

University of British Columbia

## Chapter 3.5

### Transformations – OpenGL Composition of Transformations

## Transformations in OpenGL

- An easier way to do the same thing....

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glTranslatef(3,1,0);
glScale(2,2,2);

DrawHouse();
```

University of British Columbia

## Transformations in OpenGL

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glBegin(GL_LINE_LOOP);
glVertex2f(0,0);
glVertex2f(2,0);
glVertex2f(2,2);
glVertex2f(1,3);
glVertex2f(0,2);
glEnd();
```

} DrawHouse()

$$P' = PT_{MV}$$

University of British Columbia

## Matrix Operations in OpenGL

- 2 Matrices:
  - Model/view matrix M
  - Projective matrix P
- Example:

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity(); // M=Id
glRotatef( angle, x, y, z ); // M= R(alpha)*Id
glTranslatef( x, y, z ); // M= T(x,y,z)*R(alpha)*Id
glMatrixMode( GL_PROJECTION );
glRotatef( ... ); // P= ...
```

University of British Columbia

## Transformations in OpenGL

$$[x \ y \ z \ 1]_w = [x \ y \ z \ 1]_{obj} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 3 & 1 & 0 & 1 \end{bmatrix}$$

```
GLfloat T[16] = { 2,0,0,0, 0,2,0,0,
                 0,0,2,0, 3,1,0,1};

glMatrixMode(GL_MODELVIEW);
glLoadMatrixf(T);

DrawHouse();
```

University of British Columbia

## Composing Transformations

suppose we want

**Rotate(z,-90)**

$P_A = P_h Rot(z, -90)$

**Translate(2,3,0)**

$P_w = P_A Trans(2,3,0)$

$$P_w = P_h Rot(z, -90) Trans(2,3,0)$$

University of British Columbia

## Composing Transformations

$P_W = P_h Rot(z, -90) Trans(2, 3, 0)$

- L-to-R: interpret operations wrt fixed coords
- R-to-L: interpret operations wrt local coords
- OpenGL (R-to-L, local coords)


```

glTranslatef(2,3,0);
glRotatef(-90,0,0,1);
DrawHouse();
    
```

$$M_{MV} = Trans(2,3,0) \cdot M_{MV}$$

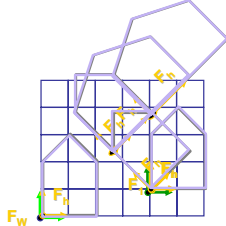
$$M_{MV} = Rot(z, -90) M_{MV}$$

updates current transformation matrix by postmultiplying




## Composing Transformations

- OpenGL example



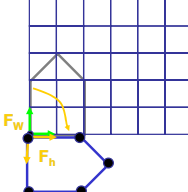
```

glLoadIdentity();
glTranslatef(4,1,0);
glPushMatrix();
glRotatef(45,0,0,1);
glTranslatef(0,2,0);
glScalef(2,1,1);
glTranslate(1,0,0);
glPopMatrix();
    
```

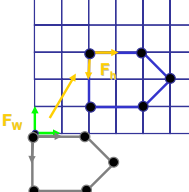


## Composing Transformations

Rotate(z, -90)




Translate(-3,2,0) in local coords



$$P_W = P_h Trans(-3,2,0) Rot(z, -90)$$

```

glRotatef(-90,0,0,1);
glTranslatef(-3,2,0);
draw_house();
    
```



## Transformation Hierarchies


- Matrix Stack

$D = C \text{ scale}(2,2,2) \text{ trans}(1,0,0)$

C	C	D	C
B	B	C	B
A	A	B	A

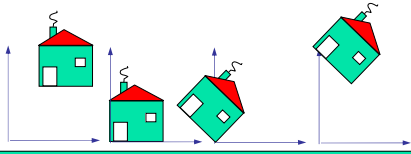
```


DrawSquare();
glPushMatrix();
glScale3f(2,2,2);
glTranslate3f(1,0,0);
DrawSquare();
glPopMatrix();
    
```



## Why


- Simplify local operations
- Avoid move to origin



$$T^{-1}(p_x, -p_y) R^\theta T^1(p_x, p_y) = \begin{bmatrix} 1 & 0 & 0 & \cos\theta & \sin\theta & 0 \\ 0 & 1 & 0 & -\sin\theta & \cos\theta & 0 \\ -p_x & -p_y & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ p_x & p_y & 1 \end{bmatrix}$$


## Matrix Stacks

- Means of returning to previously-used coordinate system
  - Support several models or model parts
    - Natural hierarchical structure
- depth of matrix stacks limited in hardware
  - typically: 16 for ModelView, 4 for Projection



## Transformation Hierarchies

```

graph TD
    world((world)) --> torso((torso))
    torso --> LUleg((LUleg))
    torso --> RUleg((RUleg))
    torso --> LUarm((LUarm))
    torso --> RUarm((RUarm))
    torso --> head((head))
    LUleg --> LLleg((LLleg))
    LUleg --> RLleg((RLleg))
    LLleg --> Lfoot((Lfoot))
    RLleg --> Rfoot((Rfoot))
    LUarm --> LLarm((LLarm))
    LUarm --> RLarm((RLarm))
    LLarm --> Lhand((Lhand))
    RLarm --> Rhand((Rhand))
    
```

trans(0.30,0,0) rot(z,θ)

University of British Columbia

## Transformation Hierarchies

■ Example

```

glTranslatef(x,y,0);
glRotatef(θ,0,0,1);
DrawBody();
glPushMatrix();
glTranslatef(0,7,0);
DrawHead();
glPopMatrix();
glPushMatrix();
glTranslatef(2.5,5.5,0);
glRotatef(θ2,0,0,1);
DrawUArm();
glTranslatef(0,-3.5,0);
glRotatef(θ3,0,0,1);
DrawLArm();
glPopMatrix();
... (draw other arm)
    
```

University of British Columbia

## Projective Rendering Pipeline

```

graph TD
    OCS --> modeling[modeling transformation]
    modeling --> WCS[WCS]
    WCS --> viewing[viewing transformation]
    viewing --> VCS[VCS]
    VCS --> projection[projection transformation]
    projection --> CCS[CCS]
    CCS --> division[perspective division]
    division --> NDCCS[NDCS]
    NDCCS --> viewport[viewport transformation]
    viewport --> DCS[DCS]
    
```

glVertex3f(x,y,z)

OCS modeling transformation WCS viewing transformation VCS projection transformation CCS perspective division NDCCS viewport transformation DCS

glTranslatef(x,y,z)  
glRotatef(th,x,y,z)  
gluLookAt(...)  
glFrustum(...)  
glutInitWindowSize(x,y)

OCS - object coordinate system  
WCS - world coordinate system  
VCS - viewing coordinate system  
CCS - clipping coordinate system  
NDCCS - normalized device coordinate system  
DCS - device coordinate system

University of British Columbia