Transformations & Matrices

Introduction to Computer Graphics
CS 351-50

Quiz #2 Review

- Dot Product

$a \cdot b = |a|\,|b| \cos\theta = a_x b_x + a_y b_y + a_z b_z$

$a \cdot b = a^T b$

thus

$$[a_x \;\; a_y \;\; a_z] \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = a_x b_x + a_y b_y + a_z b_z$$

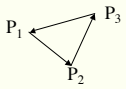
Quiz #2 Review

- Cross Product & Normals
- Points to Vectors:

$P_{tip} - P_{tail}$

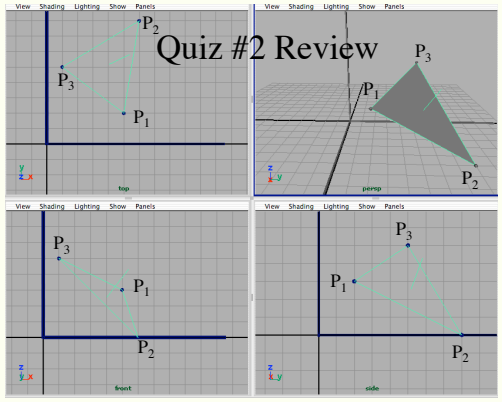
  - Point ordering for triangles

$N = (P_2 - P_1) \times (P_3 - P_2)$
$= (P_3 - P_2) \times (P_1 - P_3)$
$= (P_1 - P_3) \times (P_2 - P_1)$

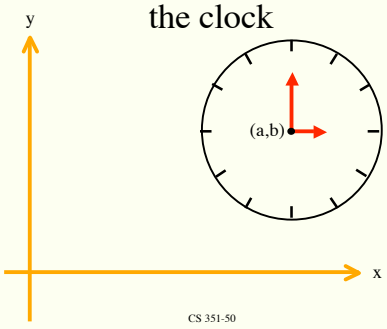


  - OpenGL assumes Counter-Clockwise Order

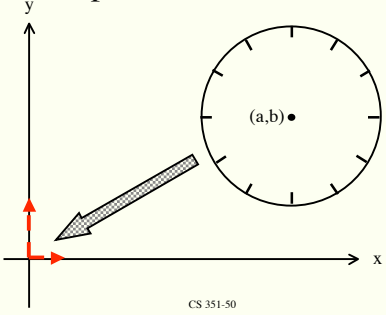
Quiz #2 Review



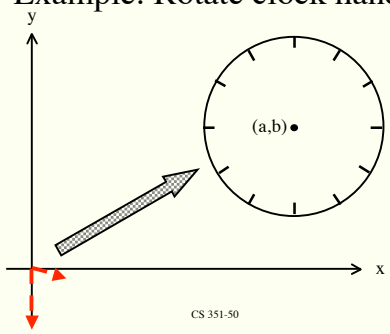
Example: Change the time on the clock



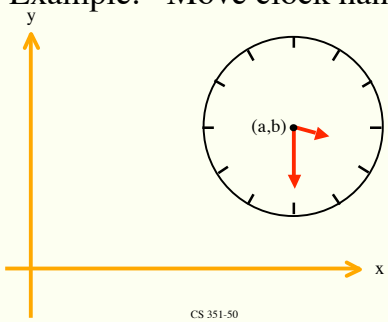
Example: Move clock hands



Example: Rotate clock hands



Example: Move clock hands



Clock Transformations

- Translate to Origin
- Move hand with rotation
- Move hand back to clock
- Do other hand

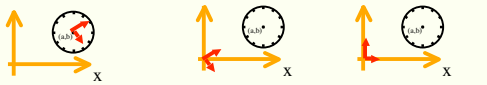
Clock Transformations

$M = T(a, b)\ R(t)\ T(-a, -b)$

1rst: Translate to Origin

2nd: Rotate hands

3rd: Translate back to clock



Translations

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale

$$S = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad S^{-1} = \begin{bmatrix} 1/x & 0 & 0 & 0 \\ 0 & 1/y & 0 & 0 \\ 0 & 0 & 1/z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

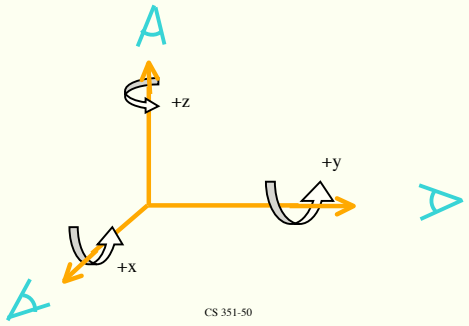
Rotations

Rotate(q, 1, 0, 0) = $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotate(q, 0, 1, 0) = $\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotate(q, 0, 0, 1) = $\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

3D Positive Rotations



More on 2D Transformations

Think of a bunny (one we are drawing to a screen)
- Where is she?
- Where is she facing?
- How big is she?

Think of drawing this bunny's movement

2D Transformations

This state information
- position, facing, scaling

determines where to draw the bunny

Think of our representation of this bunny as Matrix M

OpenGL will apply M to all of the vertices of bunny if we load M into OpenGL MODELVIEW Matrix

Encode state in Matrix

Bunny at Initial State
- Position = 0,0
- Facing = up (Y)

Temporary matrix:
- TempM1, TempM2 = identity

SaveViewMat = identity
CurrentViewMatrix = identity

Example: Move the bunny

Want to move bunny to :

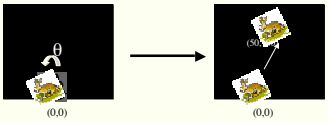
(50,100)

facing 30 degrees to left

- Two good ways to think about how to get there
  - User's point of view
    - (world coordinates)
  - From the bunny's point of view
    - (object coordinates)
    - Bunny follows directions we give her

Example: Move the bunny

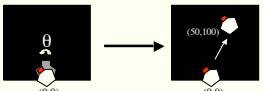
- User's point of view
  - (world coordinates)
  - Rotate bunny around (0,0) until she is facing 30 degrees ($\theta = \pi/3$) to left, then Translate (0,0) to (50,100)



$$\begin{bmatrix} 1 & 0 & 50 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 50 \\ \sin\theta & \cos\theta & 100 \\ 0 & 0 & 1 \end{bmatrix} = T_{(50,100)}\, R_\theta$$

Works same for any geometry

- User's point of view
  - (world coordinates)
  - Rotate around (0,0) facing 30 degrees ($\theta = \pi/3$) to left, then Translate (0,0) to (50,100)



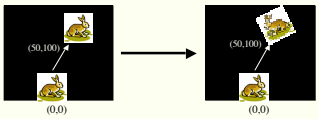
Note: write down matrix in opposite order in which they are applied from the user's point of view

$$\begin{bmatrix} 1 & 0 & 50 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 50 \\ \sin\theta & \cos\theta & 100 \\ 0 & 0 & 1 \end{bmatrix} = T_{(50,100)}\, R_\theta$$

From the point of view of bunny

- Give her directions
  - Walk from (0,0) to (50,100)
  - Rotate 30 degrees to left ($\theta = \pi/3$)

Note: Transformations occur in the same order in which the matrices are written down.

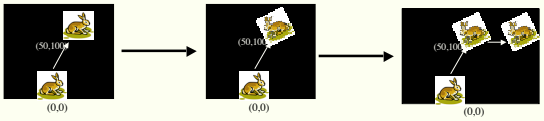


$$\begin{bmatrix} 1 & 0 & 50 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 50 \\ \sin\theta & \cos\theta & 100 \\ 0 & 0 & 1 \end{bmatrix} = T_{(50,100)}\, R_\theta$$

From the point of view of bunny

- Move her another hop:

Note: Post multiply M by another transformation matrix



$$\begin{bmatrix} \cos\theta & -\sin\theta & 50 \\ \sin\theta & \cos\theta & 100 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 30 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 50 + 30\cos\theta \\ \sin\theta & \cos\theta & 100 + 30\sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

Example Code: Movement 1

```
MyVec4f new_origin(0,0,0,1);
float init_rot = 0;

tempMat1.identity();
tempMat2.identity();
currentMat.identity();
tempMat1.setTranslation(new_origin);
tempMat1.print("tempMat1 Identity");
tempMat2.rot_z(init_rot);
tempMat2.print("tempMat2 Rotate by 0");

currentMat=tempMat2*tempMat1;
currentMat.print("current = tempMat1 * tempMat2\n");
load4DMatrix(currentMat);
drawBunnyPoly();
```

```
tempMat1 Identity
        1.00 0.00 0.00 0.00
        0.00 1.00 0.00 0.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00

Rotate z: 0.00 radians or 0.00 degrees
tempMat2 Rotate by 0
        1.00 0.00 0.00 0.00
        0.00 1.00 0.00 0.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00

current = tempMat1 * tempMat2

        1.00 0.00 0.00 0.00
        0.00 1.00 0.00 0.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00
```

Example Code: Movement 2

```
new_origin.x(20);
new_origin.y(30);
new_origin.z(0);
new_origin.w(1);

tempMat1.identity();
tempMat2.identity();
tempMat1.setTranslation(new_origin);
tempMat2.print("tempMat1  = Translate  (20,30,0,1)");
tempMat2 = currentMat;
tempMat2.print("tempMat2 = current Matrix");
currentMat=tempMat2*tempMat1;
if (DEBUG_PRINT) {printf("current = temp2 *
   temp1\n");  currentMat.print();}
load4DMatrix(currentMat);
drawBunnyPoly();
```

```
tempMat1  = Translate
(20,30,0,1)
        1.00 0.00 0.00 20.00
        0.00 1.00 0.00 30.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00

tempMat2 = current Matrix
        1.00 0.00 0.00 0.00
        0.00 1.00 0.00 0.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00

current = temp2 * temp1
        1.00 0.00 0.00 20.00
        0.00 1.00 0.00 30.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00
```

Example Code: Movement 3

```
float new_rot = M_PI/6;

tempMat1.identity();
tempMat2.identity();
tempMat1.rot_z(new_rot);
tempMat1.print("tempMat1 = rotate by PI/6");
tempMat2 = currentMat;
tempMat2.print("tempMat2 = current Matrix");
currentMat=tempMat2*tempMat1;
currentMat.print("current = temp2*temp1");
load4DMatrix(currentMat);
drawBunnyPoly();
```

```
Rotate z: 0.52 radians or 30.00 degrees
tempMat1 = rotate by PI/6
        0.87 -0.50 0.00 0.00
        0.50 0.87 0.00 0.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00

tempMat2 = current Matrix
        1.00 0.00 0.00 20.00
        0.00 1.00 0.00 30.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00

current = temp2*temp1
        0.87 -0.50 0.00 20.00
        0.50 0.87 0.00 30.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00
```

Example Code: Movement 4

```
new_origin.x(30);
new_origin.y(0);
new_origin.z(0);
new_origin.w(1);

tempMat1.identity();
tempMat2.identity();
tempMat1.setTranslation(new_origin);
tempMat1.print("tempMat1  = Translate by
  30,0,0,1");
tempMat2 = currentMat;
tempMat2.print("tempMat2 = currentMat");
currentMat=tempMat2*tempMat1;
if (DEBUG_PRINT) {printf("current =temp1 *
  temp2\n");  currentMat.print();}
load4DMatrix(currentMat);
drawBunnyPoly();
```

```
tempMat1  = Translate by 30,0,0,1
        1.00 0.00 0.00 30.00
        0.00 1.00 0.00 0.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00

tempMat2 = currentMat
        0.87 -0.50 0.00 20.00
        0.50 0.87 0.00 30.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00

current =temp1 * temp2
        0.87 -0.50 0.00 45.98
        0.50 0.87 0.00 45.00
        0.00 0.00 1.00 0.00
        0.00 0.00 0.00 1.00
```

R T Not Always Equal to T R

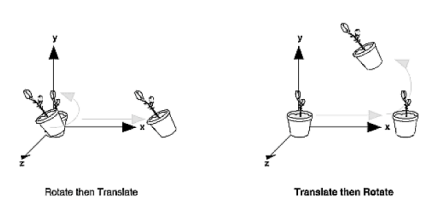


Figure 3-4 : Rotating First or Translating First

Matrix & 2D Transforms

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(N); /* apply transformation N */
glMultMatrixf(M); /* apply transformation M */
glMultMatrixf(L); /* apply transformation L */
glBegin(GL_POINTS);
glVertex3f(v); /* draw transformed vertex v */
glEnd();
```

Result: I N M L = NML = N(M(Lv))

Summary

- Either think in terms of grand, fixed coordinate system
  - Order of operations is the opposite of order of how they appear in the code
  - Hard to do…

Summary

- Instead, think of a local coordinate system tied to the object you are drawing
  - Operations occur relative to this changing coordinate system
  - Matrix multiplications happen in natural order in the code
- In the end, the code is the same, you just have to think about it differently
- Read Chapter 3 of OpenGL book (especially the "Viewing and Modeling transformations" section)

Orthographic Viewing Matrix

```
//create a viewing volume, see pg 124 of
// OGL Programming book (Version1.1)
// note: numbers should be proportional to window size

glOrtho(-50.0/*left*/,
        50.0/*right*/,
        0.0/*bottom*/,
        80.0/*top*/,
        -1.0/*near*/,
        1.0/*far*/);
```

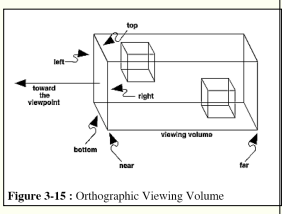


Figure 3-15 : Orthographic Viewing Volume

thus the center in screen x is 0, bottom in y is 0, top is 80;

L-systems

Handouts

http://ai.toastbrot.ch/life/intro.php