CS380: Introduction to Computer Graphics
Frames in Graphics
Chapter 5

Min H. Kim
KAIST School of Computing

Honesty policy

- [Warning] Cheating is strongly forbidden!
  Cheating on homework or exams will give an **F**!

Respect

SUMMARY

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

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

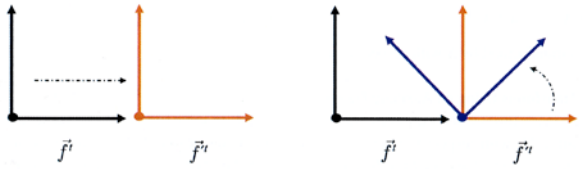
  $\vec{\mathbf{f}}^t$ is transformed by S with respect to $\vec{\mathbf{f}}^t$
- We read

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

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

Local Transformations

- Local transformations $\vec{\mathbf{f}}^t \Rightarrow \vec{\mathbf{f}}^t TR$



(a) Local translation (a) Local rotation

- In the first step,

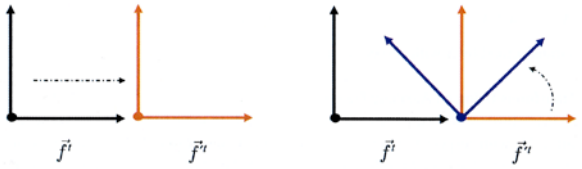
$\vec{\mathbf{f}}^t \Rightarrow \vec{\mathbf{f}}^t T = \vec{\mathbf{f}}'^t$

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

as the resulting frame: $\vec{\mathbf{f}}'^t$

Local Transformations

- Local transformations $\vec{\mathbf{f}}^t \Rightarrow \vec{\mathbf{f}}^t TR$



(a) Local translation (a) Local rotation

- In the second step,

$\vec{\mathbf{f}}^t \Rightarrow \vec{\mathbf{f}}^t TR,$

$\vec{\mathbf{f}}^t \Rightarrow \vec{\mathbf{f}}'^t R.$

$\vec{\mathbf{f}}^t$ is transformed by R with respect to $\vec{\mathbf{f}}'^t$

Global Transformations

- Global transformations $\vec{\mathbf{f}}^t \Rightarrow \vec{\mathbf{f}}^t TR$

(c) Global rotation (c) Global translation

- In the first step (in the reverse order)

$\vec{\mathbf{f}}^t \Rightarrow \vec{\mathbf{f}}^t R = \vec{\mathbf{f}}^{\circ t}$

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

as the resulting frame: $\vec{\mathbf{f}}^{\circ t}$

Global Transformations

- Global transformations $\vec{\mathbf{f}}^t \Rightarrow \vec{\mathbf{f}}^t TR$

(c) Global rotation (c) Global translation

- In the second step

$\vec{\mathbf{f}}^{\circ t} = \vec{\mathbf{f}}^t R \Rightarrow \vec{\mathbf{f}}^t TR$

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

Chapter 5

FRAMES IN GRAPHICS

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

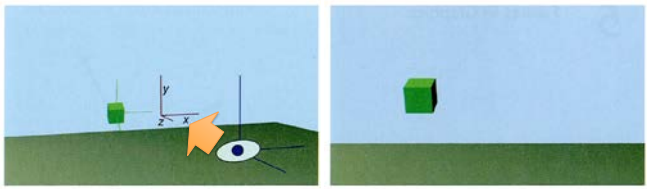
- The relationship between the world frame and object frame:
  - affine 4-by-4 matrix O (rigid body transformation: rotation + translation only)

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

- 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.

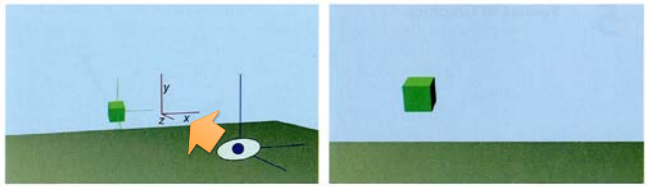


(a) The frames (b) The eye's view

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

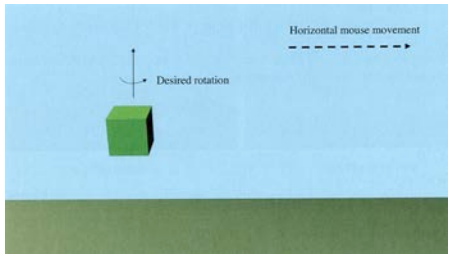
- we explicitly store the matrix E $\quad \vec{\mathbf{e}}^t = \vec{\mathbf{w}}^t E$

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

  - 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?

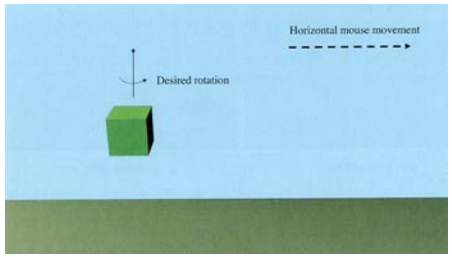
Moving an Object

- 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.

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.

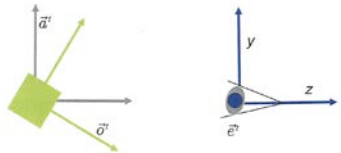
How to move an Object

- 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)

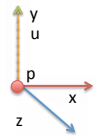
$$\vec{\mathbf{e}}^t = \vec{\mathbf{w}}^t E,$$
$$E \leftarrow EM$$

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 $\vec{u}$

NB P. 35 contains errors (see errata)!!!

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



NB matrix sent to the vertex shader is *E^{-1}*O*

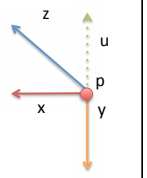
$$E = \begin{bmatrix} x_1 & y_1 & z_1 & p_1 \\ x_2 & y_2 & z_2 & p_2 \\ x_3 & y_3 & z_3 & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The view matrix (gluLookAt)

- 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 $\vec{u}$

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

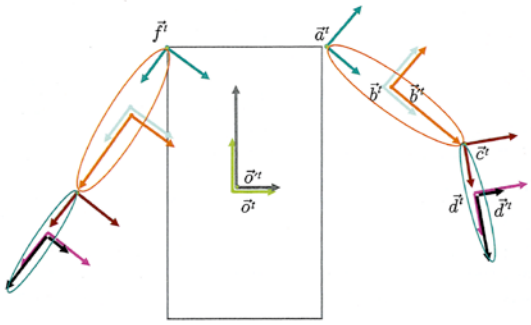
$$\mathbf{z} = normalize(q - p)$$
$$\mathbf{x} = normalize(\mathbf{u} \times \mathbf{z})$$
$$\mathbf{y} = \mathbf{x} \times \mathbf{z}$$
$$normalize(\mathbf{c}) = \mathbf{c} / \sqrt{c_1^2 + c_2^2 + c_3^2}$$



NB gluLookAt is not the part of modern OpenGL!

Hierarchy frames

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

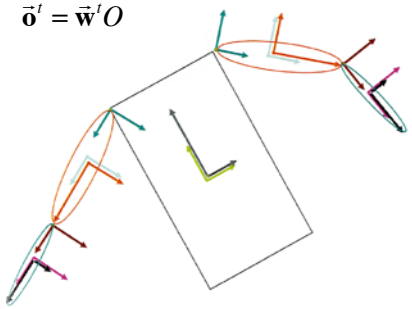


$$\vec{\mathbf{o}}^t = \vec{\mathbf{w}}^t O$$
$$\vec{\mathbf{o}}'^t = \vec{\mathbf{o}}^t O'$$
$$\vec{\mathbf{a}}^t = \vec{\mathbf{o}}^t A$$
$$\vec{\mathbf{b}}^t = \vec{\mathbf{a}}^t B$$
$$\vec{\mathbf{b}}'^t = \vec{\mathbf{b}}^t B'$$
$$\vec{\mathbf{c}}^t = \vec{\mathbf{b}}^t C$$
$$\vec{\mathbf{d}}^t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

Moving the entire robot

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

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



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

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

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

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

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

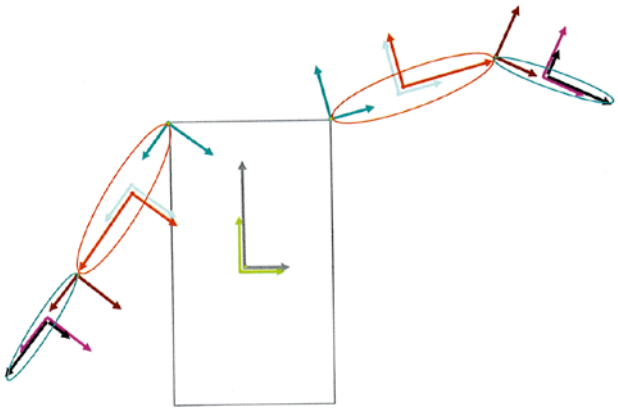
$\vec{\mathbf{d}}^t = \vec{\mathbf{w}}^t OABCD$

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

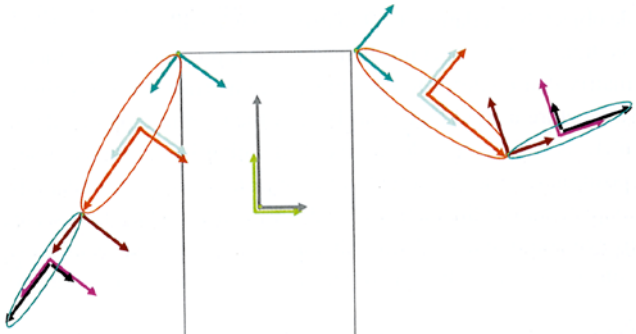
Matrix stack

- 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);
…
```