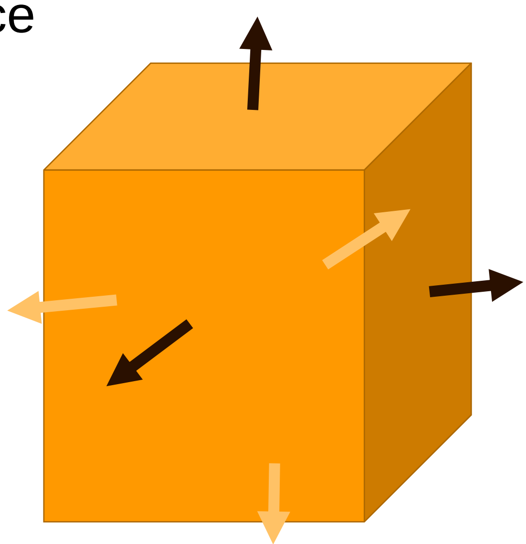
Endnote comments

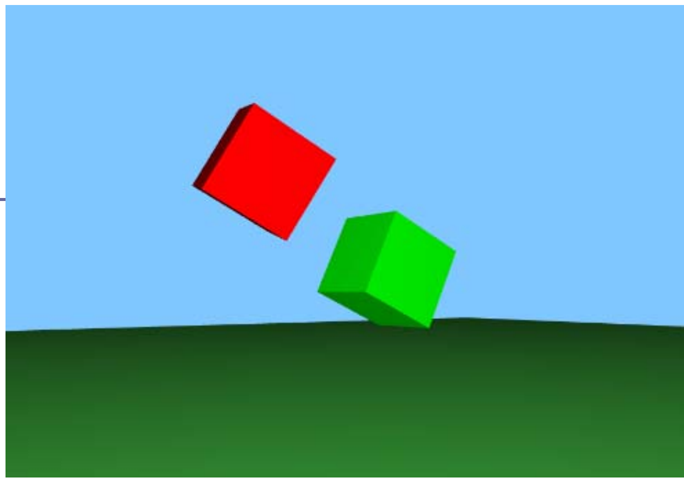
- About lecture slide
  - 수업전에 올려주세요
  - 올려주신 내용과 사용한 ppt순서가 달라 헷갈려요.
  - …
  - →Before coming, read the book and previous slides
  - →During class, please take a note for yourself while learning (No need to copy all slides though!)
  - →After lecture, review the slide uploaded.
  - Importantly, learn from the book! Not only from slides!

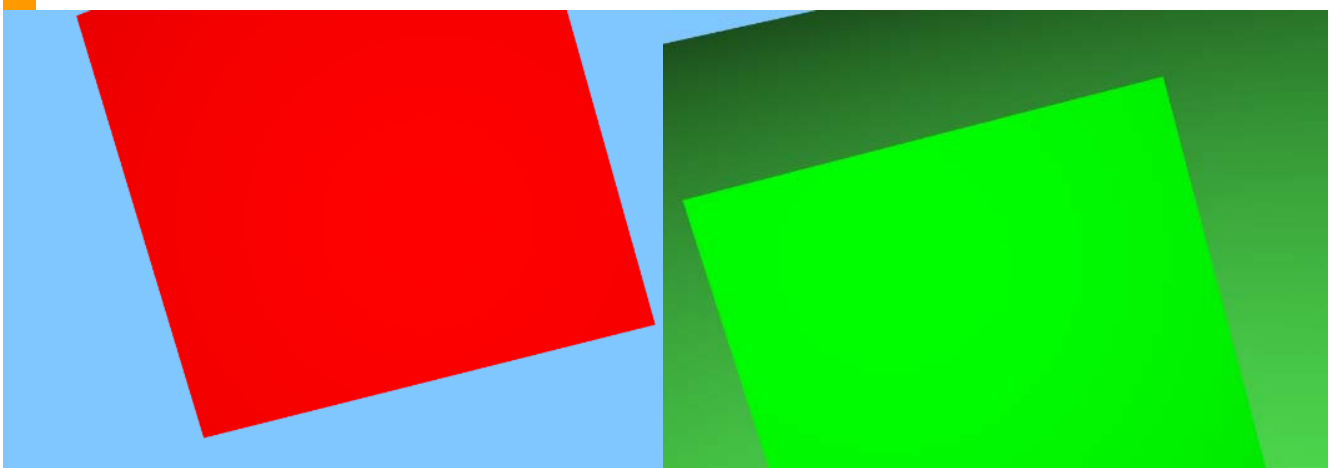
Announcement

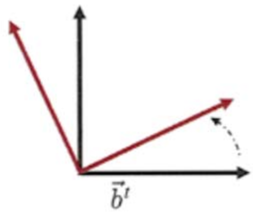
- Quiz
  - Average = 7.41 / 10
    - 10 and 9 : 41 students
    - 0 and below 2: 11 students
- Homework #1
  - Due: March 30 before midnight
- LAB yesterday
  - 57 / 70 Students attended and finished the lab.

- Front / back face









- change a basis of a vector $\vec{\mathbf{b}}^t$ to $\vec{\mathbf{a}}^t$

$$\vec{\mathbf{a}}^t = \vec{\mathbf{b}}^t M , \qquad \vec{v} = \vec{\mathbf{b}}^t \mathbf{c} = \vec{\mathbf{a}}^t M^{-1} \mathbf{c} .$$

- Linear transform of a vector

$$\vec{v} = \vec{\mathbf{b}}^t \mathbf{c} \Rightarrow \vec{\mathbf{b}}^t M \mathbf{c}$$

- Linear transform of a basis

$$\vec{v} = \vec{\mathbf{b}}^t \mathbf{c} = \vec{\mathbf{a}}^t M^{-1} \mathbf{c} .$$

Chapter 4. Respect

- Frame is important …

Chapter 5. Frames in Graphics

Scaling a point over frame

- We are transforming a point $\tilde{p}$ in a frame $\vec{\mathbf{f}}^t$

$$\tilde{p} = \vec{\mathbf{f}}^t \mathbf{c}$$

- With a matrix

$$\mathbf{S} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

the stretches by factor of two in first axis of $\vec{\mathbf{f}}^t$

- Performing a transform: $\vec{\mathbf{f}}^t \mathbf{c} \Rightarrow \vec{\mathbf{f}}^t S \mathbf{c}$

- Suppose another frame: $\vec{\mathbf{a}}^t = \vec{\mathbf{f}}^t A$

Scaling a point over frame

- We could express the point with a new coordinate vector

$$\tilde{p} = \vec{\mathbf{f}}^t \mathbf{c} = \vec{\mathbf{a}}^t \mathbf{d} \qquad \vec{\mathbf{a}}^t = \vec{\mathbf{f}}^t A$$

$$\vec{\mathbf{f}}^t \mathbf{c} = \vec{\mathbf{f}}^t A \mathbf{d} \qquad \vec{\mathbf{f}}^t = \vec{\mathbf{a}}^t A^{-1}$$

$$\mathbf{d} = A^{-1} \mathbf{c}$$

- Now S transforms the point $\tilde{p}$ with respect to $\vec{\mathbf{a}}^t$

$$\vec{\mathbf{a}}^t \mathbf{d} \Rightarrow \vec{\mathbf{a}}^t S \mathbf{d}$$

Left-of rule

- Point is transformed with respect to the the frame that appears immediately to the left of the transformation matrix in the expression.
- We read

$$\vec{\mathbf{f}}^t \Rightarrow \vec{\mathbf{f}}^t S$$

$\vec{\mathbf{f}}^t$ is transformed by S with respect to $\vec{\mathbf{f}}^t$

- We read

$$\vec{\mathbf{f}}^t = \vec{\mathbf{a}}^t A^{-1} \Rightarrow \vec{\mathbf{a}}^t S A^{-1}$$

$\vec{\mathbf{f}}^t$ is transformed by S with respect to $\vec{\mathbf{a}}^t$

Transforms using an Auxiliary Frame

- Sometimes we need to transform a frame $\vec{\mathbf{f}}^t$ in some specific way, represented by a matrix M, with respect to some auxiliary frame $\vec{\mathbf{a}}^t$

$$\vec{\mathbf{a}}^t \Rightarrow \vec{\mathbf{f}}^t A$$

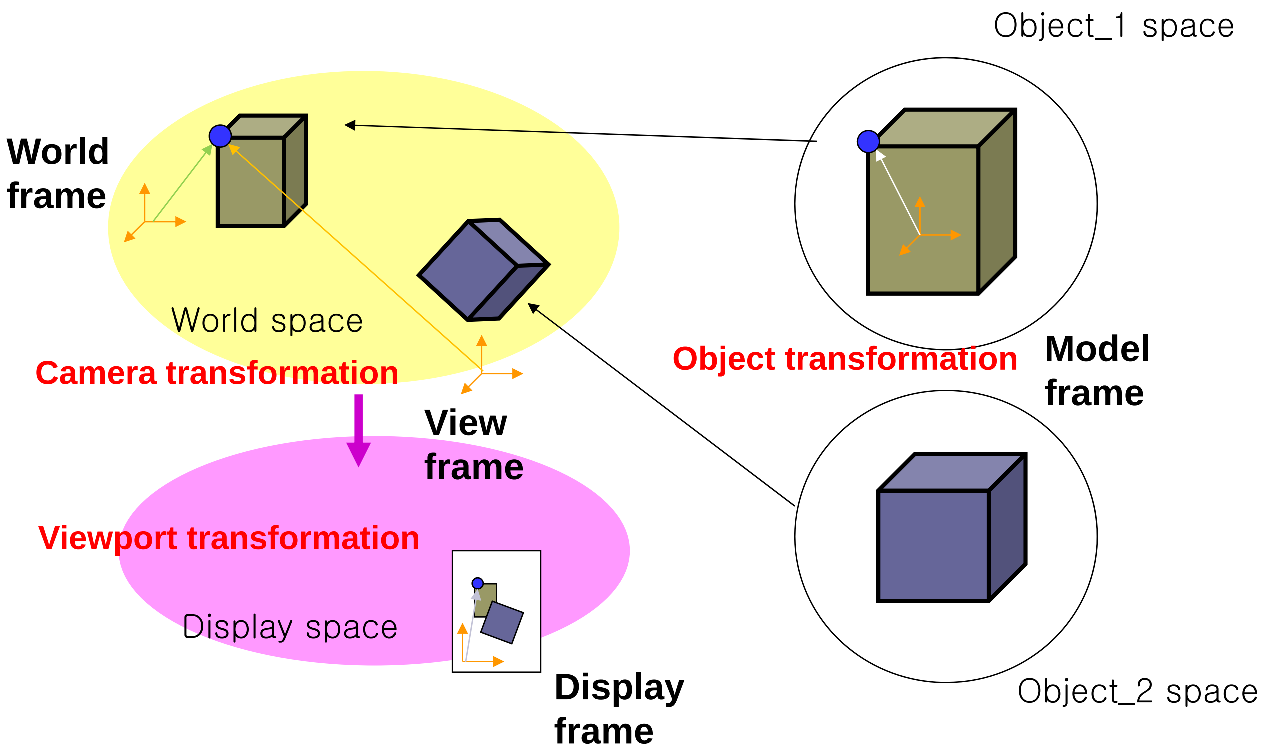
- The transform frame can then be expressed as

$$\vec{\mathbf{f}}^t$$

$$= \vec{\mathbf{a}}^t A^{-1}$$

$$\Rightarrow \vec{\mathbf{a}}^t M A^{-1}$$

$$= \vec{\mathbf{f}}^t A M A^{-1}$$



World, object and eye frames

- World frame (world coordinates)
  - a basic right-handed orthonormal frame $\vec{\mathbf{w}}^t$
  - we never alter this frame
  - other frames can be described wrt the world frame
- Object frame (object coordinates)
  - model the geometry of the object using vertex coordinates
  - not need to be aware of the global placement
  - a right-handed orthonormal frame of object $\vec{\mathbf{o}}^t$
- Eye frame (camera coordinates): later on

World vs. object frame

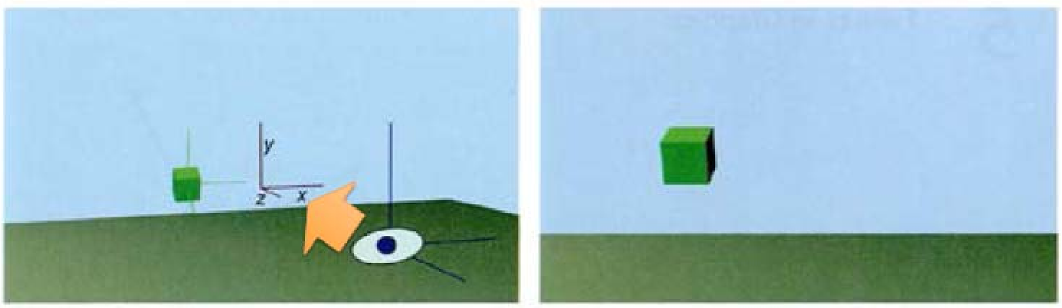
- The relationship between the world frame and object frame:
  - affine 4-by-4 matrix O (rigid body transformation: rotation + translation only)

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

- The meaning of O is the relationship between the world frame to the object's coordinate system.
- To move the object frame $\vec{\mathbf{o}}^t$ itself, we change the matrix O.

The eye's view

- The world frame is in red
- The object frame is in green
- The eye frame is in blue
  - The eye is looking down its negative z toward the object.

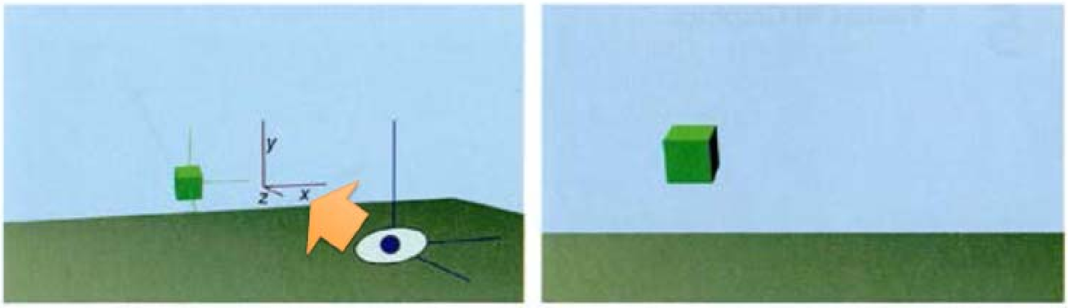


(a) The frames (b) The eye's view

The eye frame

- Eye frame (camera coordinates)
  - a right-handed orthonormal frame $\vec{\mathbf{e}}^t$
  - the eye looks down its negative z axis to make a picture

$$\vec{\mathbf{e}}^t = \vec{\mathbf{w}}^t E$$



(a) The frames (b) The eye's view

Extrinsic transformation of the eye

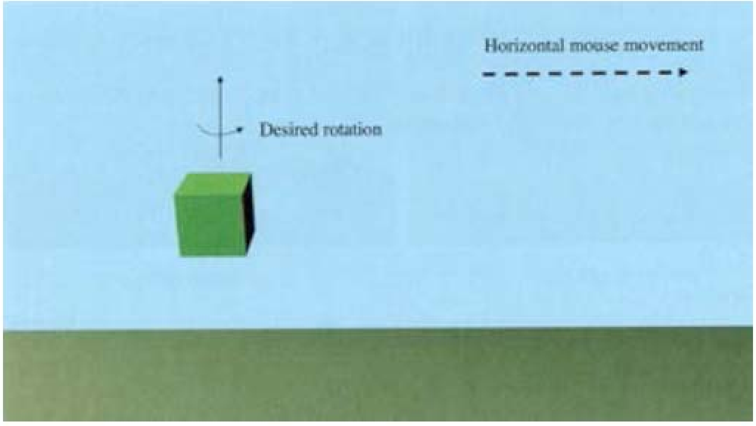
- we explicitly store the matrix E $\vec{\mathbf{e}}^t = \vec{\mathbf{w}}^t E$

$$\tilde{p} = \vec{\mathbf{o}}^t \mathbf{c} = \vec{\mathbf{w}}^t O\mathbf{c} = \vec{\mathbf{e}}^t E^{-1} O\mathbf{c}$$

  - Object coordinates: $\mathbf{c}$
  - World coordinates: $O\mathbf{c}$
  - Eye coordinates: $E^{-1}O\mathbf{c}$
  - Calculating the eye coordinates of every vertexes:

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

Moving an Object



- We want the object to rotate around its own center about the viewer's *y* axis, when we move the mouse to the right.
- How we could do this?

Moving an Object

- Basic idea: set a frame $\vec{\mathbf{a}}^t = \vec{\mathbf{w}}^t A$

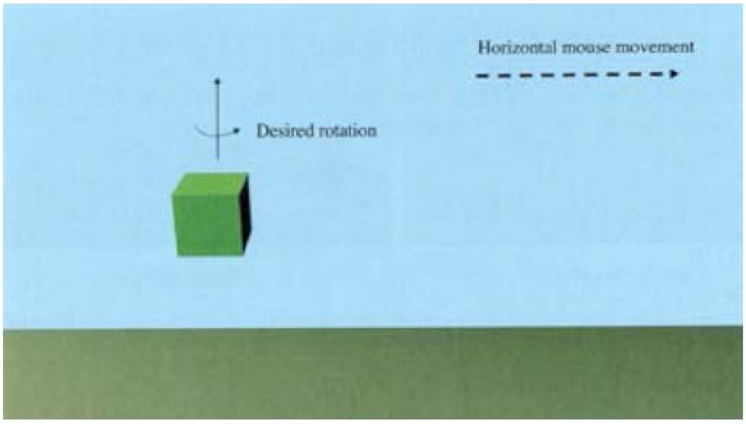
$$\vec{\mathbf{o}}^t$$
$$= \vec{\mathbf{w}}^t O$$
$$= \vec{\mathbf{a}}^t A^{-1} O$$
$$\Rightarrow \vec{\mathbf{a}}^t M A^{-1} O$$
$$= \vec{\mathbf{w}}^t A M A^{-1} O.$$

- What is the best frame $\vec{\mathbf{a}}^t$ to do this?

Moving an Object

- What if we choose $\vec{\mathbf{o}}^t$
- we transform this object with respect to $\vec{\mathbf{o}}^t$ rather than with respect to our observation through the window.
- What if we transform $\vec{\mathbf{o}}^t$ with respect to $\vec{\mathbf{e}}^t$
- we will rotate around the origin of the eye's frame $\vec{\mathbf{e}}^t$ (it appears to orbit around the eye).
- Then what frame it should be?

Moving an Object



- We actually want two different operations
  1. to transform (rotate) the object at its origin
  2. but the rotation axis should be the *y* axis of the eye.

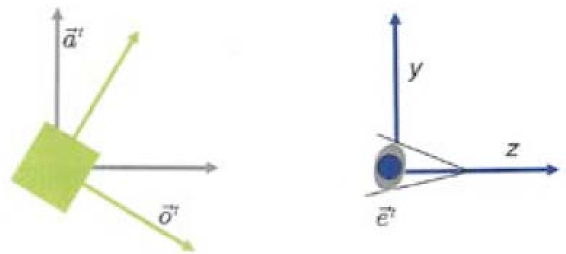
How to move an Object

- Recalling the Affine transform.: $A = TR$
- The object's Affine transform.: $O = (O)_T (O)_R$
  (we want the object's rotation about the object's origin)
- The eye's Affine transform.: $E = (E)_T (E)_R$
  (we want the object's rotation about the eye's y axis)
- The desired auxiliary frame $\vec{\mathbf{a}}^t$
  (imagine in a inverse way):

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

$$A = (O)_T (E)_R$$

From the left, we translate the world frame t the center of the object's frame, and then rotating the object's frame about that point to align with the directions of the eye.



Moving the eye

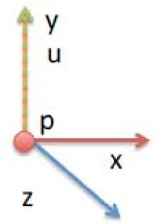
- We use the same auxiliary coordinate system.
- But in this case, the eye would orbit around the center of the object.
- Apply an affine transform directly to the eye's own frame (turning one's head, first-person motion)

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

$$E \leftarrow EM$$

The eye matrix (camera transform)

- Specifying the eye matrix $\vec{\mathbf{e}}^t = \vec{\mathbf{w}}^t E$ by:
  - the eye point $\tilde{p}$
  - the view point (where the eye looks at) $\tilde{q}$
  - the up vector $\vec{u}$



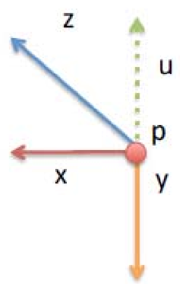
$$\mathbf{z} = normalize(p - q)$$
$$\mathbf{x} = normalize(\mathbf{u} \times \mathbf{z})$$
$$\mathbf{y} = \mathbf{z} \times \mathbf{x}$$
$$normalize(\mathbf{c}) = \mathbf{c} / \sqrt{c_1^2 + c_2^2 + c_3^2}$$

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

The view matrix (gluLookAt)

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

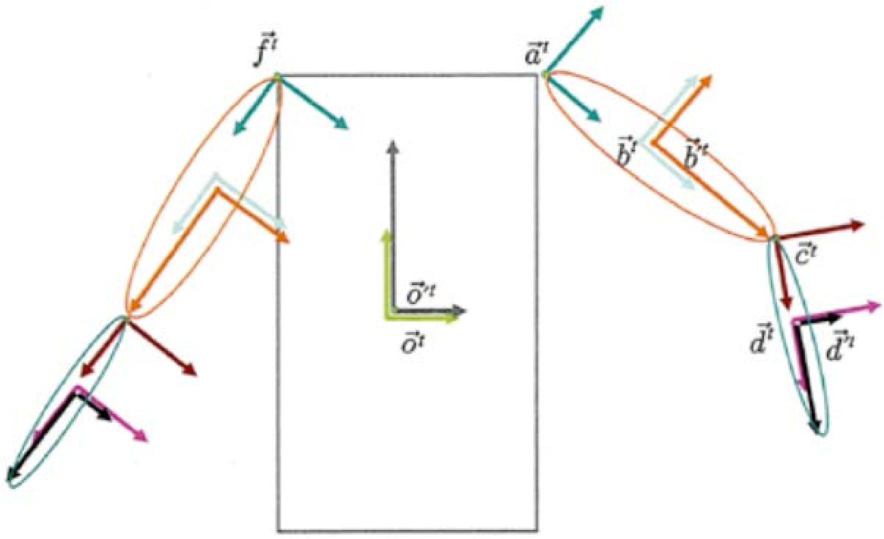
- Specifying the view matrix $V = E^{-1}$
  - the eye point $\tilde{p}$
  - the view point (where the eye looks at) $\tilde{q}$
  - the up vector $\vec{u}$



$$\mathbf{z} = normalize(q - p)$$
$$\mathbf{x} = normalize(\mathbf{u} \times \mathbf{z})$$
$$\mathbf{y} = \mathbf{x} \times \mathbf{z}$$
$$normalize(\mathbf{c}) = \mathbf{c} / \sqrt{c_1^2 + c_2^2 + c_3^2}$$

Hierarchy frames

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



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

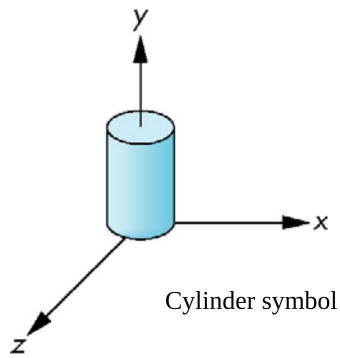
Modeling

- The spatial description and placement of imaginary 3D objects, environments and scene with a computer system.

- Models are abstractions of the world
- In computer graphics, we model our worlds with *geometric objects*.
  - Which primitive to use in our models?
  - How to show relationships among them?

Symbols and Instances

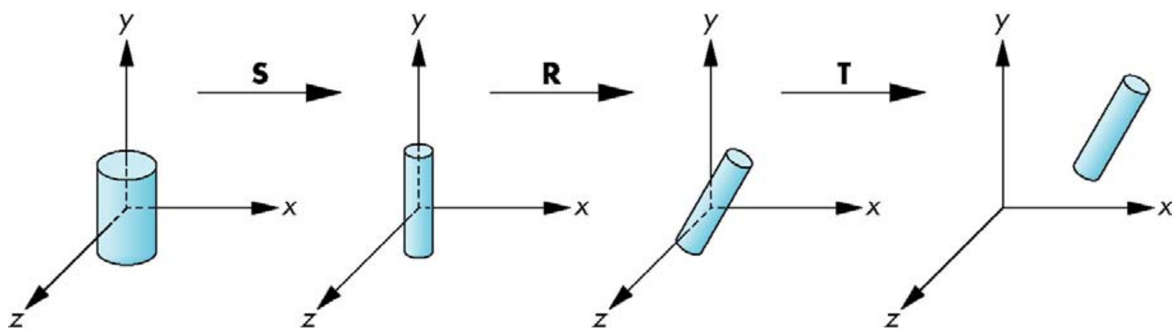
- Symbols
  - What are they?
    - primitives in the graphics library (polygon, line, cube, cylinder, …)
    - fonts
    - application-dependent graphic objects (e.g., circuit design symbols)
  - How they are represented?
    - at a convenient size and orientation



Cylinder symbol

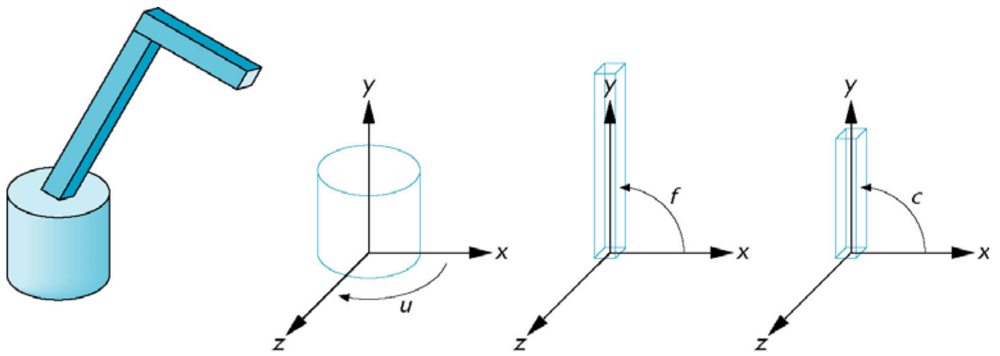
Symbols and Instances

- Instances
  - Instances of each symbol in the model are placed a the desired location with the desired size and orientation
  - By the instance *transformation*
    M = TRS



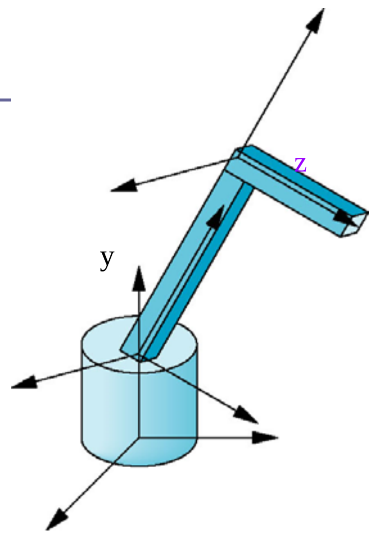
Hierarchical Models: A Robot Arm

- 3 parts
- 3 degrees of freedom
  - 2 for a joint angle between components
  - 1 by an angle the base makes with respect to ground)



(a) (b)

A Robot Arm

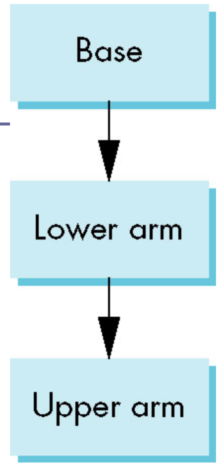


- Movement of robot components
  - Base:
    - rotate about y-axis in its frame by θ,
    - $\mathbf{R_y(\theta)}$
  - Lower arm:
    - rotate about z-axis in its own frame $R_z(\phi)$,
    - but this frame must be shifted to the top of the base by a translation matrix $T(0,h_1,0)$
    - $\mathbf{R_y(\theta)\ T(0,h_1,0)\ R_z(\phi)}$

positions the lower arm relative to the world frame

  - Upper arm:
    - translated by $T(0,h_2,0)$ relative to the lower arm, and then rotated by $R_z(\psi)$
    - $\mathbf{R_y(\theta)\ T(0,h_1,0)\ R_z(\phi)\ T(0,h_2,0)\ R_z(\psi)}$

A Robot Arm

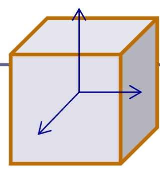



```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    model_view = RotateY(theta[0]);
    base();
    model_view = model_view *
          Translate(0.0, BASE_HEIGHT, 0.0) *
          RotateZ(theta[1]);
    lower_arm();
    model_view = model_view *
          Translate(0.0, LOWER_ARM_HEIGHT, 0.0) *
          RotateZ(theta[2]);
    upper_arm();

    glutSwapBuffer();
}
```

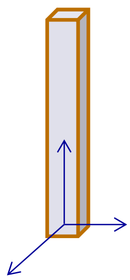
$$R_y(q)\ T(0,h_1,0)\ R_z(f)\ T(0,h_2,0)\ R_z(\psi)$$

A Robot Arm



```
mat4 instance;
mat4 model_view;
```

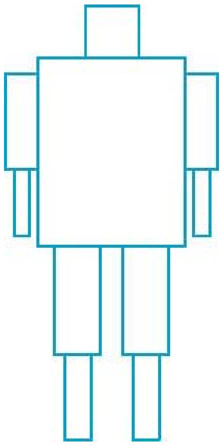
```
void base()
{
  instance = Translate(0.0, 0.5*BASE_HEIGHT, 0.0) *
             Scale(BASE_WIDTH, BASE_HEIGHT, BASE_WIDTH);
  glUniformMatrix4fv(model_view_loc, 16, GL_TRUE,
                     model_view*instance);
  glDrawArrays(GL_TRIANGLES, 0, N);
}
```

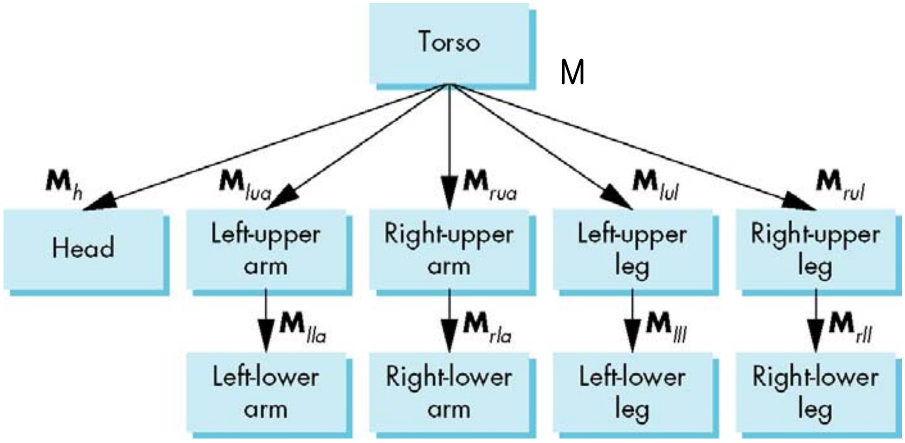


```
Void upper_arm()
{
   instance = Translate(0.0, 0.5*UPPER_ARM_HEIGHT, 0.0) *
     Scale(UPPER_ARM_WIDTH, UPPER_ARM_HEIGHT, UPPER_ARM_WIDTH);
   glUniformMatrix4fv(model_view_loc, 16, GL_TRUE,
                      model_view*instance);
   glDrawArrays(GL_TRIANGLES, 0, N);
}
```

Trees and Traversal

- Example: a humanoid figure

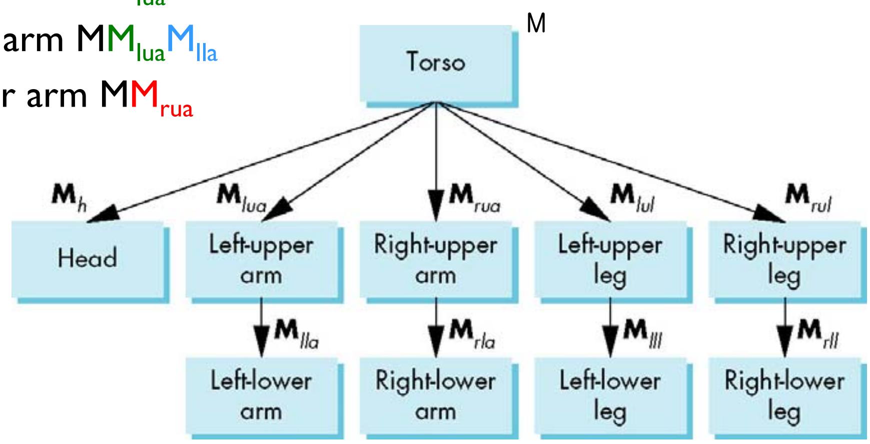




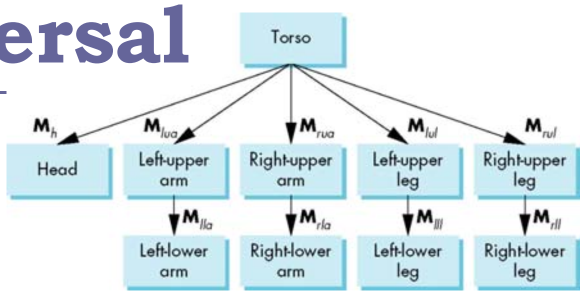
- How to traverse the tree to draw the figure?
  - left to right, depth first (pre-order traversal)

Trees and Traversal

- A stack-based traversal
  - torso: model-view matrix M
  - head: MM_h
  - left-upper arm MM_{lua}
  - left-lower arm $MM_{lua}M_{lla}$
  - right-upper arm MM_{rua}



Trees and Traversal



- A stack-based traversal

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

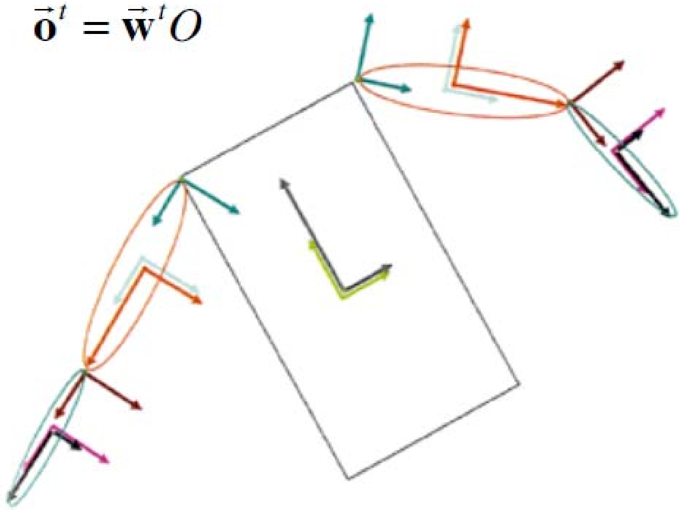
- ```
  figure() {
      mvstack.push(mv);
      torso();
      mv = mv * Translate()*
               Rotate();
      head();
      mv = mvstack.pop();
      mvstack.push(mv);
      mv = mv * Translate()*
              Rotate();
      left_upper_arm();
  ```

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


Moving the entire robot

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

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

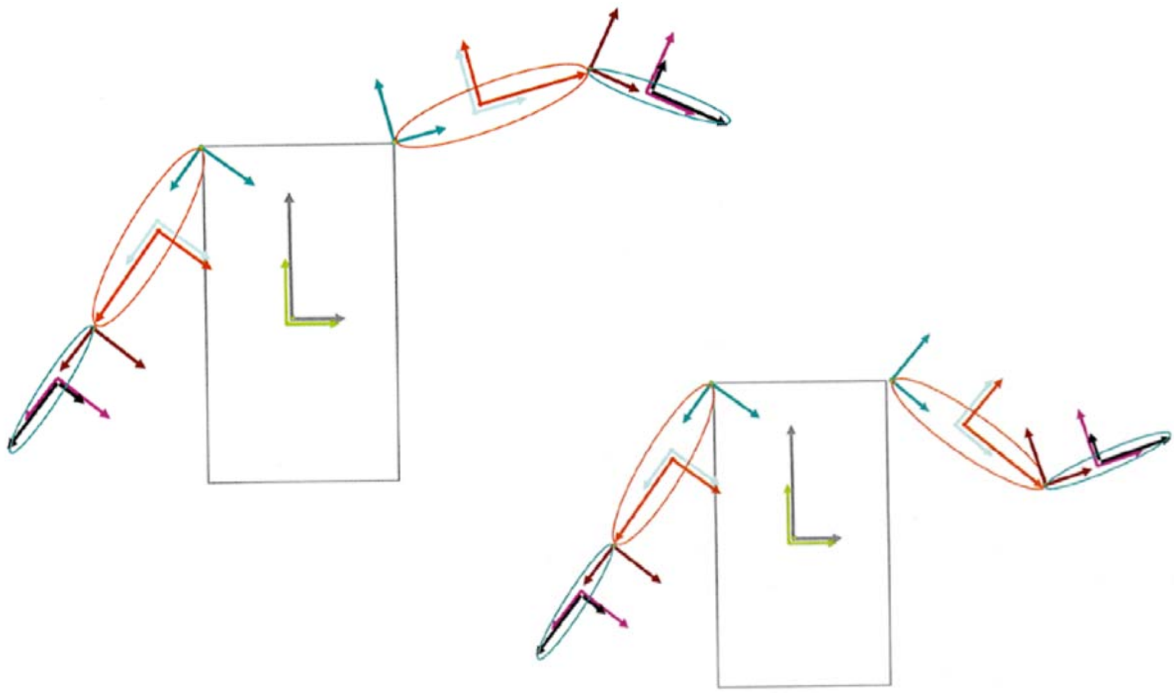


$$\vec{\mathbf{o}}^t = \vec{\mathbf{w}}^t O$$
$$\vec{\mathbf{a}}^t = \vec{\mathbf{w}}^t OA$$
$$\vec{\mathbf{b}}^t = \vec{\mathbf{w}}^t OAB$$
$$\vec{\mathbf{b}}'^t = \vec{\mathbf{w}}^t OABB'$$
$$\vec{\mathbf{c}}^t = \vec{\mathbf{w}}^t OABC$$
$$\vec{\mathbf{d}}^t = \vec{\mathbf{w}}^t OABCD$$
$$\vec{\mathbf{d}}'^t = \vec{\mathbf{w}}^t OABCDD'$$
```

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



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);
```


Chapter 6

HELLO WORLD 3D