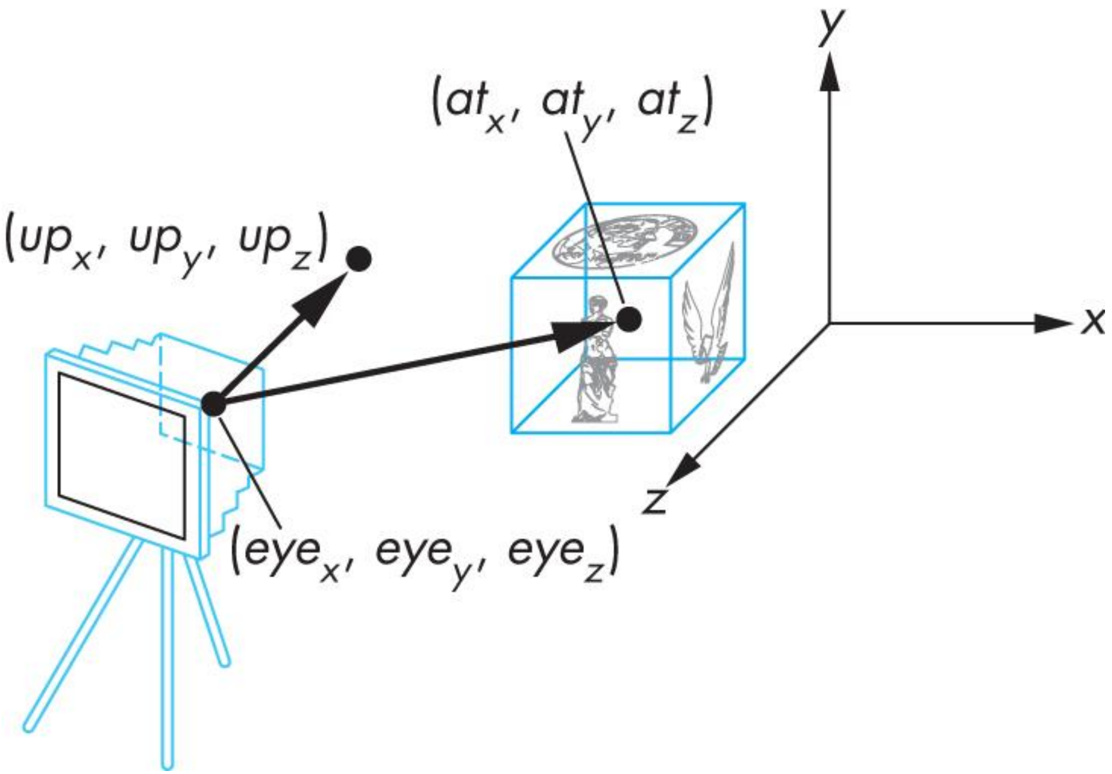
Transform Camera = Transform Scene

- Camera position is identified with a frame
- Either move and rotate the objects
- Or move and rotate the camera
- Initially, camera at origin, pointing in negative z-direction



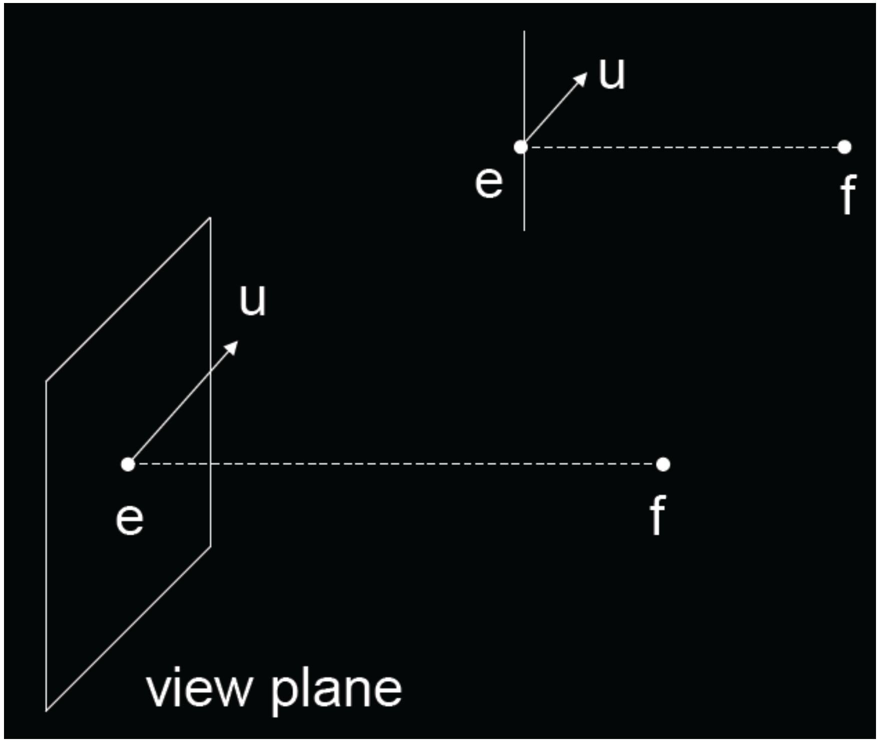
The Look-At Function

- Convenient way to position camera with three parameters:



The Look-At Function (OLD Style)

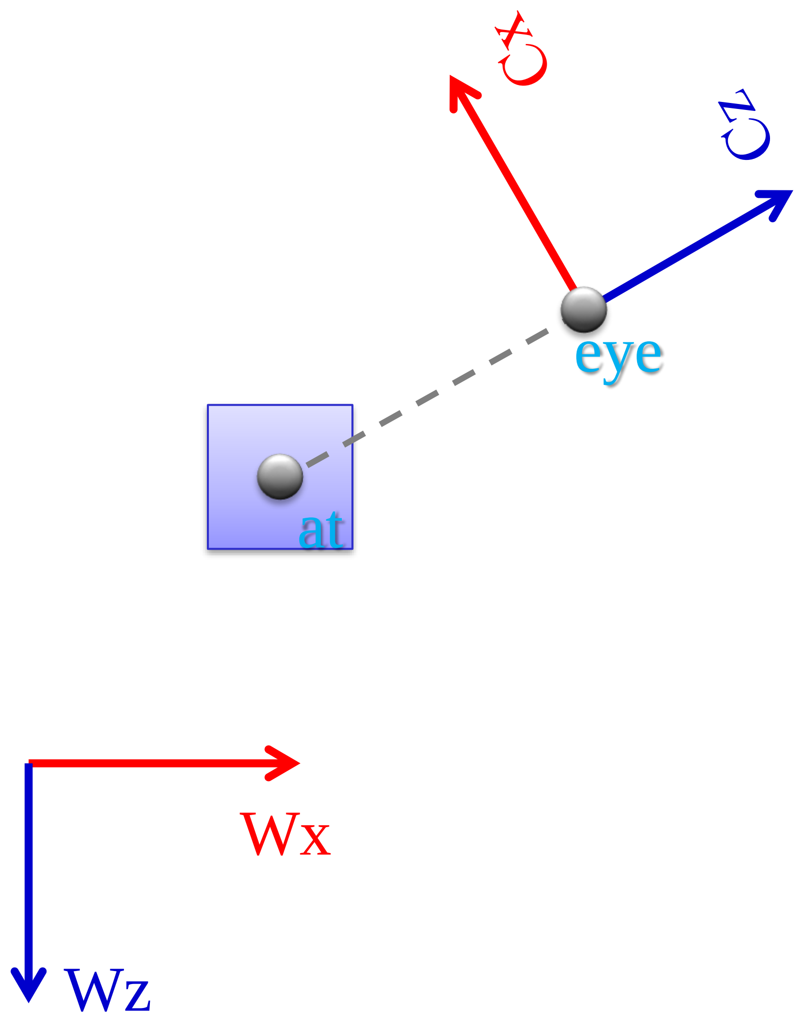
- `gluLookAt (ex,ey,ez, fx,fy,fz, ux,uy,uz);`
- e = eye point (eye)
- f = focus point (at)
- u = up vector (up)



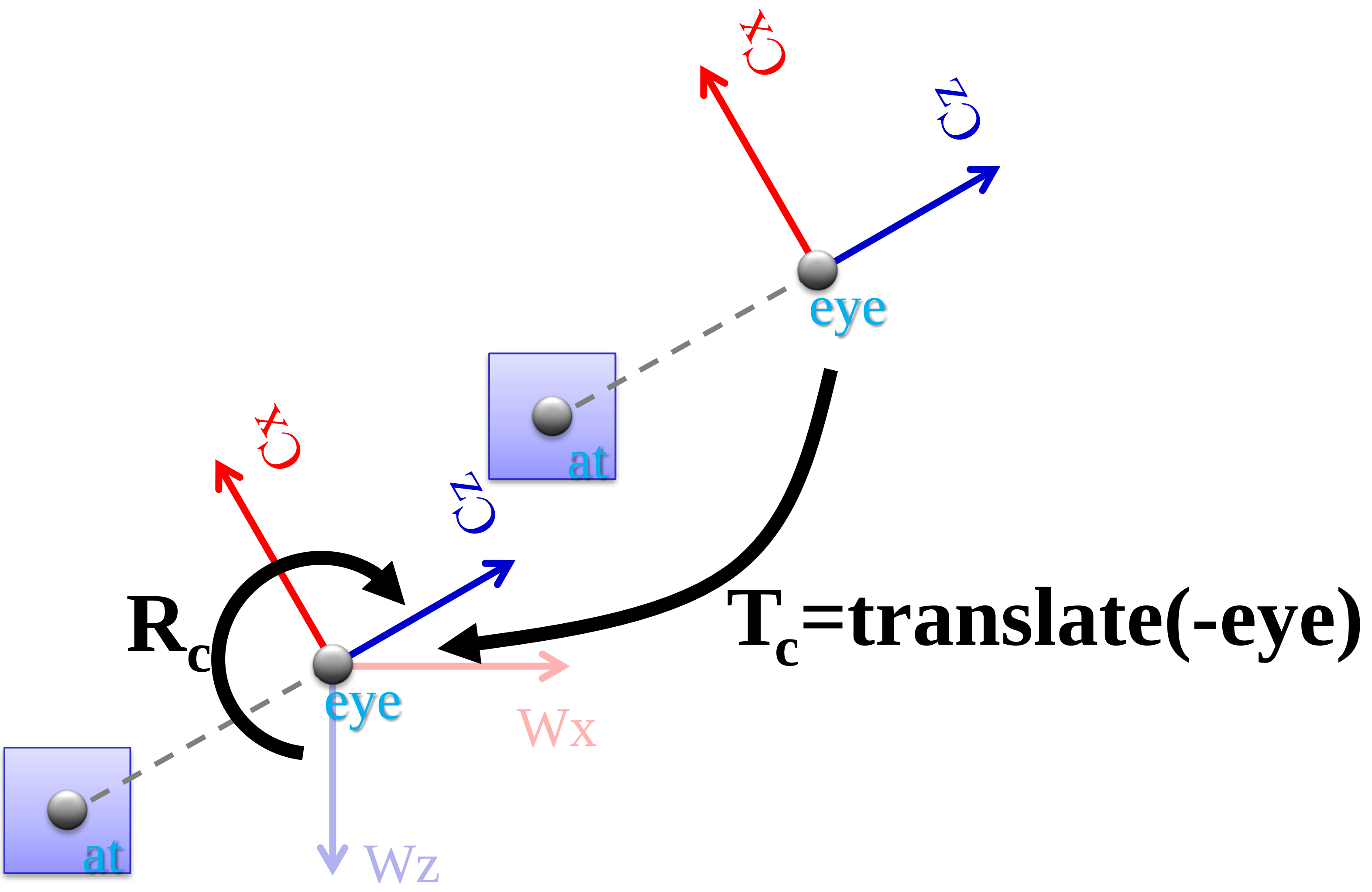
Old OpenGL code

```
void display()
{
      glClear (GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT);
      glMatrixMode (GL_MODELVIEW);
      glLoadIdentity ();
      gluLookAt (ex,ey,ez,fx,fy,fz,ux,uy,uz);
      ...
      renderObjects();
      glutSwapBuffers();
}
```

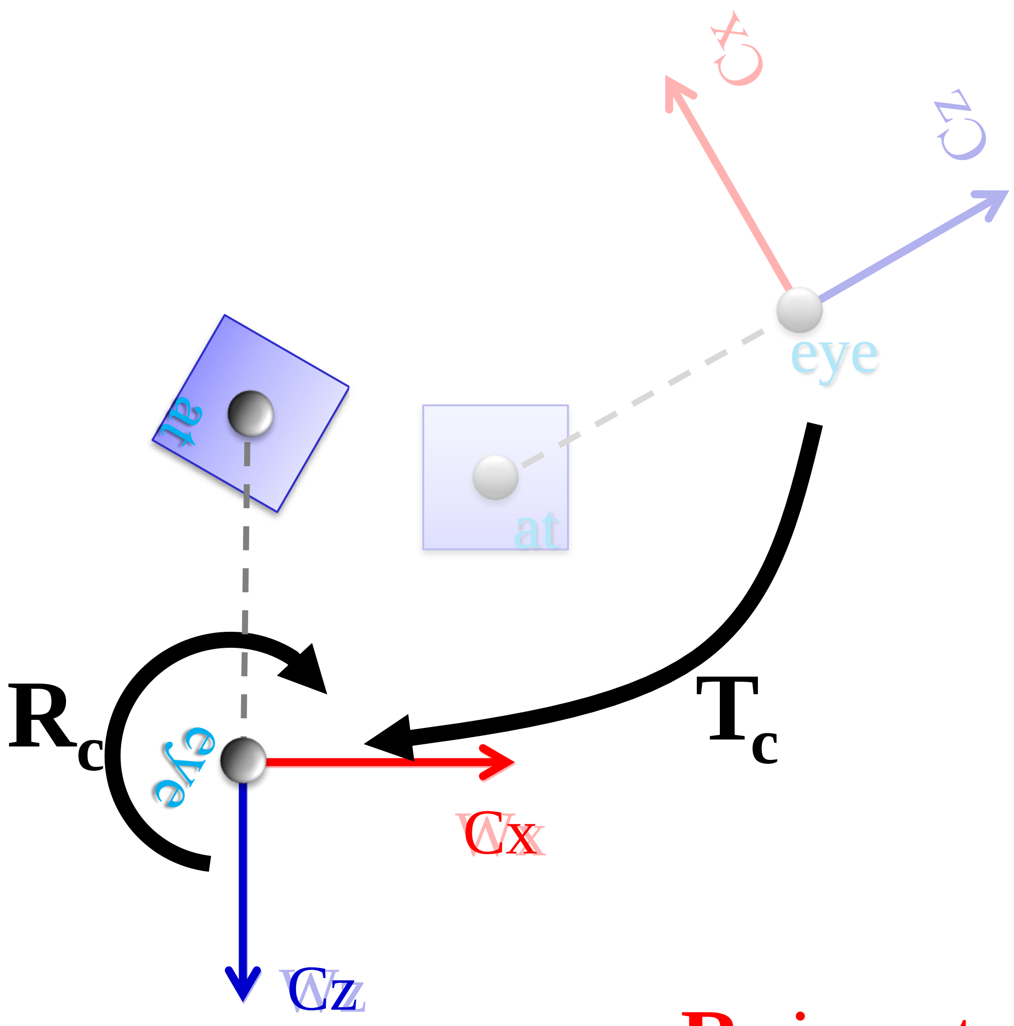
How to Compute?



2D case: Translate + Rotation



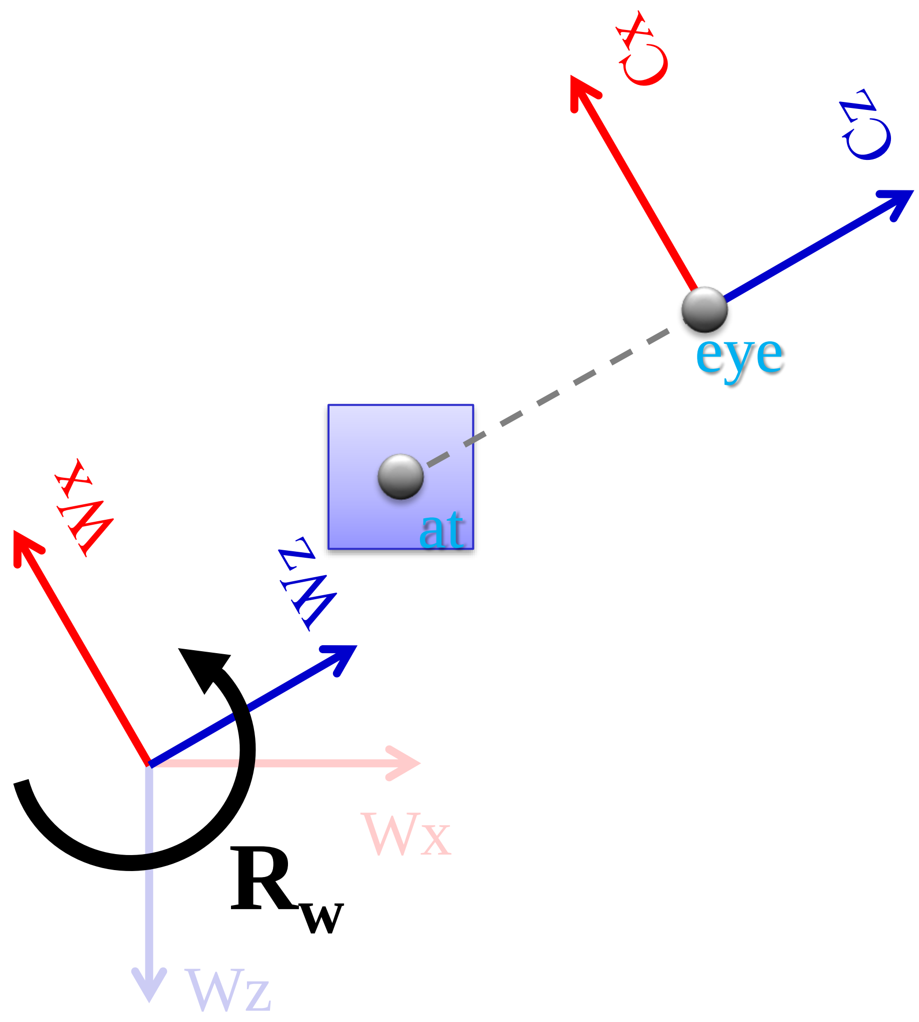
2D case: Translate + Rotation



$$V=R_cT_c$$

R_c is not easy to compute!

Instead,
Think about inverse transform:



$$R_c = (R_w)^{-1}$$
$$= (R_w)^T$$

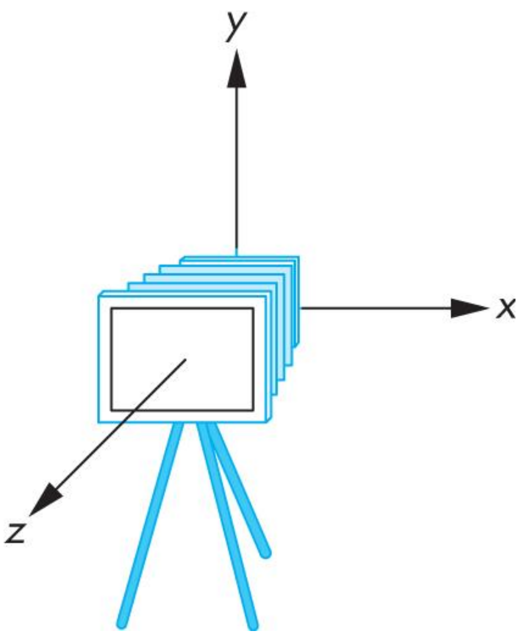
Implementing the Look-At Function

Plan:

1. Transform world frame to camera frame
   - Compose a rotation R with translation T
   - $W = T\,R$
2. Invert W to obtain viewing transformation V
   - $V = W^{-1} = (T\,R)^{-1} = R^{-1}\,T^{-1}$
   - Derive R, then T, then $R^{-1}\,T^{-1}$

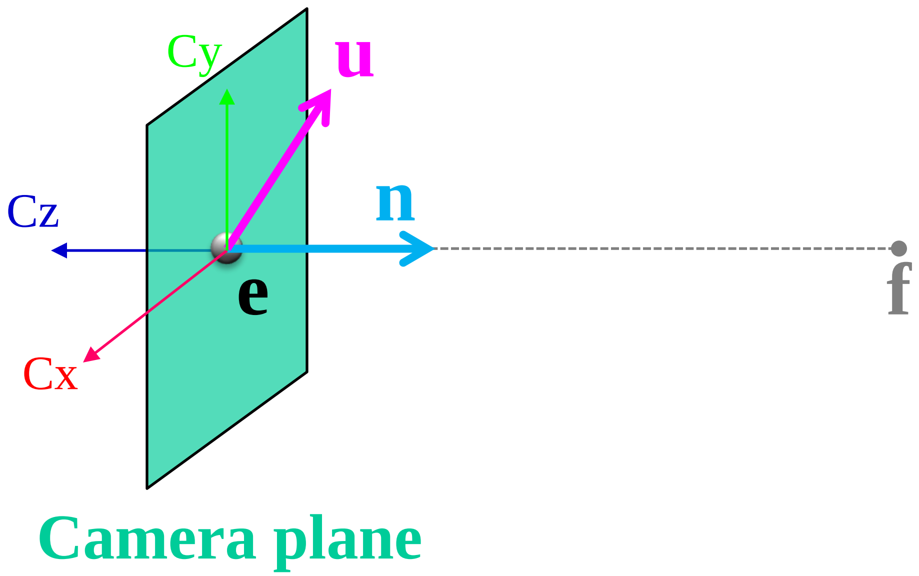
Viewing in OpenGL

- Remember:
  camera is pointing in the negative z direction



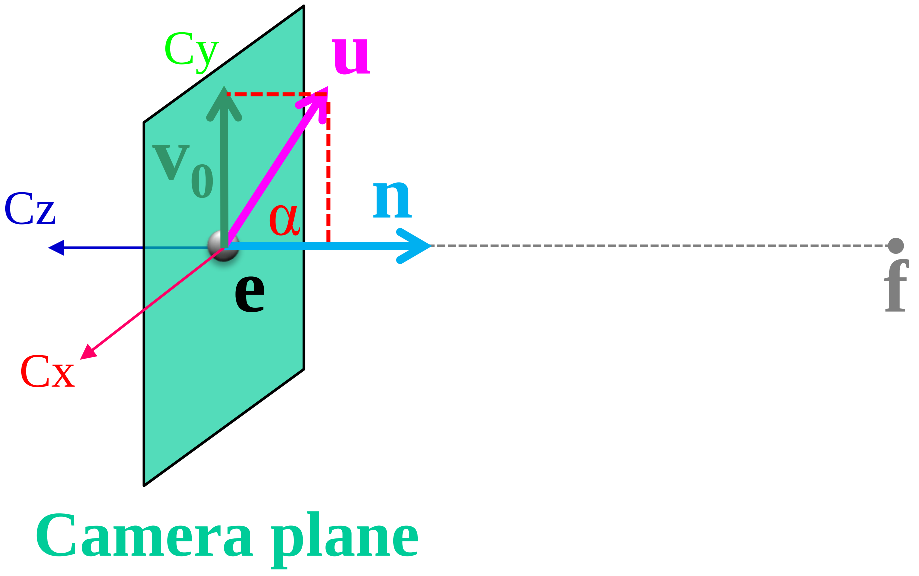
World Frame to Camera Frame I

- Camera points in negative z direction
- n = (f – e) / |f – e| is unit normal to view plane
- Therefore, R_w maps $[0\ 0\ -1]^T$ to $[nx\ ny\ nz]^T$



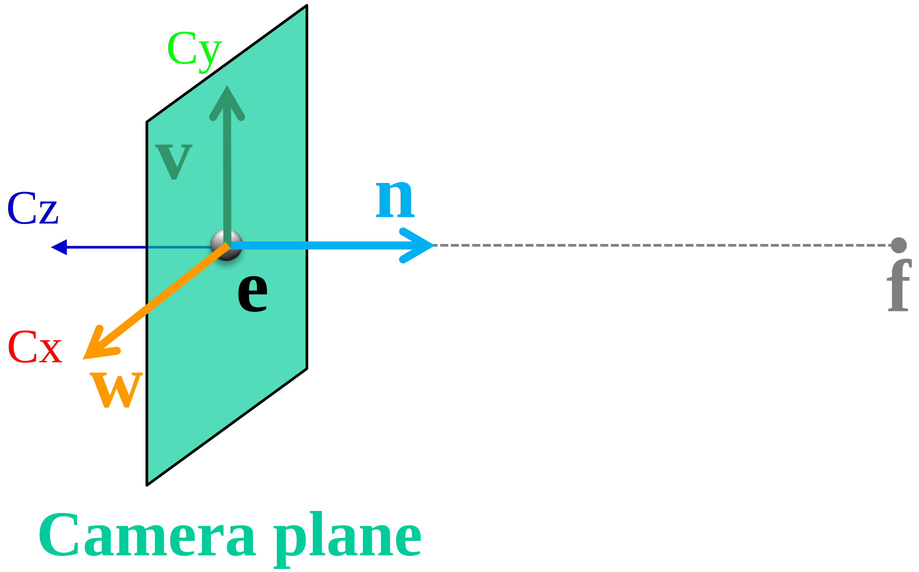
World Frame to Camera Frame II

- R_w maps $[0,1,0]^T$ to projection of u onto view plane
- This projection v equals:
  - $\alpha = (u \cdot n) / |n| = u \cdot n$
  - $v_0 = u - \alpha\, n$
  - $v = v_0 / |v_0|$



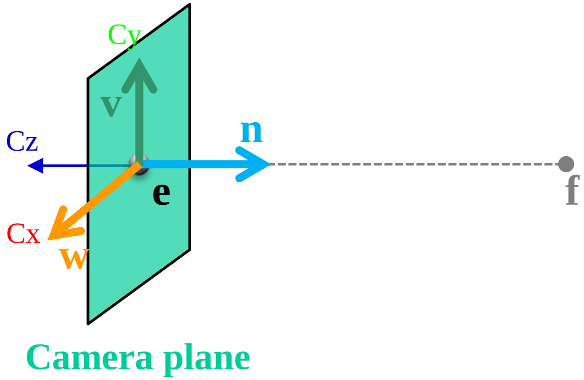
World Frame to Camera Frame III

- Set w to be orthogonal to n and v
- w = n x v
- (w, v, -n) is right-handed



Summary of Rotation

- **gluLookAt (ex,ey,ez, fx,fy,fz, ux,uy,uz);**
- n = (f – e) / |f – e|
- v = (u – (u · n) n) / |u – (u · n) n|
- w = n x v



- Rotation must map:
  - (1,0,0) to w
  - (0,1,0) to v
  - (0,0,-1) to n

$$\mathbf{R}_w = \begin{bmatrix} w_x & v_x & -n_x & 0 \\ w_y & v_y & -n_y & 0 \\ w_z & v_z & -n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

World Frame to Camera Frame IV

- Translation of origin to $e = [e_x \; e_y \; e_z \; 1]^T$

$$\mathbf{T}_w = \begin{bmatrix} 1 & 0 & 0 & e_x \\ 0 & 1 & 0 & e_y \\ 0 & 0 & 1 & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera Frame to Rendering Frame

- $V = W^{-1} = (T_w R_w)^{-1} = R_w^{-1} T_w^{-1}$
- R is rotation, so $R_w^{-1} = R_w^T$

$$\mathbf{R}^{-1} = \begin{bmatrix} w_x & w_y & w_z & 0 \\ v_x & v_y & v_z & 0 \\ -n_x & -n_y & -n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- T is translation, so T^{-1} negates displacement

$$\mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Putting All Together

- Calculate $V = R_w^{-1} T_w^{-1}$

$$\mathbf{V} = \mathbf{R}^{-1}\mathbf{T}^{-1} = \begin{bmatrix} w_x & w_y & w_z & -w_x e_x - w_y e_y - w_z e_z \\ v_x & v_y & v_z & -v_x e_x - v_y e_y - v_z e_z \\ -n_x & -n_y & -n_z & n_x e_x + n_y e_y + n_z e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- This is different from book [Angel, Ch. 4.3.2]
- There, u, v, n are right-handed (here: u, v, -n)