CS148: Introduction to Computer Graphics and Imaging

Transforms

Today's Outline

Purpose of transformations

Types of transformations: rotations, translates, …

Composing multiple transformations

Representing transformations as matrices

Hierarchical transformations

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**

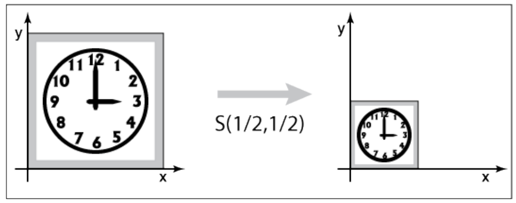
Modeling

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

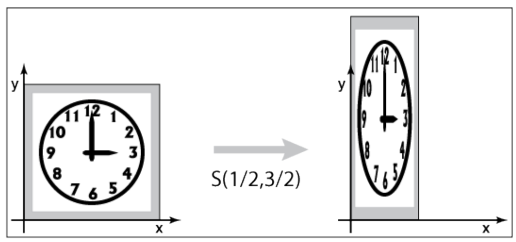
Gallery of Transformations

Scale

Uniform

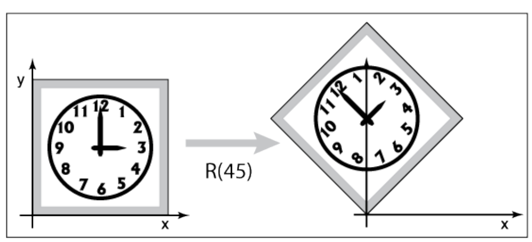


Nonuniform



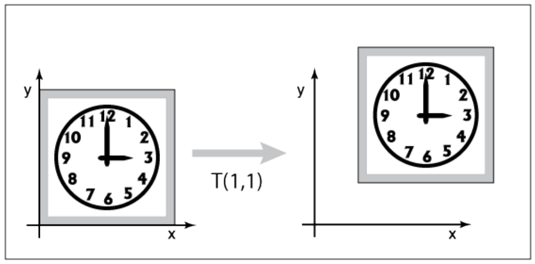
glScalef(sx,sy,sz)

Rotate



glRotatef(angle,ax,ay,az)

Translate

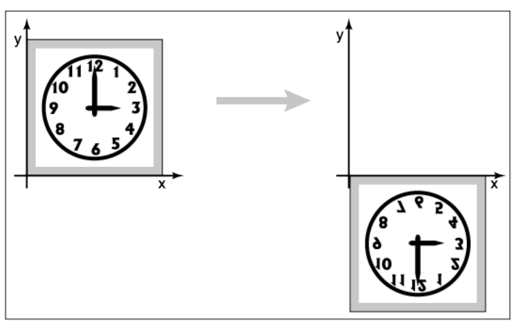


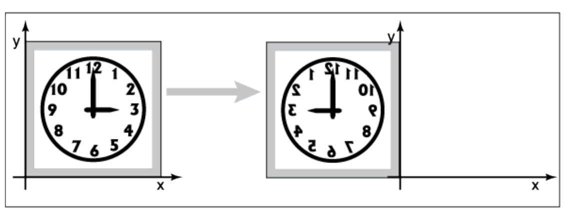
glTranslatef(tx,ty,tz)

CS148 Lecture 4

Pat Hanrahan, Fall 2009

Reflect

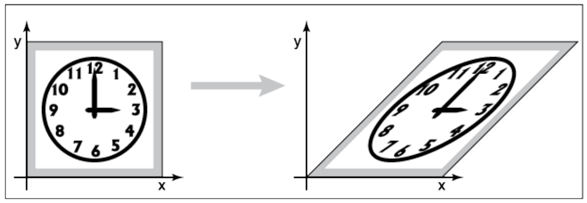


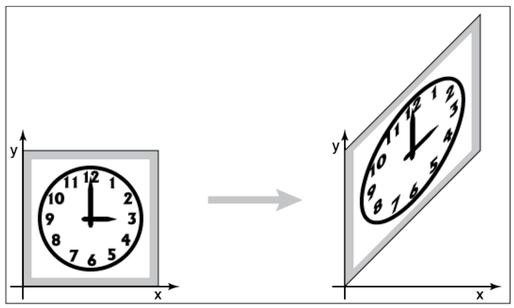


CS148 Lecture 4

Pat Hanrahan, Fall 2009

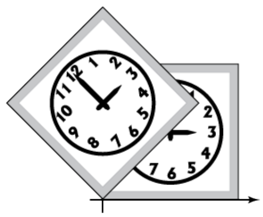
Shear



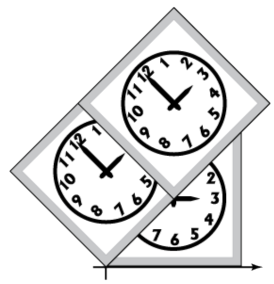


Composing Transformations

Rotate, Then Translate

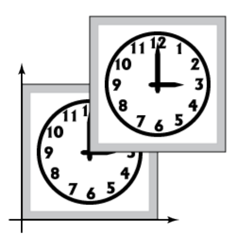


R(45)

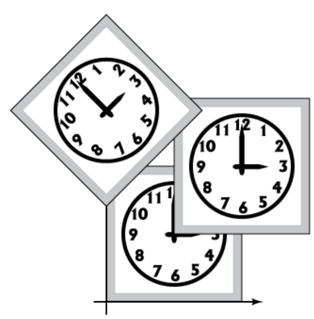


T(1,1) R(45)

Translate, Then Rotate

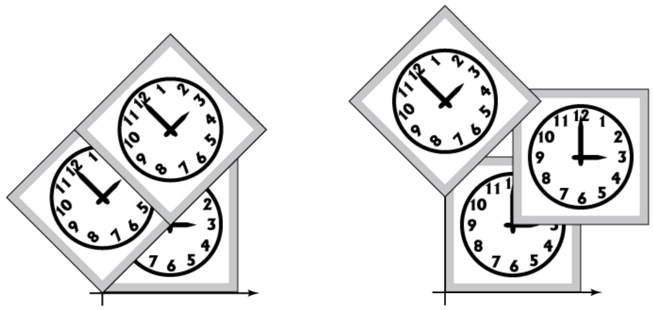


T(1,1)



R(45) T(1,1)

Order Matters



T(1,1) R(45) ≠ R(45) T(1,1)

CS148 Lecture 4 Pat Hanrahan, Fall 2009

OpenGL Order of Transformations

Math

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

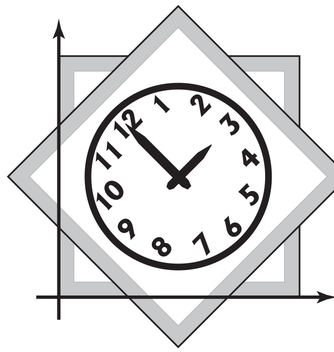
OpenGL (last one specified is the first one applied)

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

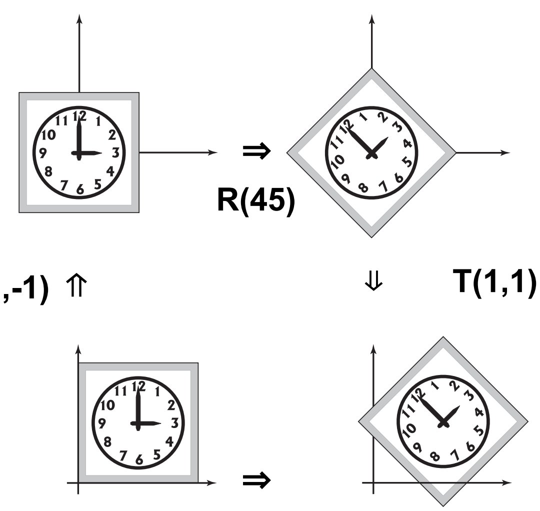
That is, the translate is applied before the rotate

CS148 Lecture 4 Pat Hanrahan, Fall 2009

Rotate 45 @ (1,1) ?



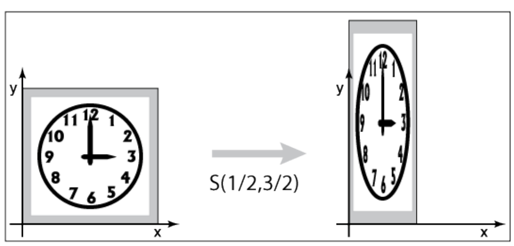
Rotate 45 @ (1,1)



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

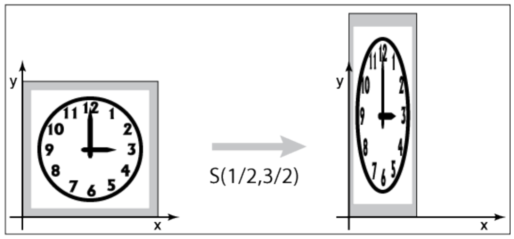
Math of Linear Transformations (Matrices)

Scale



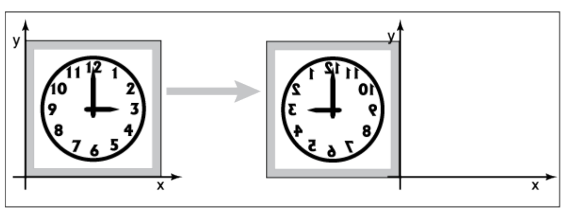
$$x' = s_x\, x$$
$$y' = s_y\, y$$

Scale



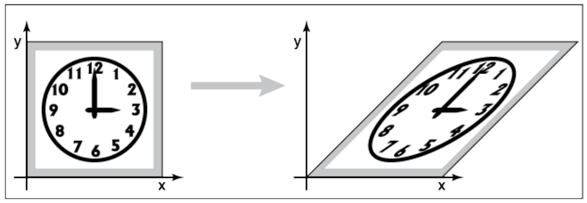
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Reflection Matrix?



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

Shear Matrix?



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

Linear Transformations = Matrices

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

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

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

Advantages of the Matrix Formulation

1. Combine a sequence of transforms into a single transform

   p' = A (B (C (D (p))))

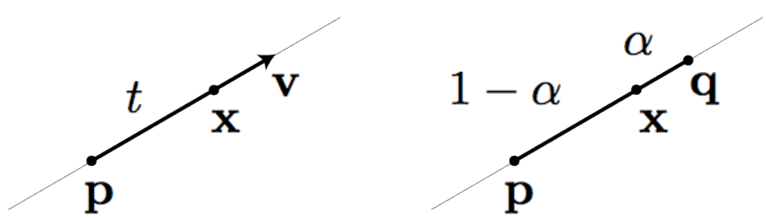
   = (A B C D) P

   = M P

2. Compute the matrix M once; apply to many points

   Very inefficient to keep recomputing the matrices

Parametric Forms of a Line



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

Why Called Linear Transformations?

Because lines are transformed into lines

Start with a line

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

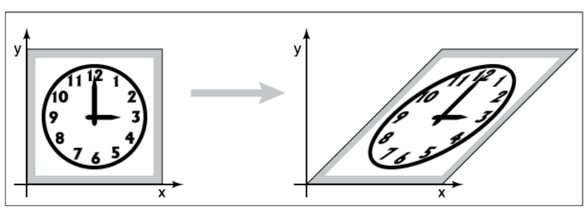
Transform it

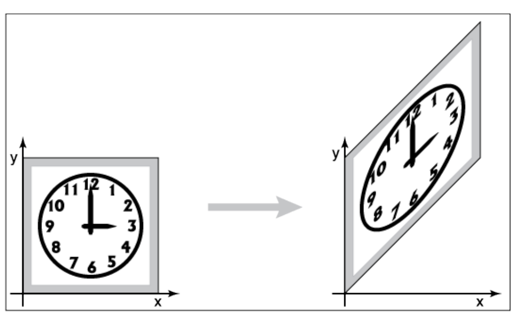
$$\mathbf{x}' = \mathbf{M}\,\mathbf{x} = (1-\alpha)\mathbf{M}\,\mathbf{p} + \alpha\,\mathbf{M}\,\mathbf{q}$$
$$= (1-\alpha)\mathbf{p}' + \alpha\mathbf{q}'$$

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

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

Lines go to Lines -> Linear Transform





Non-Linear Transforms!

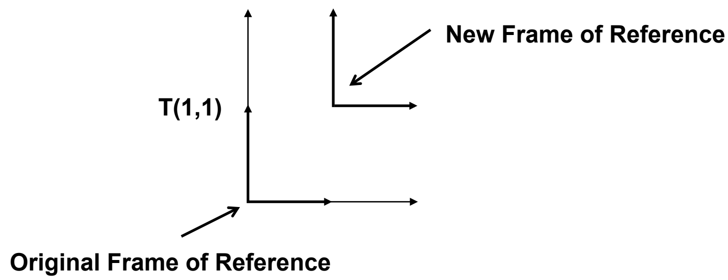
Coordinate Frames

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

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

Thus, can interpret columns of the matrix as the positions of the new x and y axis

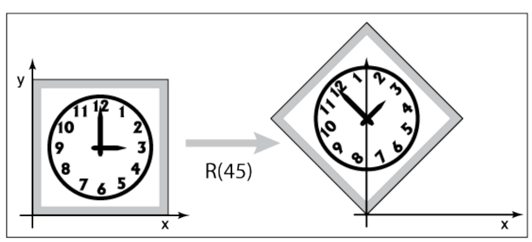
Coordinate Frame

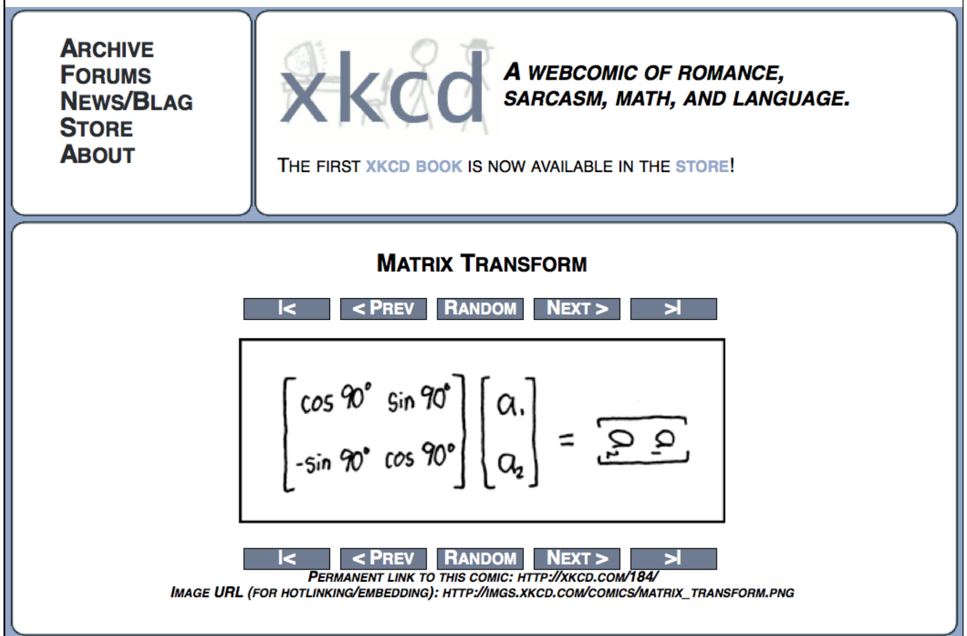


Transforms create new frames of reference

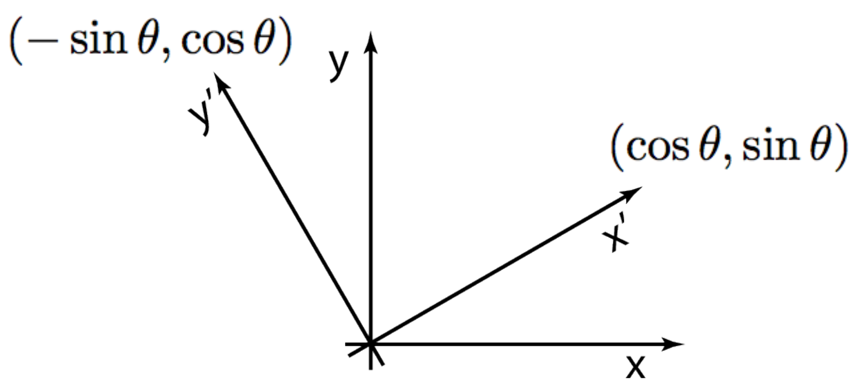
These “frames” define “coordinate systems”

Rotate



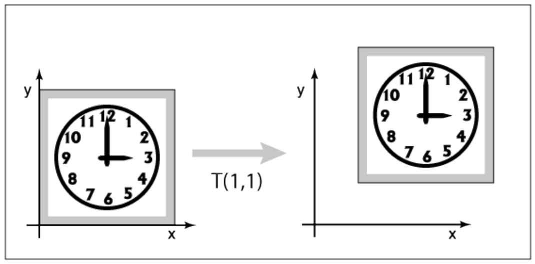


Rotation Matrix



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

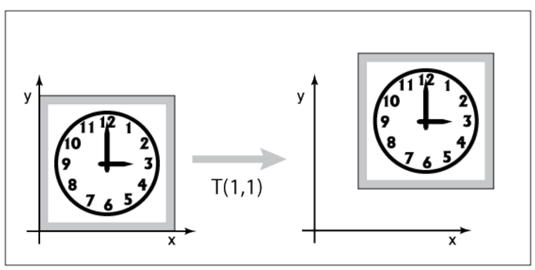
Translate



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

CS148 Lecture 4 Pat Hanrahan, Fall 2009

Translate



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

CS148 Lecture 4 Pat Hanrahan, Fall 2009

Points and Vectors Translate Differently

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

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

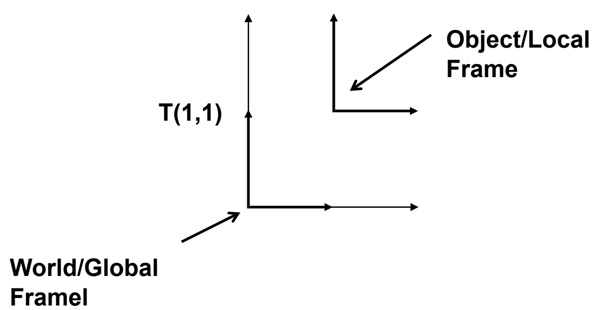
Vectors (x,y,0) are NOT shifted by translates

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

This is the *homogenous coordinate* representation p/v

OpenGL

Global vs. Object Frames / Coordinates



Transforms create new frames of reference
These “frames” define “coordinate systems”

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

OpenGL Matrix Functions

glLoadIdentity( )

- Reset the selected transform matrix to the identity matrix

glLoadMatrix(matrix M)

- Replace the selected transform matrix with M

glMultMatrix(matrix M)

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

OpenGL Matrix Functions

There is a matrix stack for each matrix mode

The stack makes it possible to save/restore the matrix

glPushMatrix ( )

- Adds the current matrix to 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 complex hierarchical assemblies of parts (robots, avatars, etc.)

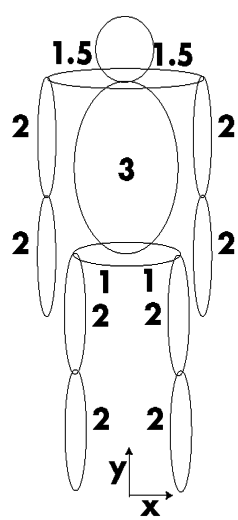
Current Transformation Matrix

- OpenGL maintains a *current transformation matrix (CTM).* All geometry is transformed by the CTM.
- The CTM defines the current or *object* or *local* coordinate system. All geometry is defined in the current coordinate system.
- Transformation commands are concatenated onto the ctm. Note: The last transformation specified is the first to be performed.

  CTM' = CTM * T
- The CTM may be pushed and popped from a stack, onto a transformation stack.

Hierarchical Models

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. Hierarchy represents the connected nature of the assembly of parts

   "The ankle joint is connected to the knee joint"

2. Different connections move differently

   e.g. a ball and socket joint, linear actuator

Implications

1. Lower levels of the hierarchy move when the upper levels moves

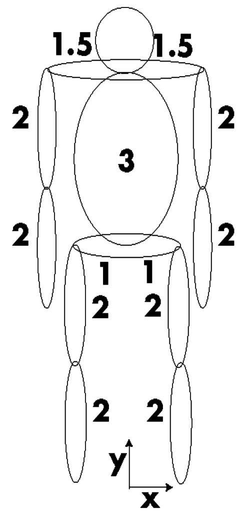
   e.g. moving the left shoulder moves the left hand

2. 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

3. Leads to a hierarchical set of transformations.
   - Some transformations are fixed
   - Some change when the object moves
4. May "instance" the shape in different positions
   - Saves space and code

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();
...
```


Things to Remember

How to use transforms?

- Different types: translate, rotates, scales
- Order matters
- Current transformation matrix
- Coordinate frames
- Hierarchical modeling using push/pop

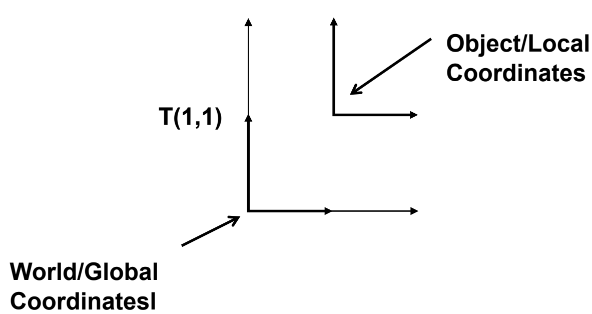
How transforms work?

- Matrix representation of transforms
- Matrix concatenation

Coordinate Systems

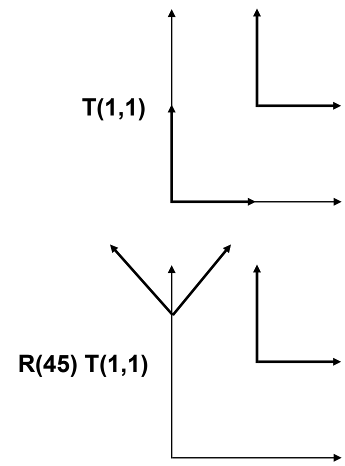
Transforms Create New Coord. Systems

Transforms create new coordinate systems



Transforms Create New Coord. Systems

Transforms create new coordinate systems

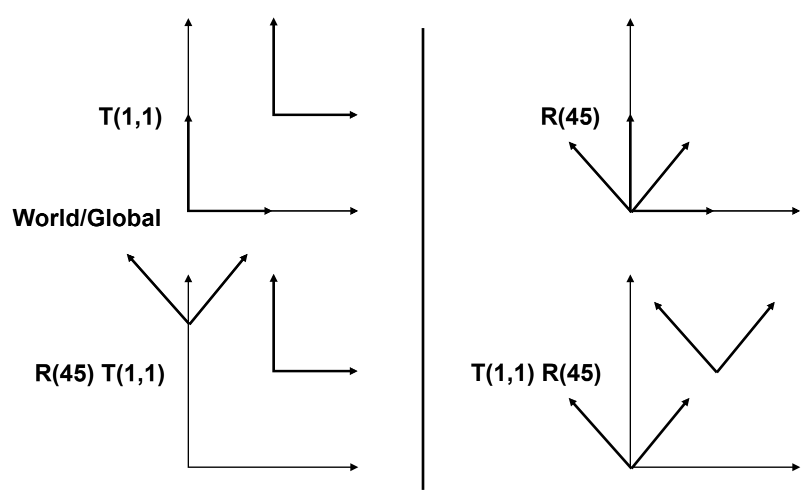


CS148 Lecture 4

Pat Hanrahan, Fall 2009

Specify in World/Global Coordinates

Transform the object => Apply from right to left

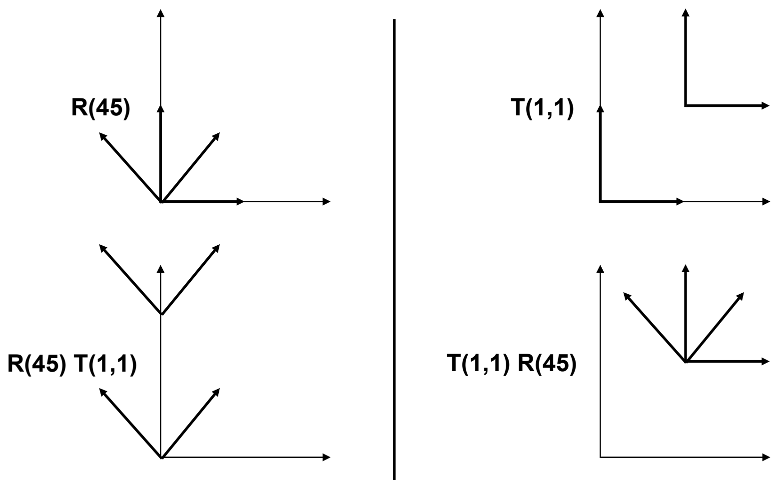


CS148 Lecture 4

Pat Hanrahan, Fall 2009

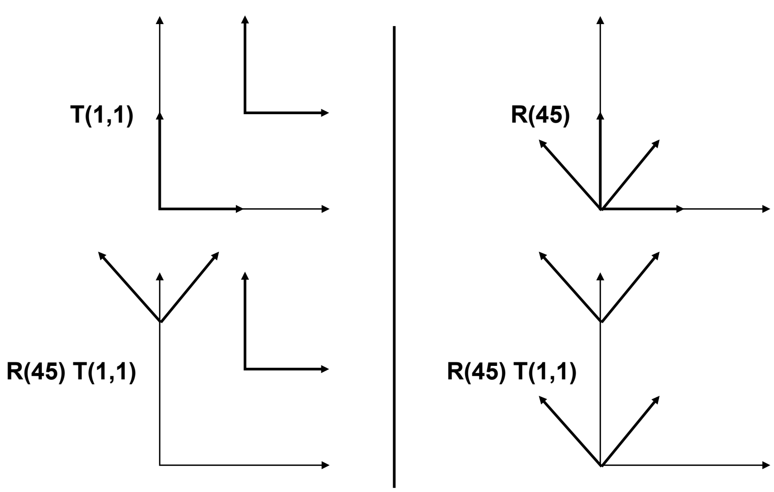
Transform in Object/Local Coordinates

Transform the coordinate system => Apply from left to right



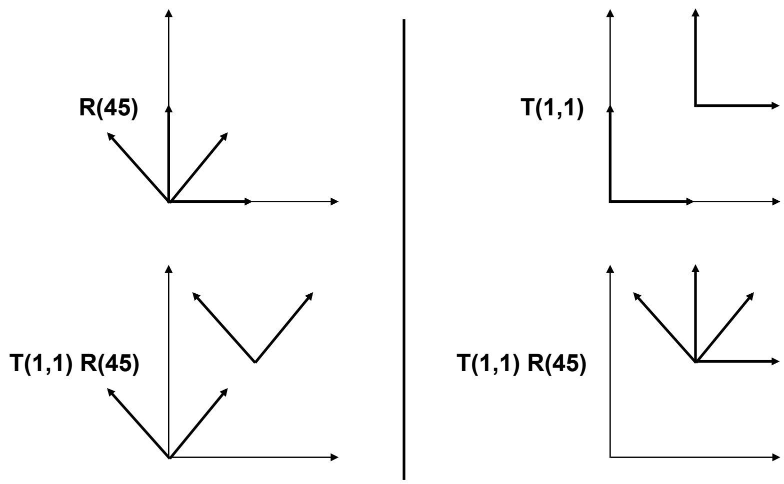
Two Interpretations are Equivalent

Global/World **Local/Object**



Two Interpretations are Equivalent

Global/World | **Local/Object**



Pat Hanrahan, Fall 2009