Today's Outline

Purpose of transformations

How to use transformations?

- **Types: scales, rotations, translates, ...**
- **Composing multiple transformations**
- **Coordinate frames**
- **Hierarchical transformations**

How to implement transformations using matrices?

- **Representing transformations as matrices**
- **OpenGL implementation**

Transformations

What? Functions acting on points

$(x', y', z') = T(x, y, z)$ or $P' = T(P)$

Why?

Viewing

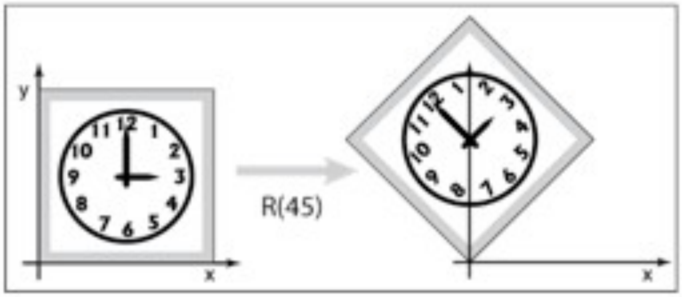
- Window coordinates to framebuffer coordinates
- Virtual camera: parallel/perspective projections

Modeling

- Create objects in convenient coordinates
- Multiple instances of a prototype shape
- Kinematics of linkages/skeletons - characters/robots

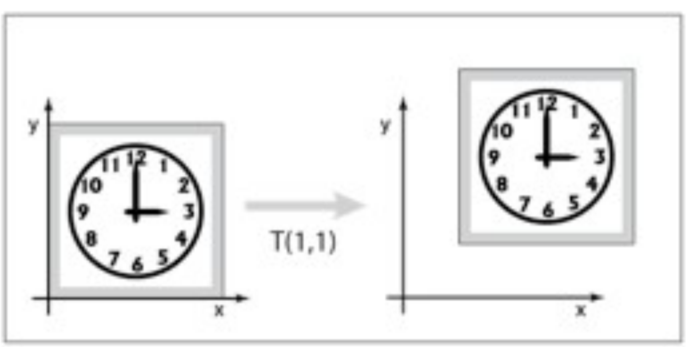
How to Use Transformations?

Rotate



glRotatef(angle,ax,ay,az)

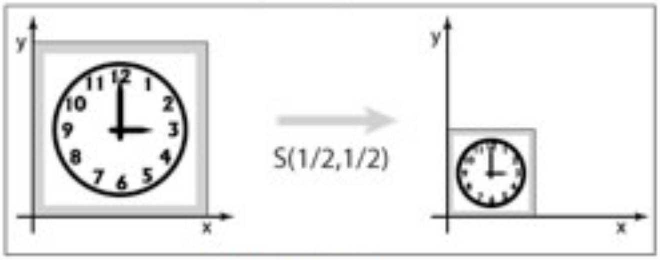
Translate



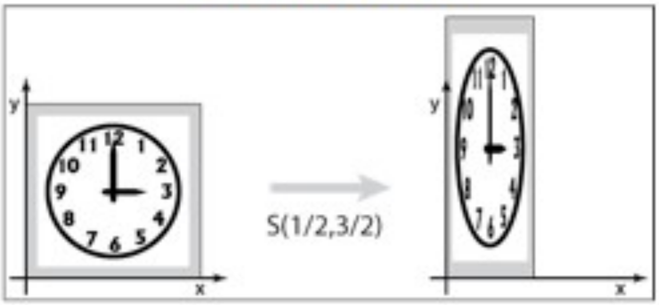
glTranslatef(tx,ty,tz)

Scale

Uniform



Nonuniform



glScalef(sx,sy,sz)

Composing Transformations

Rotate, Then Translate

R(45)

T(1,1) R(45)

Order of Transformations

The rightmost transform applied to the point first
That is, the rotate is applied before the translate

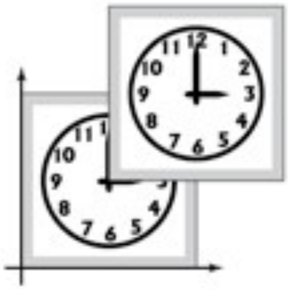
$P' = T(1,1)\,(R(45)(P))$

OpenGL

```
glTranslatef( 1.0, 1.0, 0.0 );
glRotatef( 45.0, 0., 0., 1. );
```

The last one is applied first; we'll see why latter

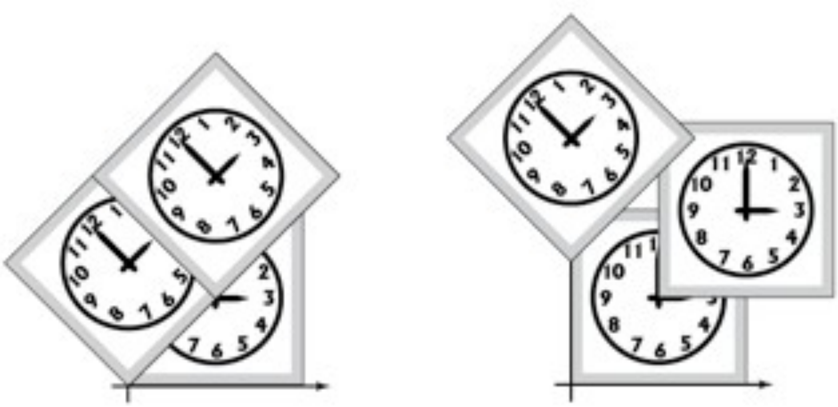
Translate, Then Rotate



T(1,1)

R(45) T(1,1)

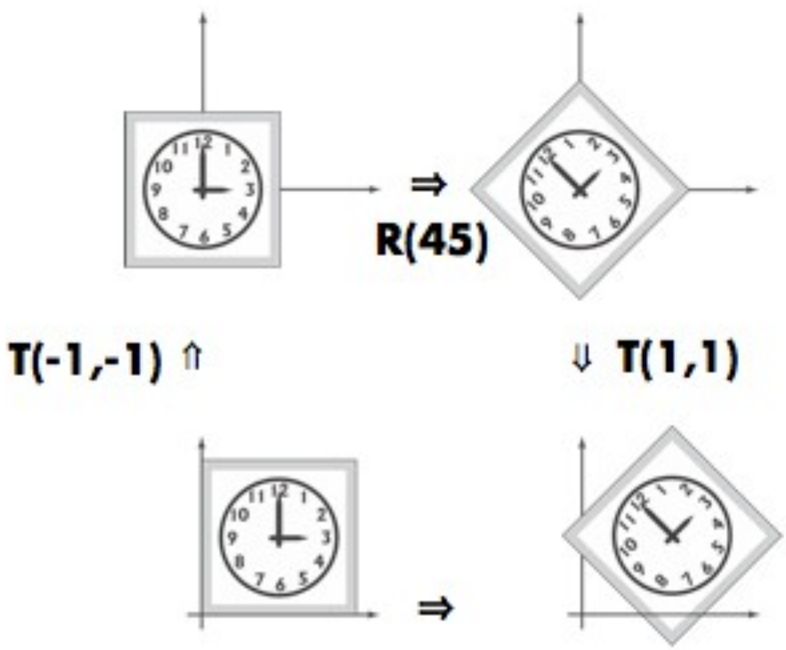
Order Matters



T(1,1) R(45) ≠ R(45) T(1,1)

Rotate 45 about the Center (1,1)?

Rotate 45 @ (1,1)

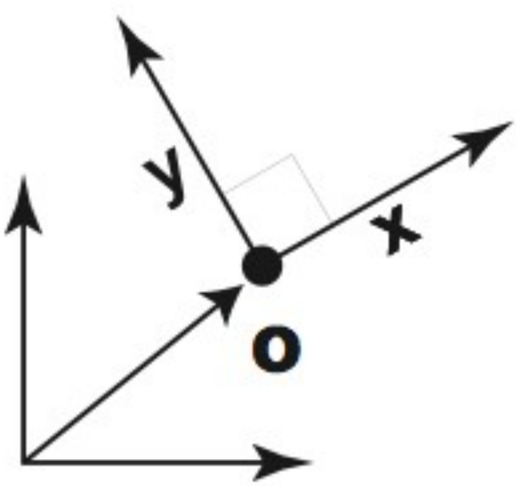


T(1,1) R(45) T(-1,-1)

CS148 Lecture 4 Pat Hanrahan, Fall 2010

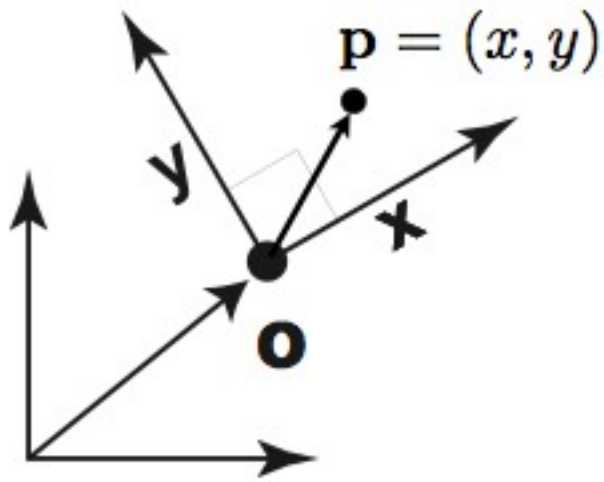
Coordinate Systems

Reference Frame



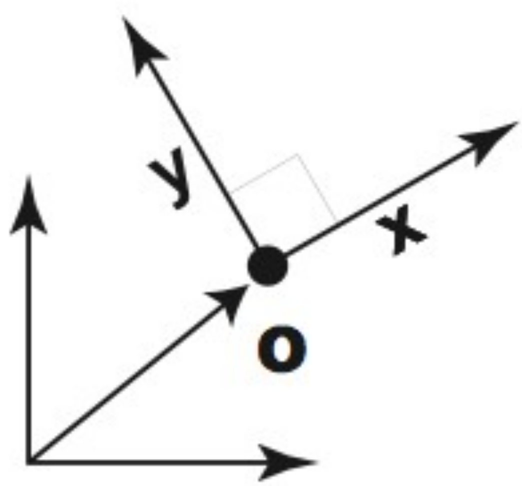
An origin o and two unit vectors x, y
define a frame of reference

Points Defined wrt to the Frame



$$\mathbf{p} = \mathbf{o} + x\,\mathbf{x} + y\,\mathbf{y}$$

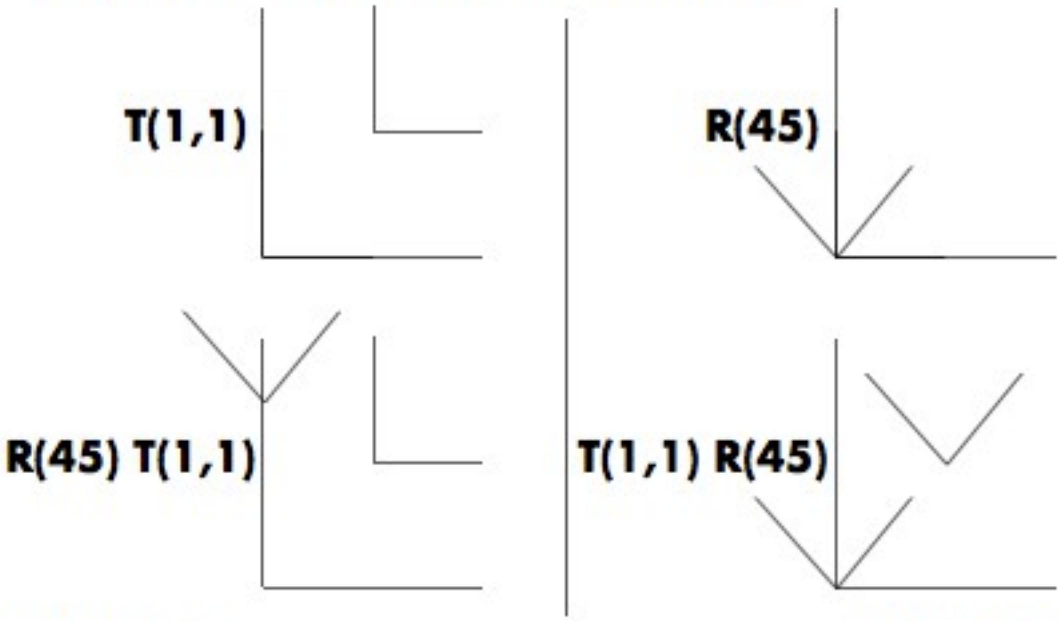
Transform the Frame



- Frame of reference defined by o, x, y (and z)
- Apply transformation to change the frame
- Points defined wrt to the frame also move

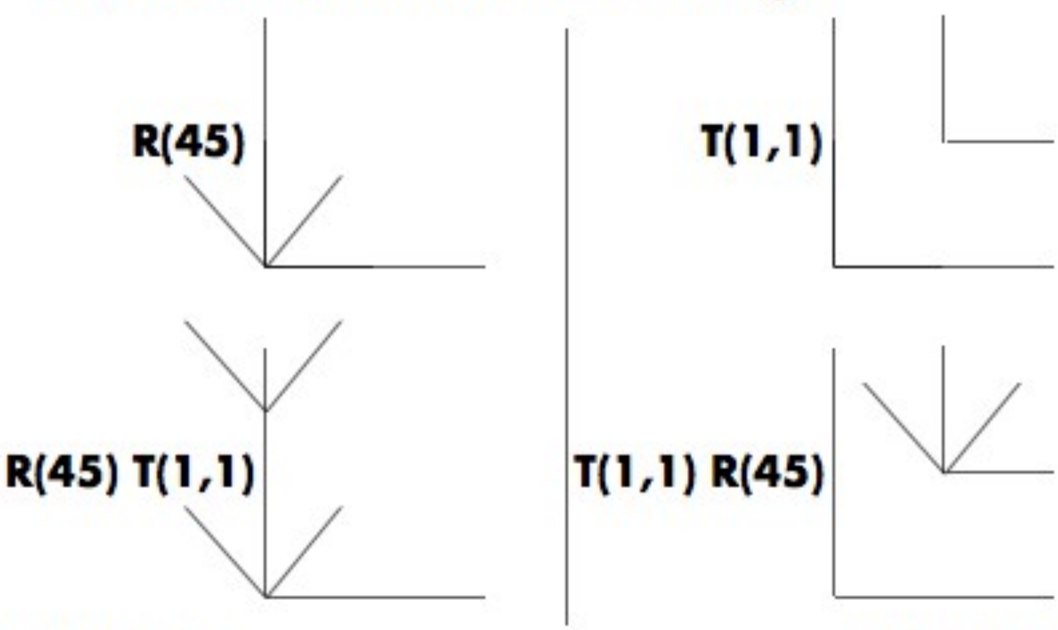
Transform in World Coordinates

Transform defined in world/global coordinates
Concatenate transforms on the left

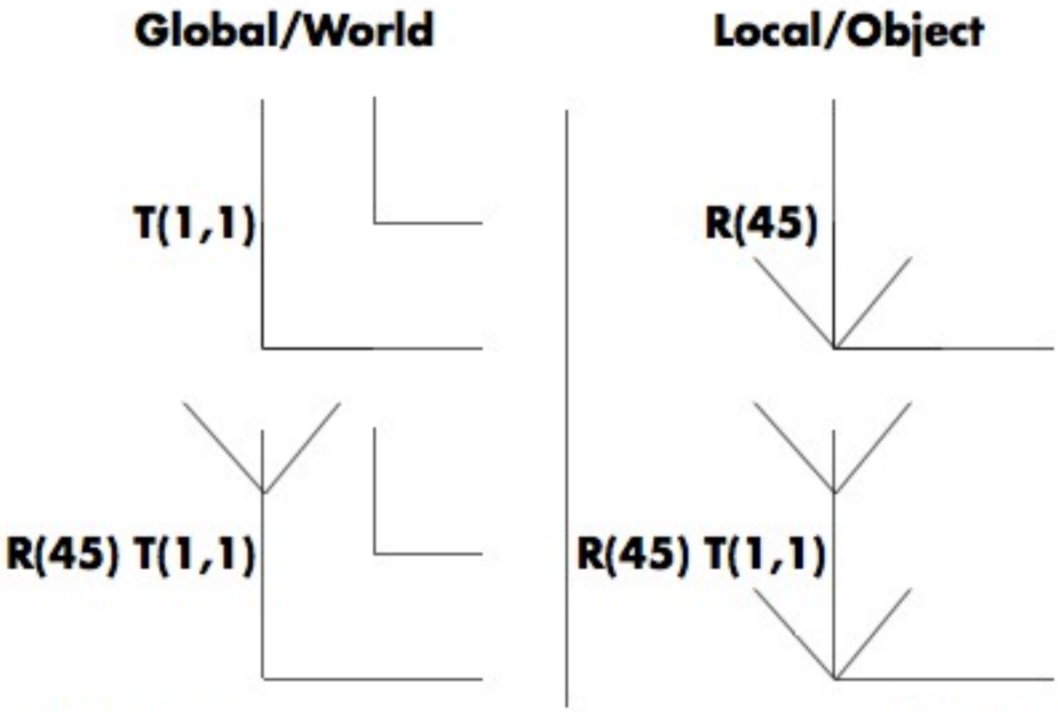


Transform in Object Coordinates

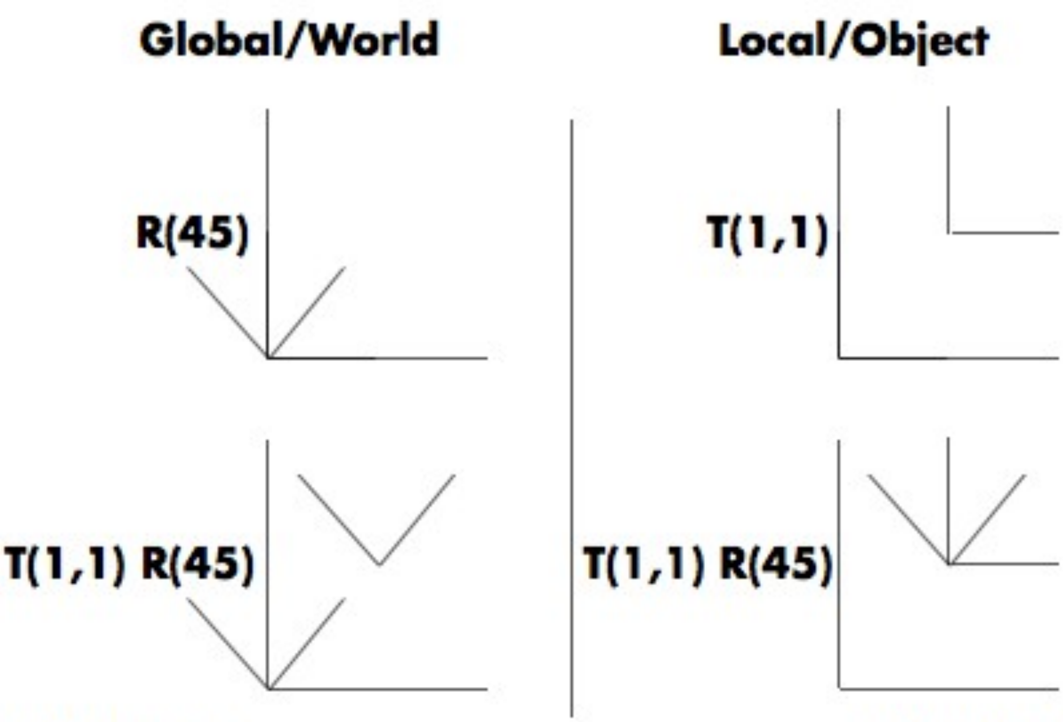
Transform defined in object/local coordinates
Concatenate transforms on the right



Two Interpretations are Equivalent

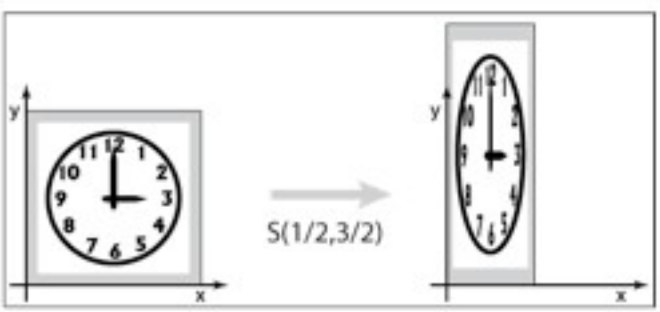


Two Interpretations are Equivalent



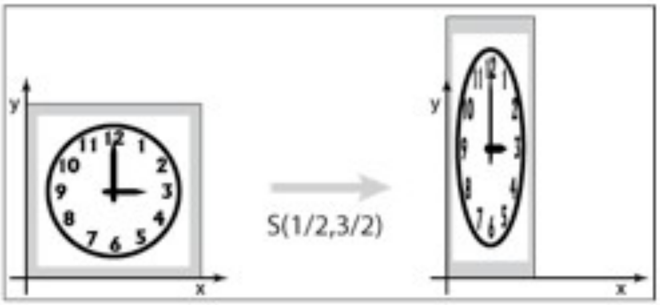
Mathematics of Linear Transformations and Matrices

Scale



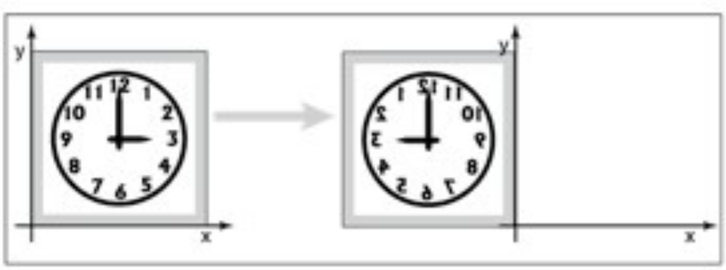
$$x' = s_x\, x$$
$$y' = s_y\, y$$

Scale



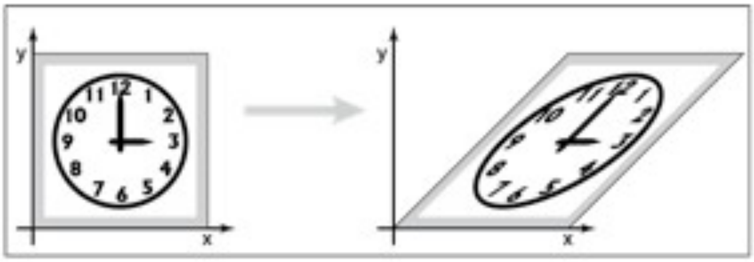
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Reflection Matrix?



$$x' = ?$$
$$y' = ?$$

Shear Matrix?



$$x' = ?$$
$$y' = ?$$

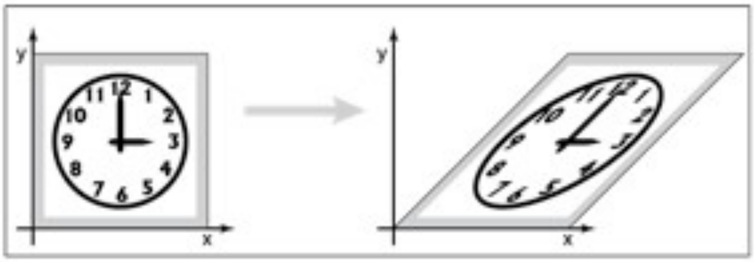
Linear Transformations = Matrices

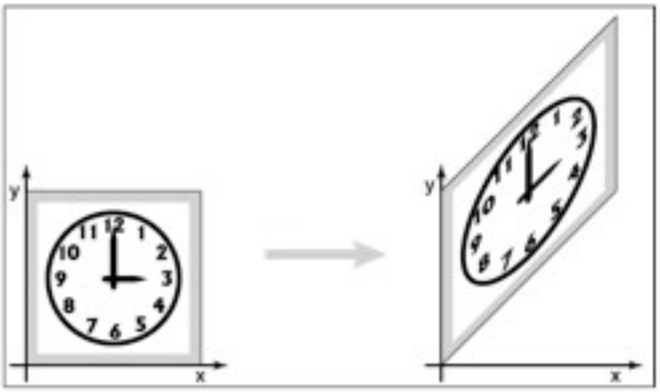
$$x' = m_{xx}\,x + m_{xy}\,y$$
$$y' = m_{yx}\,x + m_{yy}\,y$$

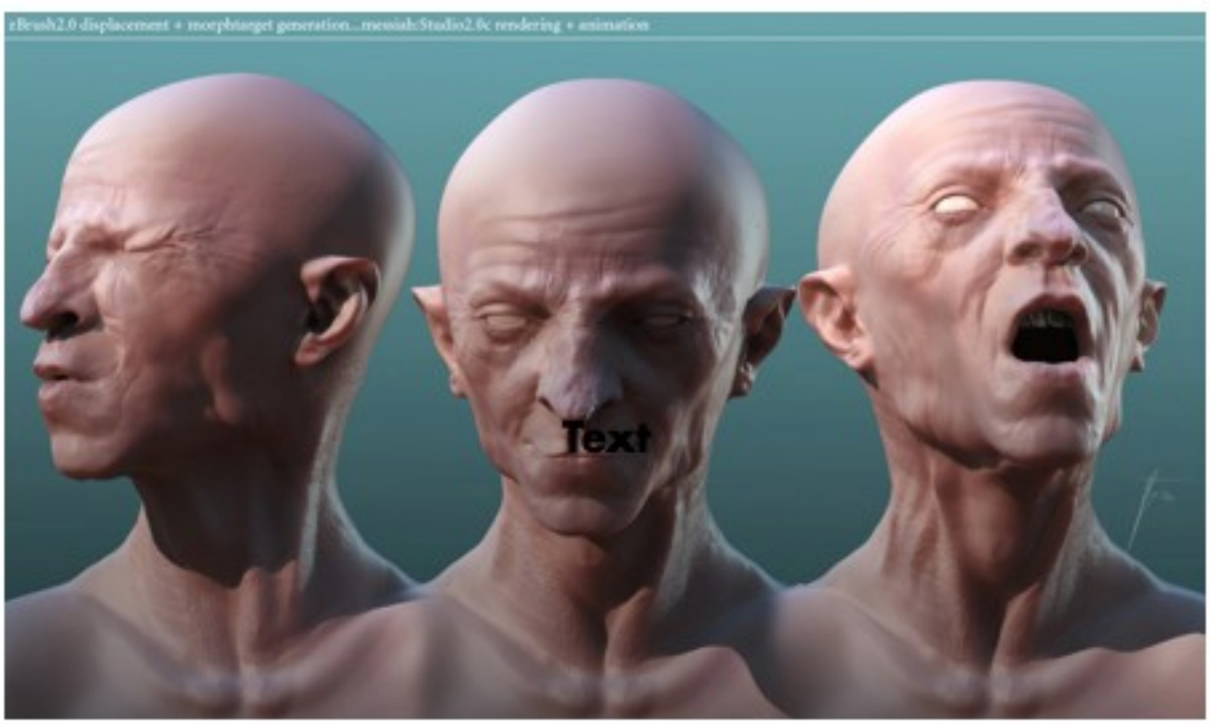
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} \\ m_{yx} & m_{yy} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{x}' = \mathbf{M}\,\mathbf{x}$$

Lines go to Lines -> Linear Transform

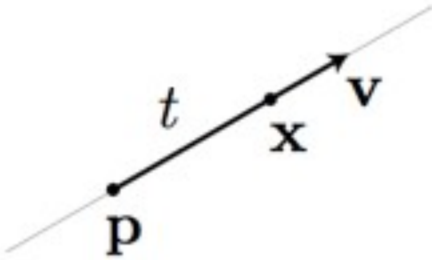


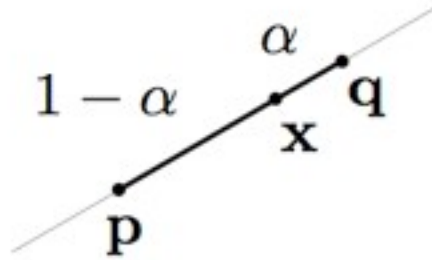




Non-Linear Transforms!

Parametric Forms of a Line





$$\mathbf{x} = \mathbf{p} + t\mathbf{v} \qquad\qquad \mathbf{x} = (1-\alpha)\mathbf{p} + \alpha\mathbf{q}$$

Proof that Lines map to Lines

Start with the parametric form of a line

$$\mathbf{x} = (1-\alpha)\mathbf{p} + \alpha\mathbf{q}$$

Transform all the points on the line

$$\mathbf{x}' = \mathbf{M}\,\mathbf{x} = (1-\alpha)\mathbf{M}\,\mathbf{p} + \alpha\,\mathbf{M}\,\mathbf{q}$$
$$= (1-\alpha)\mathbf{p}' + \alpha\mathbf{q}'$$

Thus, a line transforms into a linear combination of transformed points, which is a line

$$\mathbf{p}' = \mathbf{M}\,\mathbf{p}$$
$$\mathbf{q}' = \mathbf{M}\,\mathbf{q}$$

Advantages of the Matrices

Combine a sequence of transforms into a single transform

$$p' = A (B (C (D (p))))$$
$$= (A B C D) P$$
$$= M P$$

Why? Matrix multiplication is associative

Advantages

- Very inefficient to recompute the matrix entries
- Very inefficient to multiply matrix times point multiple times
- Compute the matrix M once and then apply that matrix to many points

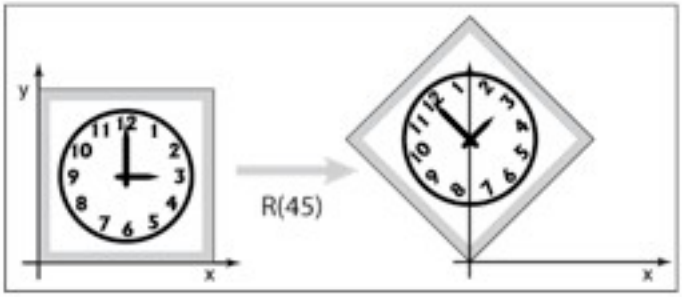
Coordinate Frames

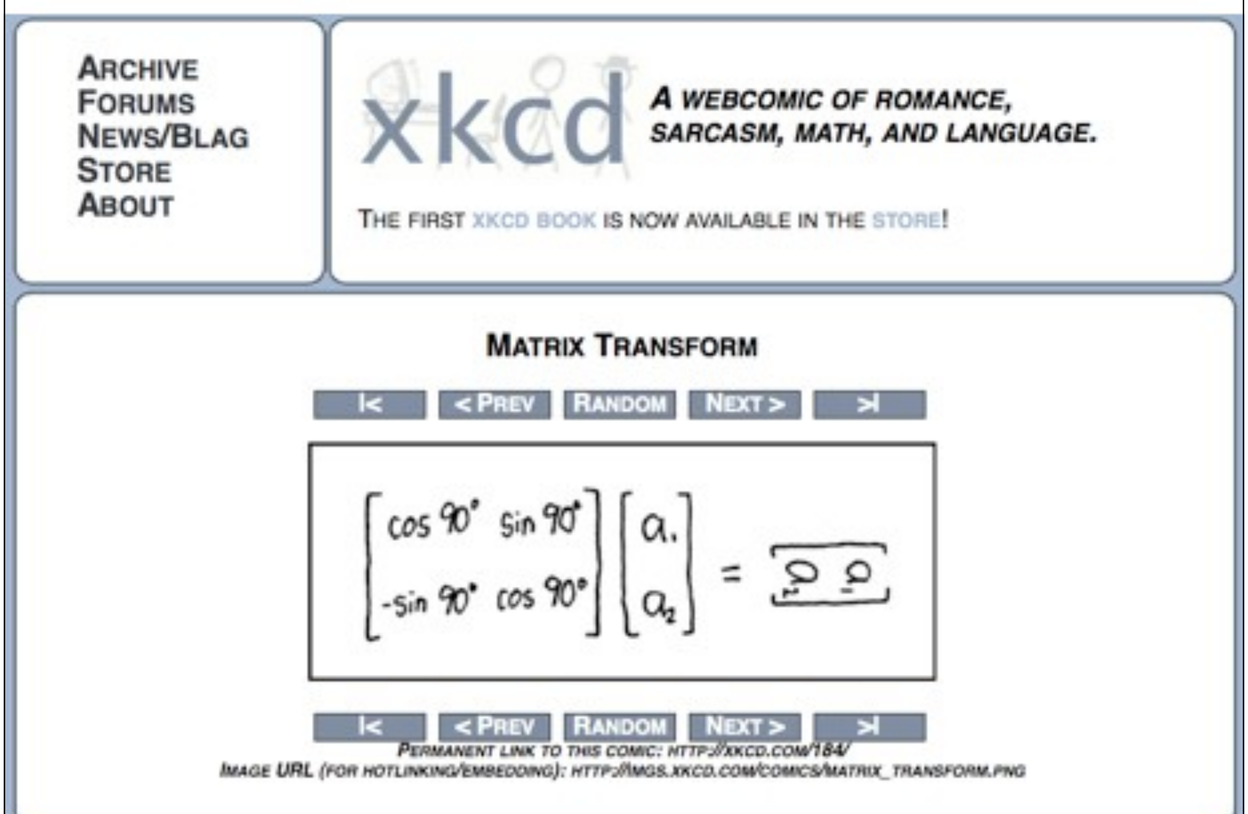
$$\begin{bmatrix} m_{xx} \\ m_{yx} \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} \\ m_{yx} & m_{yy} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} m_{xy} \\ m_{yy} \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} \\ m_{yx} & m_{yy} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

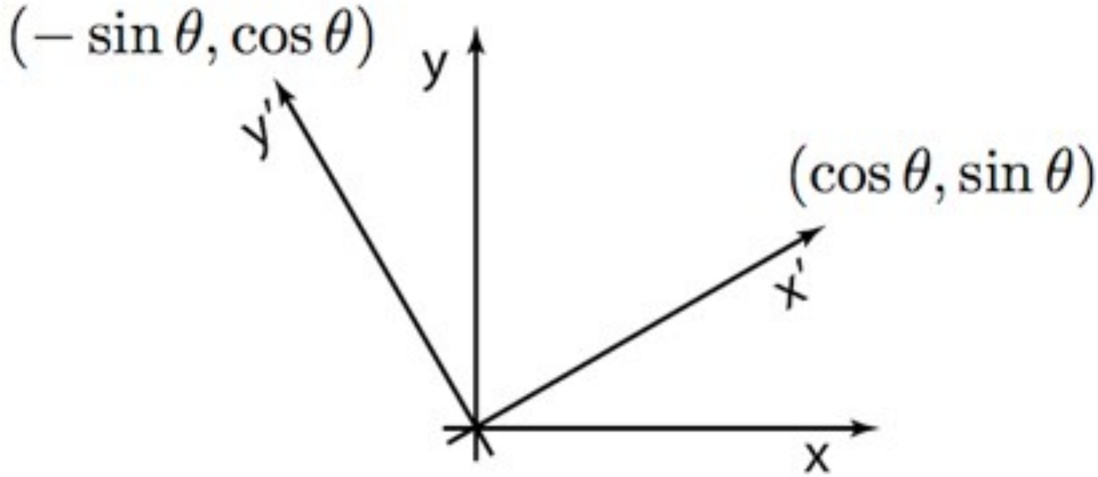
Thus, can interpret columns of the matrix as the positions of the x and y axis that define the frame

Rotation Matrix?



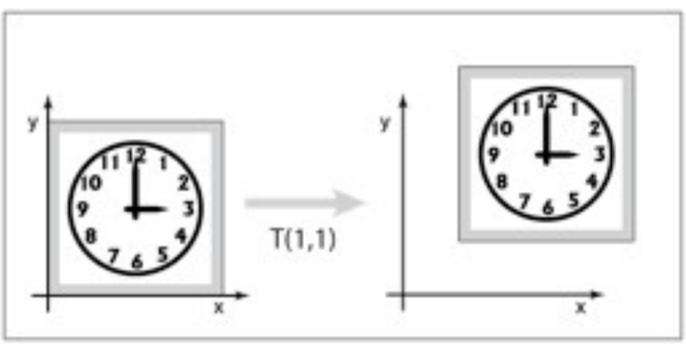


Rotation Matrix



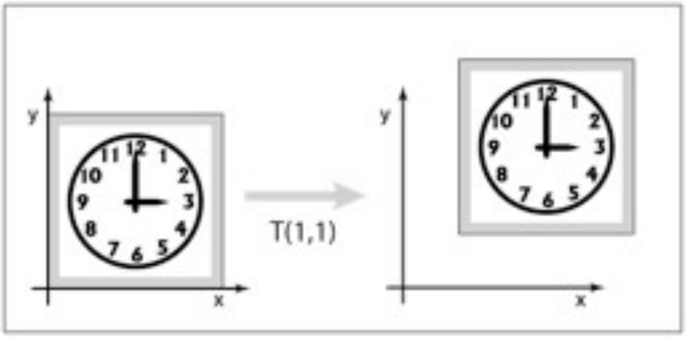
$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Translate??



$$x' = x + t_x$$
$$y' = y + t_y$$

Clever Solution: Add Extra Coordinate



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Points & Vectors Translate Differently

Points (x,y,1) are changed by translates

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Vectors (x,y,0) are NOT changed by translates

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

This is the homogenous coordinate representatiom

OpenGL

Current Transformation Matrix

OpenGL maintains a current transformation matrix (CTM).

All geometry is transformed by the CTM.

Transformation commands are concatenated onto the CTM on the right.

CTM = CTM * T

This causes T to be applied before the old CTM

OpenGL Matrix Functions

`glLoadIdentity( )`

- Set the transform matrix to the identity matrix

`glLoadMatrix( matrix M )`

- Replace the transform matrix with M

`glMultMatrix( matrix M )`

- Multiplies transform matrix by M
- `glRotate`, `glTranslate`, `glScale` etc. are just wrappers for `glMultMatrix`

Graphics Coordinate Frames

Object

- Raw values as provided by glVertex (ex. teacup centered at origin)

World

- Object at final location in environment (ex. teacup recentered to be on top of a table)

Screen

- Object at final screen position

OpenGL Matrix Functions

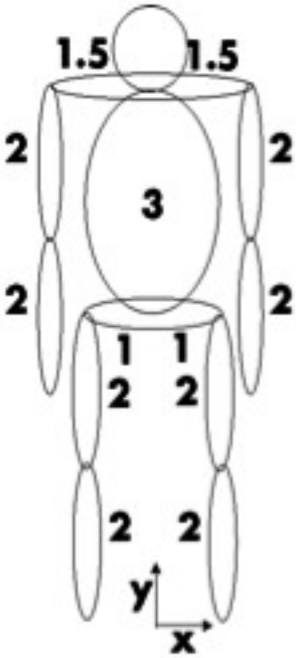
`glMatrixMode( mode )`

- **Sets which transformation matrix to modify**
- `GL_MODELVIEW`: **object to world transform**
- `GL_PROJECTION`: **world to screen transform**
- `CTM = GL_PROJECTION * GL_MODELVIEW`

Why? May need to transform by one of these matrices for certain calculations like lighting. More on this latter.

Hierarchical Transformations

Skeleton




```
body
   torso
      head
      shoulder
         larm
            upperarm
            lowerarm
            hand
         rarm
            upperarm
            lowerarm
            hand
      hips
         lleg
            upperleg
            lowerleg
            foot
         rleg
            upperleg
            lowerleg
            foot
```


Skeletons and Linkages

1. Skeleton represents the hierarchy of an assembly of parts
   - The arm is made of an upper arm and a lower arm
   - The shapes are different parts are "geometric instances"

2. Parts connected by joints
   - "The ankle joint is connected to the knee joint"

3. Different types of joints move differently

How Linkages Work

Lower levels of the hierarchy move when the upper levels moves

- e.g. moving the left shoulder moves the left arm and the left hand

Motion in one sub-tree does not effect the position of a part in another sub-tree

- e.g. moving the left hand does not effect the right hand

OpenGL Matrix Stack

Save and restore transformations

`glPushMatrix ( )`

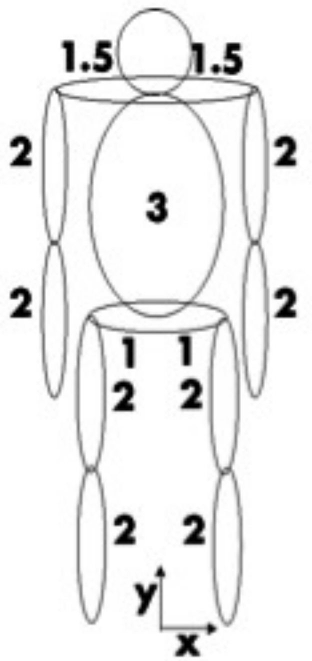
- Pushes the current matrix onto the top of the matrix stack

`glPopMatrix ( )`

- Pops the matrix off the top of the matrix stack and loads it as the current matrix

This makes it possible to model hierarchical assemblies of parts (robots, avatars, etc.)

Skeleton

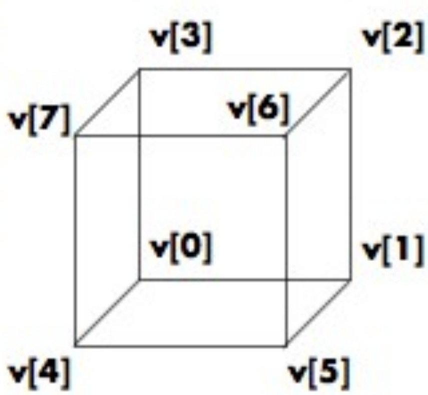



```
translate(0,4,0);
torso();
pushmatrix();
     translate(0,3,0);
     shoulder();
     pushmatrix();
       rotatey(necky);
       rotatex(neckx);
       head();
     popmatrix();
     pushmatrix();
       translate(1.5,0,0);
       rotatex(lshoulderx);
       upperarm();
       pushmatrix();
         translate(0,-2,0);
         rotatex(lelbowx);
         lowerarm();
         ...
       popmatrix();
     popmatrix();
...
```


Points/Polygons (also .obj files)

```
typedef float Point[3];

Point verts[8] = {
 {-1.,-1.,-1.},
 { 1.,-1.,-1.},
 { 1., 1.,-1.},
 {-1., 1.,-1.},
 {-1.,-1., 1.},
 { 1.,-1., 1.},
 { 1., 1., 1.},
 {-1., 1., 1.},
};
```

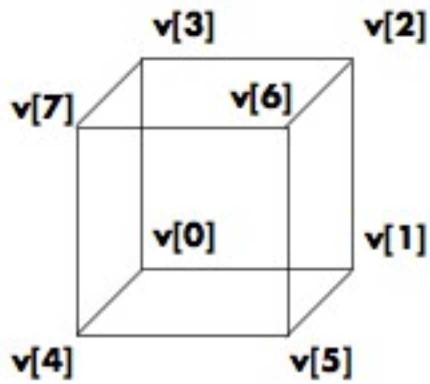



```
int polys[6][4] = {
 {0,3,2,1},
 {2,3,7,6},
 {0,4,7,3},
 {1,2,6,5},
 {4,5,6,7},
 {0,1,5,4}
}
```

Points/Polygons (also .obj files)

```
face(int poly[4]) {
   glBegin(GL_POLYGON);
     glVertex3fv(poly[0]);
     glVertex3fv(poly[1]);
     glVertex3fv(poly[2]);
     glVertex3fv(poly[3]);
   glEnd();
}

cube() {
 for( int i = 0; i < n; i++ )
   face(polys[i]);
}
```



More efficient to use glDrawElements()

Demo of Shape.c

Things to Remember

How to use transforms?

- Different types: translate, rotate, scale, ...
- Order of transforms matters
- Coordinate frames
- Hierarchical modeling using push/pop

How transforms work?

- Matrix representation of transforms
- Matrix concatenation