Endnote question

- About Quiz score
  - The score itself does not count towards your final grade. The fact that you took it earnestly counts!
  - The score helps you to assess yourself as well as me to figure out how the class understands the subject.
- About Lab session results
  - Lab attendance is counted towards your grade.
  - 1 *completed* mandatory lab worth 5 attendance points.
    - 1 lecture = 1 attendance point.
    - We have 3 mandatory labs!
    - Lab 2 was optional.
  - What you submit at the end of the lab (or as a homework for the lab) will be considered for 'completion' of the lab.
    - Our TA will just check if you did finish it or not!
  - The lab is open for your learning! (just like a lecture)

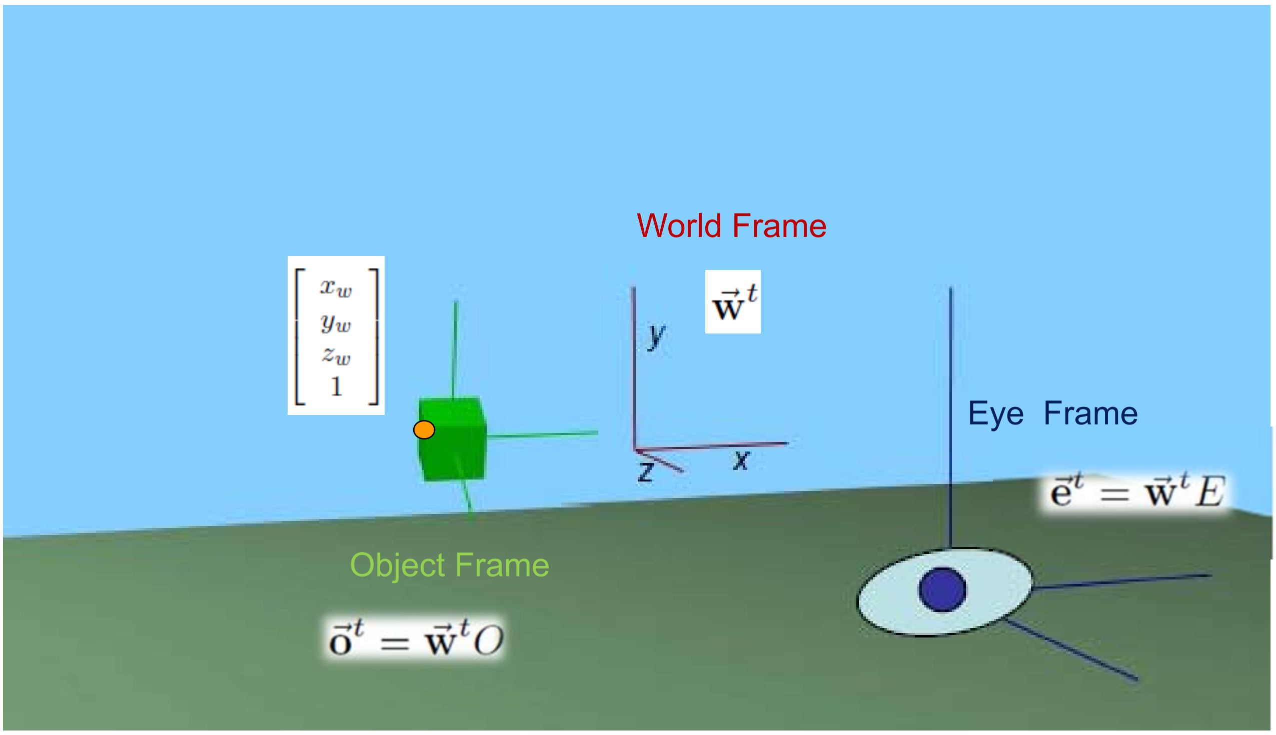
Homework schedule

Thu/Thur			Wednesday	
Date	**Topic**	**Assignment**		**TA (LAB)**
Mar 8, 10	Introduction and HelloWorld 2D	HW #0	9	OpenGL Intro 1 (Simple 2D)
Mar 15, 17	Linear and Affine Transformation	HW #1	14	open lab
Mar 22, 24	Frames in Graphics	**Due:3/30**	23	OpenGL Intro 2 (3D & viewing)
Mar 29, 31	HelloWorld 3D, Projection	HW #2	30	open lab
Apr 5, 7	Depth	<transformation 1>	6	open lab
Apr 12, 14	From Vertex to Pixels	**Due:4/12**	13	<Election day>
Apr 19	Geometric Modeling,	HW #3	20	open lab
Apr 20~26	Midterm Exam	<transformation 2>		
Apr 28, May 3	Color and Shading	**Due:4/28**	4	Lighting setup exercise
May 10, 12	Raytracing	HW #4	11	open lab
May 17, 19	Lighting	Shading/Lighting	18	open lab
May 24, 26	Texture Mapping		25	Texture mapping exercise
May 31, Jun 2	Sampling	HW #5	1	open lab
Jun 7, 9	Resampling	Texture mapping	8	open lab
Jun 14	Animation			
Jun 15~21	Final Exam			

Frames in Graphics

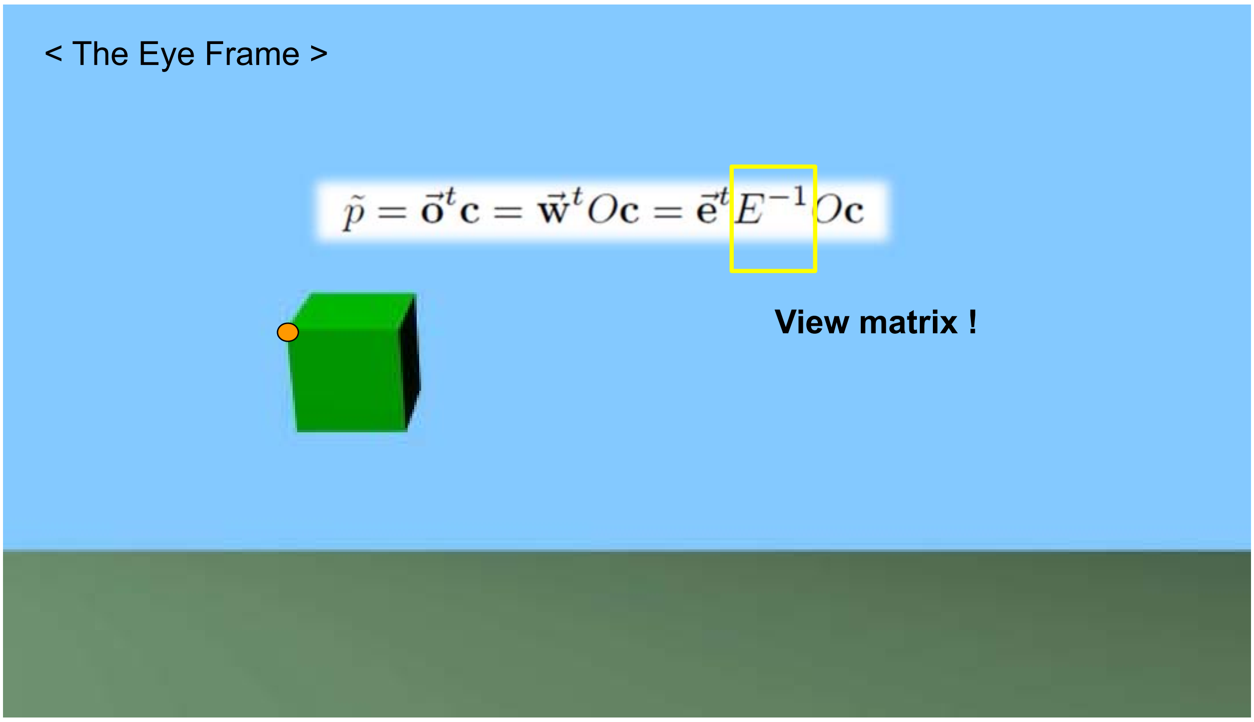
Chapter 5

World frame is the fixed one.



It is these eye coordinates which specify where **each vertex** appears **in the rendered image.**

$$\begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = E^{-1}O \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$



Moving Things Around

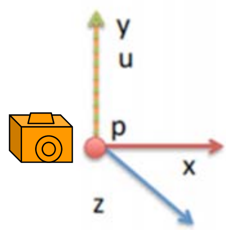
- **Moving the Eye** <Lookat>
- → We use the auxiliary coordinate system where the eye would orbit around the (center of the) object.

$$\vec{\mathbf{e}}^t = \vec{\mathbf{w}}^t E, \qquad E \leftarrow EM$$

$$E = \begin{bmatrix} x_1 & y_1 & z_1 & p_1 \\ x_2 & y_2 & z_2 & p_2 \\ x_3 & y_3 & z_3 & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{z} = normalize(p - q)$$
$$\mathbf{x} = normalize(\mathbf{u} \times \mathbf{z})$$
$$\mathbf{y} = \mathbf{z} \times \mathbf{x}$$

- Specify E by
  – the eye point $\tilde{p}$
  – the view point $\tilde{q}$
  – the up vector $\vec{u}$



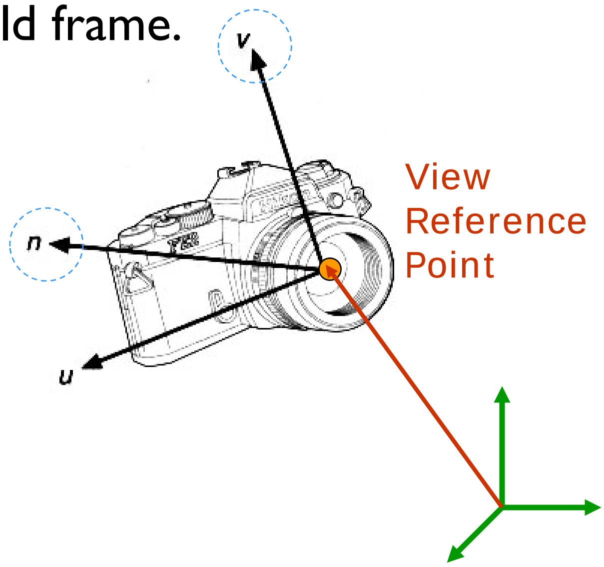
< Slide from Lec06 (March 22) >

Direct Camera Placement

- Describe the camera's position and orientation in the world frame.

- **Define the viewing coordinate system, u-v-n.**

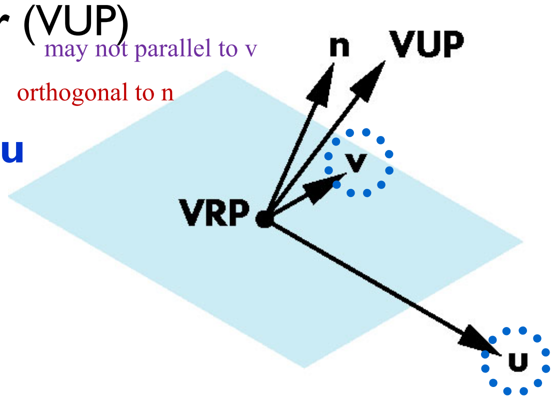
User specifies a view-reference point,
a view-up vector and
a view-plane normal.



Direct Camera Placement

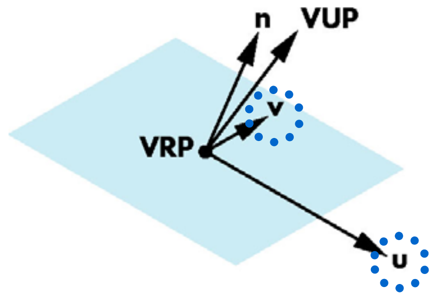
- Describe the camera's position and orientation in the world frame.
  - Specify the **view reference point** (VRP)
  - Specify the **view plane normal, n** orientation of the projection plane / back of the camera
  - Specify the **view-up vector** (VUP) may not parallel to v
  - Compute the **up-vector, v** orthogonal to n
  - Compute the **side vector, u**

  - Defines the viewing coordinate system, u-v-n.

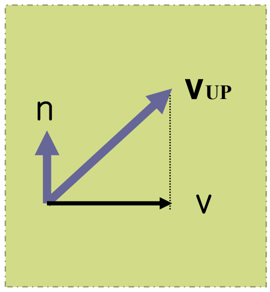


Direct Camera Placement

- Viewing-coordinate system.
  - **VRP = p = $(x,y,z,1)^T$**
  - **n = $(n_x, n_y, n_z, 0)^T$**
  - **$v_{UP} = (v_{up\_x}, v_{up\_y}, v_{up\_z}, 0)^T$**



  - Compute the up-vector, **v**
    - **$n \bullet v = 0$**
    - **$v = v_{UP} - \{(v_{UP} \bullet n) / (n \bullet n)\}n$**
  - Compute the side vector, **u**
    - **$u = v \times n$**
  - Normalize **u,v,n**
  - **u' – v' – n' system**

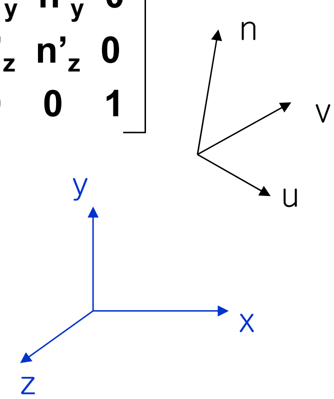


v is a projection of vup into the plane formed by n and vup
therefore v = a n + b vup, let b=1 solve for a, therefore v = …

Direct Camera Placement

- View-orientation matrix
  - Orients a vector in the u'-v'-n' with respect to the original system

$$E = \begin{bmatrix} u'_x & v'_x & n'_x & 0 \\ u'_y & v'_y & n'_y & 0 \\ u'_z & v'_z & n'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

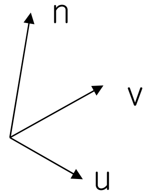


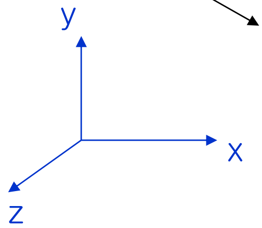
  - The rotation matrix
  - We want to represent the vector in the original system with respect to the camera system.
    - u – v – n system

Direct Camera Placement

- View-orientation matrix
  - Orients a vector in the u'-v'-n' with respect to the original system
  - The rotation matrix

$$E = \begin{bmatrix} u'_x & v'_x & n'_x & 0 \\ u'_y & v'_y & n'_y & 0 \\ u'_z & v'_z & n'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$





- **We want E^{-1}**
  - Because E is a rotation matrix, $E^{-1} = E^T$

$$E^T = \begin{bmatrix} u'_x & u'_y & u'_z & 0 \\ v'_x & v'_y & v'_z & 0 \\ n'_x & n'_y & n'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

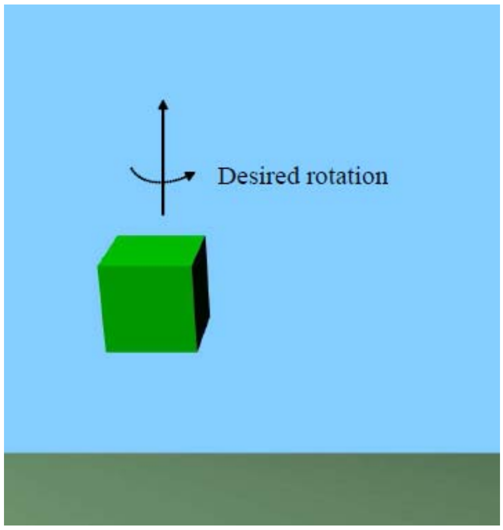
Moving Things Around

- Suppose we want to rotate the object about its own center about the viewers y-axis
  - What will be a good choice for the auxiliary frame?

$$\vec{\mathbf{a}}^t = \vec{\mathbf{w}}^t A$$

$$\boxed{A = (O)_T(E)_R}$$

$$\begin{aligned} \vec{\mathbf{o}}^t &= \vec{\mathbf{w}}^t O \\ &= \vec{\mathbf{a}}^t A^{-1} O \quad \Rightarrow \vec{\mathbf{a}}^t M A^{-1} O \\ &\qquad\qquad\qquad = \vec{\mathbf{w}}^t A M A^{-1} O \end{aligned}$$



$$\boxed{\vec{\mathbf{a}}^t = \vec{\mathbf{w}}^t (O)_T (E)_R}$$

For Moving the Eye

- Eye frame can be moved just like an object frame:
- $$E \leftarrow AMA^{-1}E.$$
- To have eye orbit around the object: $A = (O)_T(E)_R$

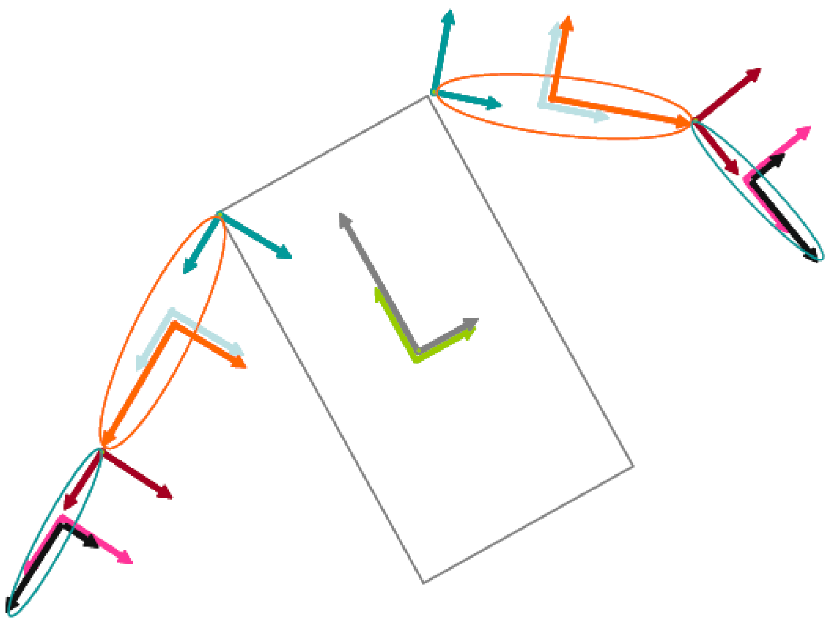
  $\vec{\mathbf{a}}^t = \vec{\mathbf{e}}^t$
- To have eye orbit center of the room: $A = (E)_R$
- To have egomotion, we can choose $\vec{\mathbf{a}}^t = \vec{\mathbf{e}}^t$

  giving us $A = E$

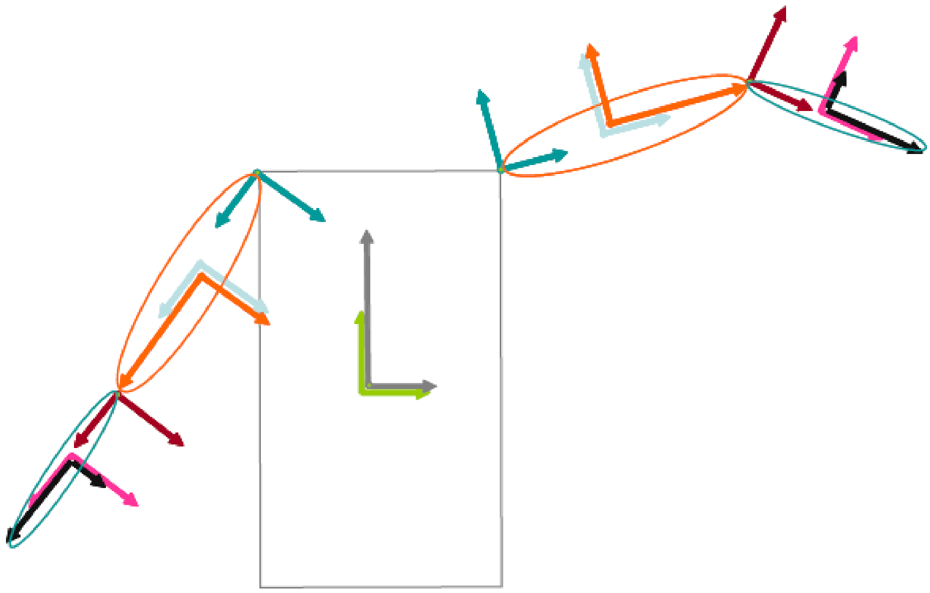
  <Lookat>

$$\boxed{\begin{array}{l}\vec{\mathbf{e}}^t = \vec{\mathbf{w}}^t E, \\ E \leftarrow EM\end{array}}$$

Moving object

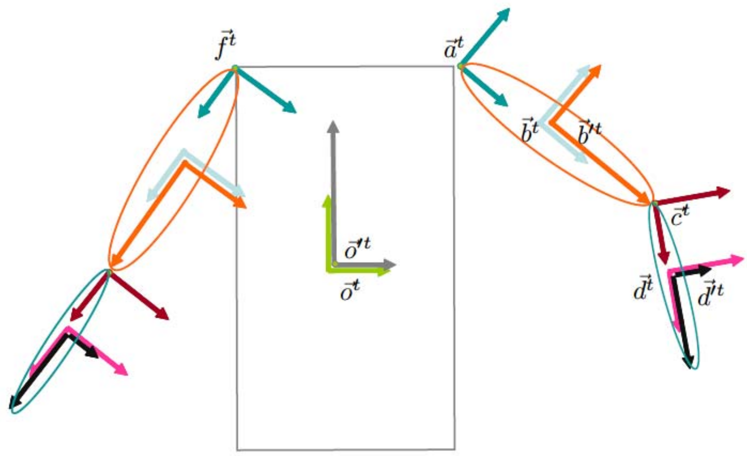


Moving object parts



Moving object parts in Hierarchy

- An object can be treated as being assembled by some fixed and movable subobjects.

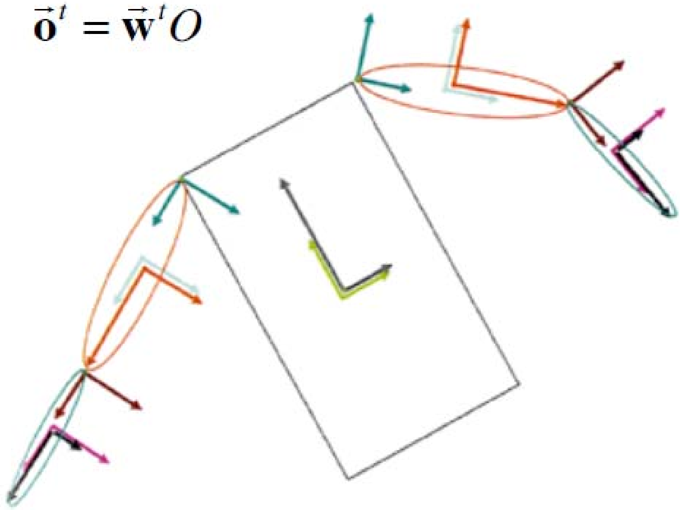


$$\vec{\mathbf{o}}^t = \vec{\mathbf{w}}^t O$$
$$\vec{\mathbf{o}}'^t = \vec{\mathbf{o}}^t O'$$
$$\vec{\mathbf{a}}^t = \vec{\mathbf{o}}^t A$$
$$\vec{\mathbf{b}}^t = \vec{\mathbf{a}}^t B$$
$$\vec{\mathbf{b}}'^t = \vec{\mathbf{b}}^t B'$$
$$\vec{\mathbf{c}}^t = \vec{\mathbf{b}}^t C$$
$$\vec{\mathbf{d}}^t = \vec{\mathbf{c}}^t D$$
$$\vec{\mathbf{d}}'^t = \vec{\mathbf{d}}^t D'$$
$$\vec{\mathbf{f}}^t = \vec{\mathbf{o}}^t F$$

Moving the entire robot

- We just update its O matrix to the object frame, instead of relating it to the world frame

$$\vec{\mathbf{o}}^t = \vec{\mathbf{w}}^t O$$



$$\vec{\mathbf{o}}^t = \vec{\mathbf{w}}^t O$$

$$\vec{\mathbf{a}}^t = \vec{\mathbf{w}}^t OA$$

$$\vec{\mathbf{b}}^t = \vec{\mathbf{w}}^t OAB$$

$$\vec{\mathbf{b}}'^t = \vec{\mathbf{w}}^t OABB'$$

$$\vec{\mathbf{c}}^t = \vec{\mathbf{w}}^t OABC$$

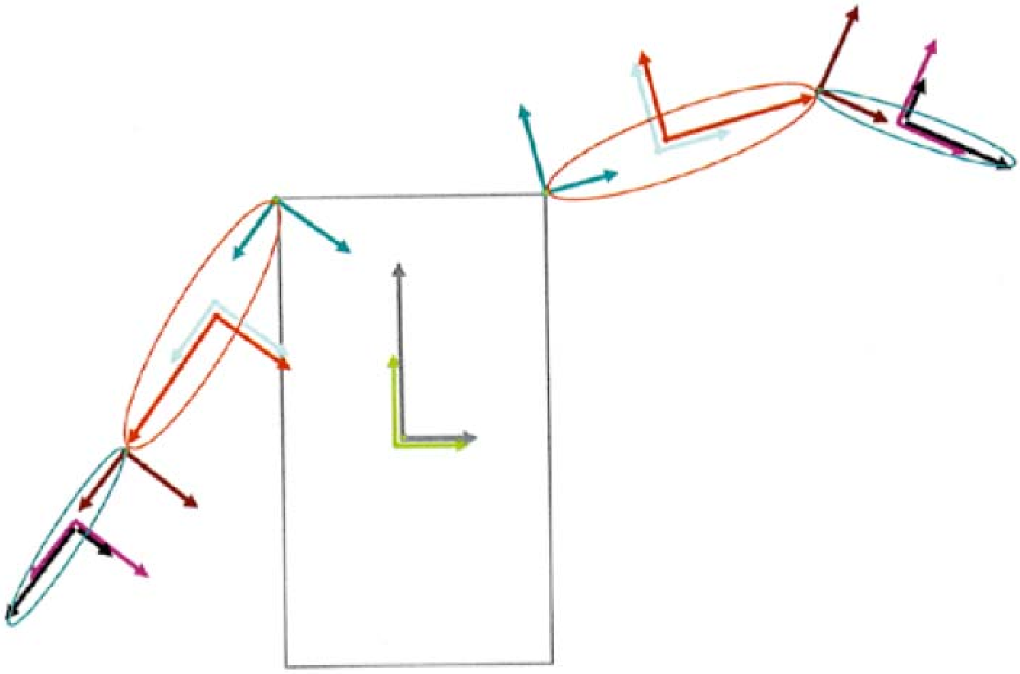
$$\vec{\mathbf{d}}^t = \vec{\mathbf{w}}^t OABCD$$

$$\vec{\mathbf{d}}'^t = \vec{\mathbf{w}}^t OABCDD'$$

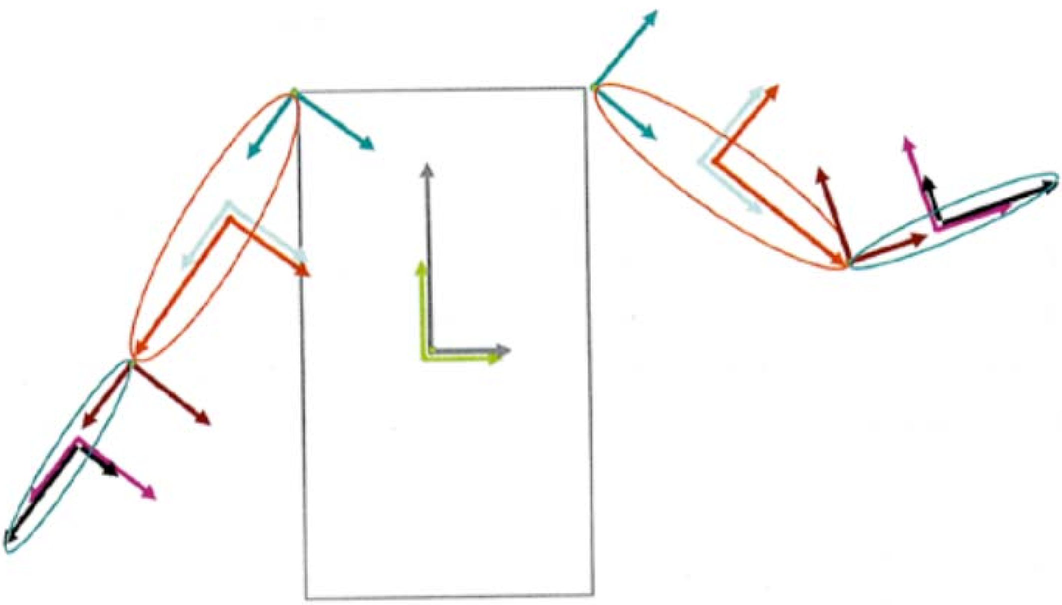
Matrix stack

- Matrix stack data structure can be used to keep track of the matrix
- push(M)
  - creates a new 'topmost' matrix
  - a copy of the previous topmost matrix
  - M. multiplies this new top matrix
- pop()
  - removes the topmost layer of the stack
- descending
  - descend down to a subobject, when a push operation is done
  - this matrix is popped off the stack when returning from this descent to the parent

Moving limbs



Moving limbs



Scene graph pseudocode

```
...
matrixStack.initialize(inv(E));
matrixStack.push(O);
    matrixStack.push(O');
        draw(matrixStack.top(), cube); \\ body
    matrixStack.pop(); \\ O'

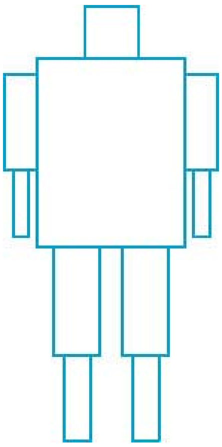
    matrixStack.push(A); \\ grouping
        matrixStack.push(B);
            matrixStack.push(B');
                draw(matrixStack.top(), sphere); \\ upper arm
            matrixstack.pop(); \\ B'

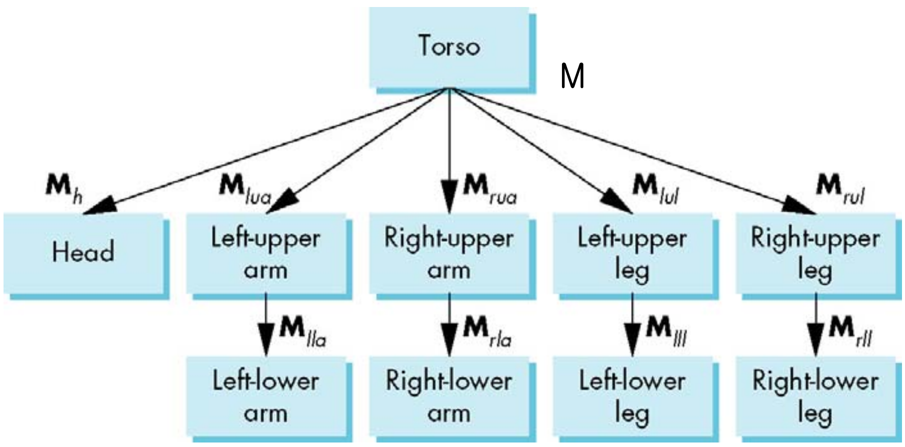
            matrixStack.push(C);
                matrixStack.push(C');
                    draw(matrixStack.top(), sphere); \\ lower arm
                matrixStack.pop(); \\ C'
            matrixStack.pop(); \\ C
        matrixStack.pop(); \\ B
    matrixStack.pop(); \\ A
\\ current top matrix is inv(E)*O

\\ we can now draw another arm
    matrixStack.push(F);
```


Trees and Traversal

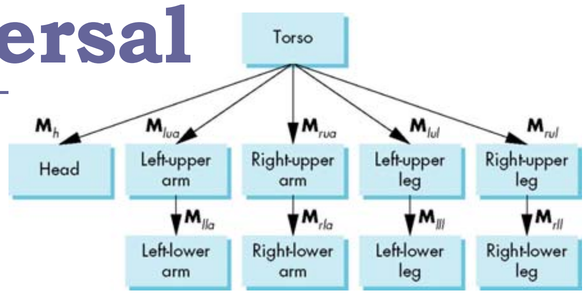
- Example: a humanoid figure





- How to traverse the tree to draw the figure?
  - left to right, depth first (pre-order traversal)

Trees and Traversal



- A stack-based traversal

```
mat4 mv; /* model_view */
matrix_stack mvstack;
```

- ```
  figure() {
      mvstack.push(mv);
      torso();
      mv = mv * Translate()*
                Rotate();
      head();
      mv = mvstack.pop();
      mvstack.push(mv);
      mv = mv * Translate()*
               Rotate();
      left_upper_arm();
  ```

```
mv = mv * Translate()*
          Rotate ();
left_lower_arm();
mv = mvstack.pop();
mvstack.push(mv);
mv = mv * Translate()*
          Rotate ();
right_upper_arm();
      :
      :
```

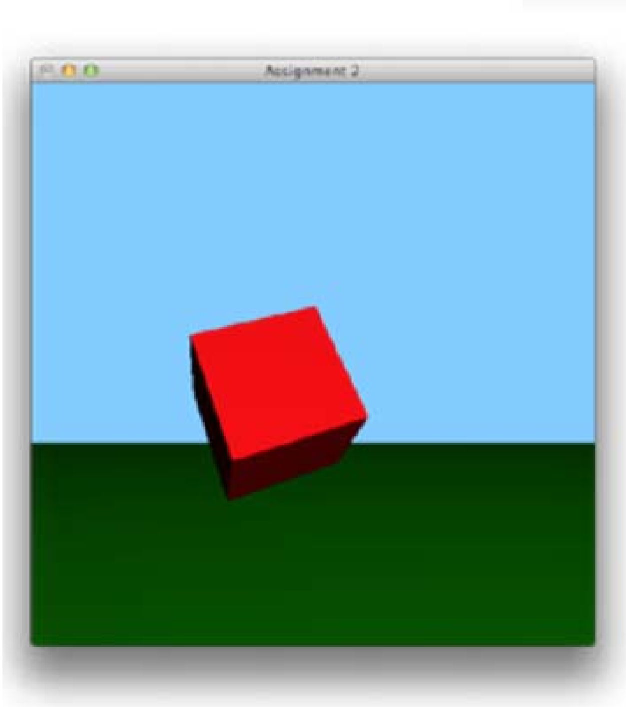

- Homework #3 (originally #2 part2) will cover this part for you to exercise!

Moving object parts in Hierarchy
```

Chapter 6

- 2D was a special case of 3D
- *Rendering pipeline itself* is the same!

HELLO WORLD 3D



Chapter 7 & 8

Homework #2 !

- 3D Rotations
- Quaternions

Chapter 10

- Cameras
- Projection