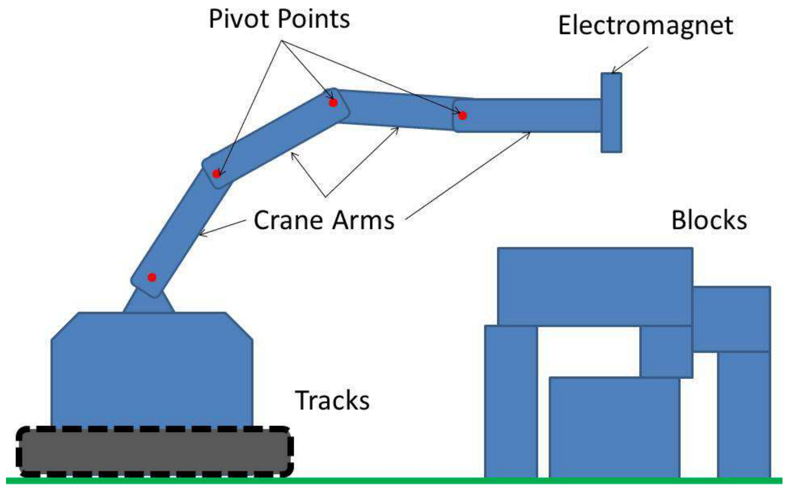
2D Graphics

Direct Manipulation

- GUIs often allow direct manipulation of on-screen artifacts with the mouse
- Need to perform many inside tests to implement DM
  - Easy for rectangles
  - Not so easy for other shapes
  - Need a general strategy

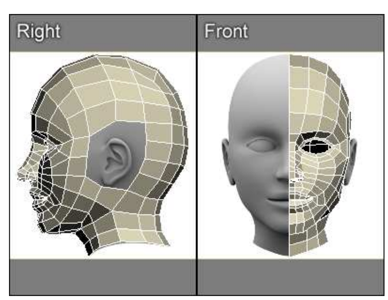
```
for (Item item : displayList)
{ if (item.contains(mouse.x, mouse.y)) {
  {  ...
  }
}
```

Assignment 2



Introduction

- What is Computer Graphics?
  - Creation, storage, and manipulation of images and their models
- Images: what we see on the display
- Model: a representation (often mathematical) of the image
  - 2D array of color values
  - Lines making up a stick figure
  - Points on the surface of an object, arranged in a mesh
  - A graph representing wires in electrical circuit



4

http://www.secondpicture.com/tutorials/3d/3d_model_of_a_human_head.png

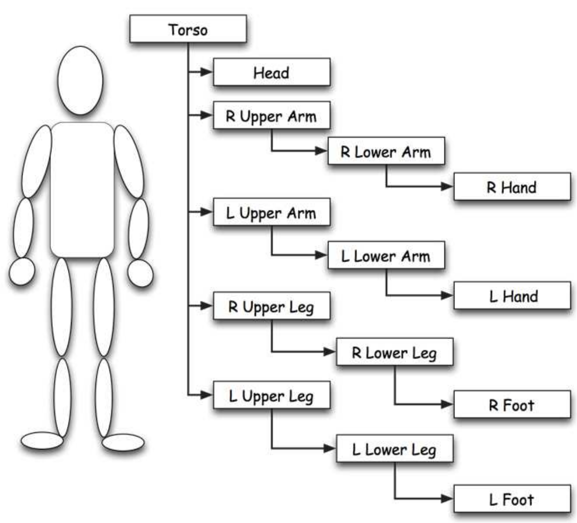
Modeling versus Rendering

- Modeling: Representing the important properties of an object (location, size, orientation, color, texture, etc) in data structures
- Rendering: Using the properties of the model to create an image to display on the screen
  - For pixel-based graphics (photos, Photoshop or GIMP output) the rendering is trivial
  - Other models may involve very complex steps to render the image (Illustrator, rendering movie scenes for *Toy Story* or *Transformers* or ...)
- CS349: modeling and rendering in 2D; CS488: 3D

5

Modeling with a Scene Graph

- See A02 sample code
- Each part draws its children
- Each part specifies its location, size, and orientation



6

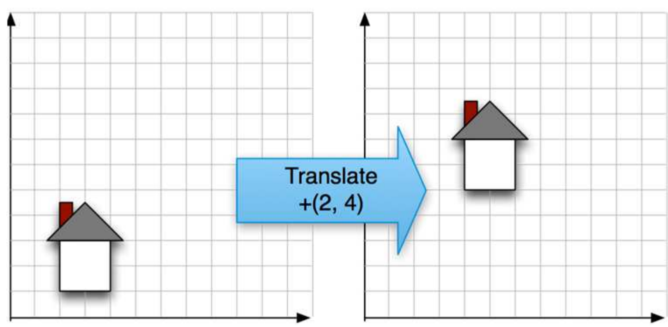
Modeling with a Scene Graph

- To specify the location, size, and orientation of each part, we need several transformations on geometric objects:
  - – translation (location)
  - – scaling (size)
  - – rotation (orientation)

7

Translation

- Translating a coordinate means adding a vector to each of its components

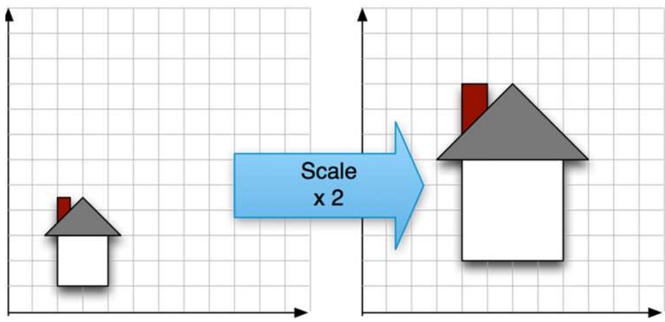


$$x' = x + t_x$$
$$y' = y + t_y$$

8

Scaling

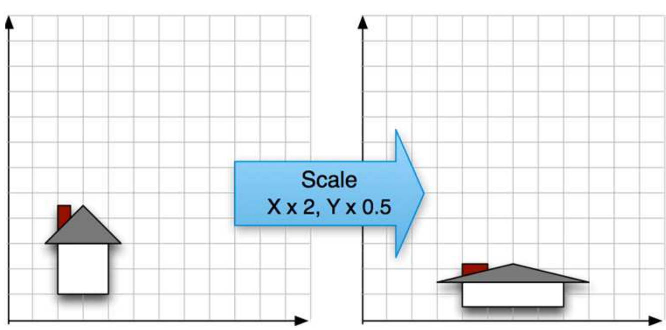
- Scaling a coordinate means multiplying each of its components by a scalar
- Uniform scaling means this scalar is the same for all components:



9

Scaling

- Non-uniform scaling: different scalars per component:

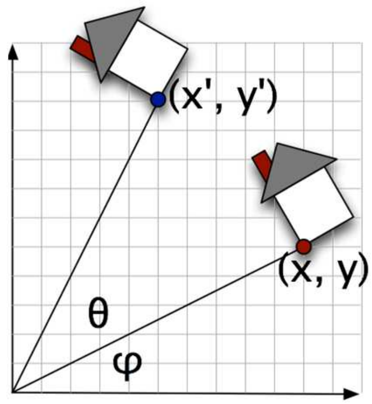


$$x' = x \times s_x$$

$$y' = y \times s_y$$

10

2-D Rotation



$$x = r\cos(\varphi)$$
$$y = r\sin(\varphi)$$
$$x' = r\cos(\varphi + \theta)$$
$$y' = r\sin(\varphi + \theta)$$

Trig Identities...

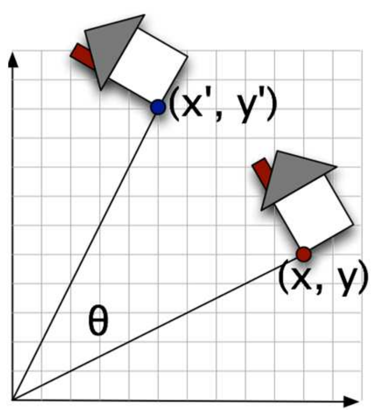
$$x' = r\cos(\varphi)\cos(\theta) - r\sin(\varphi)\sin(\theta)$$
$$y' = r\sin(\varphi)\sin(\theta) + r\cos(\varphi)\cos(\theta)$$

Substitute...

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

11

2-D Rotation



$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

12

Combining 2D Transformations

- Rotate:

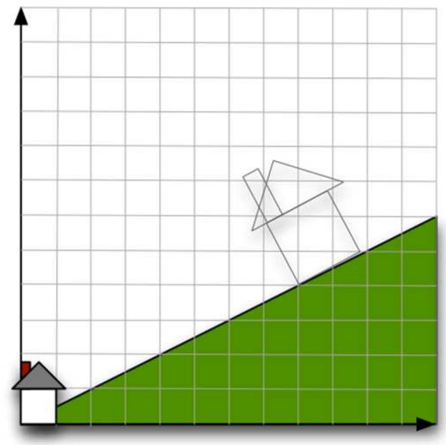
$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

- Translate:

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

- Scale:

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



13

Combining 2D Transformations

- Rotate:

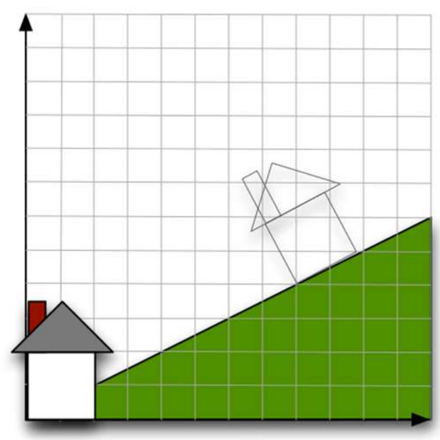
$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

- Translate:

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

- **Scale**:

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

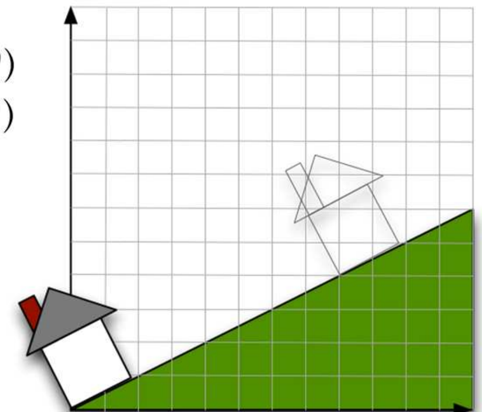


$$x_1 = 2x$$
$$y_1 = 2y$$

14

Combining 2D Transformations

- **Rotate**:
  $x' = x\cos(\theta) - y\sin(\theta)$
  $y' = x\sin(\theta) + y\cos(\theta)$
- Translate:
  $x' = x + t_x$
  $y' = y + t_y$
- Scale:
  $x' = x \times s_x$
  $y' = y \times s_y$



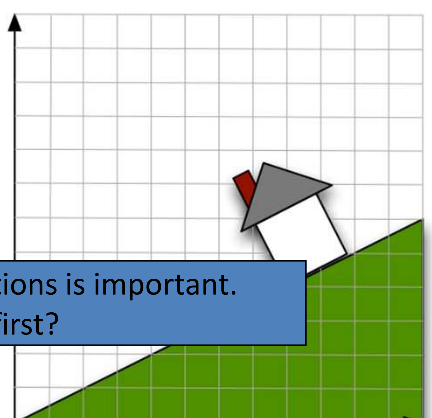
$$x_2 = 2x\cos(30) - 2y\sin(30)$$
$$y_2 = 2x\sin(30) - 2y\sin(30)$$

15

Combining 2D Transformations

- Rotate:
  $x' = x\cos(\theta) - y\sin(\theta)$
  $y' = x\sin(\theta) + y\cos(\theta)$
- **Translate**:
  $x' =$
  $y' =$
- Scale:
  $x' = x \times s_x$
  $y' = y \times s_y$

Note: Order of operations is important.
What if you translate first?



$$x_3 = 2x\cos(30) - 2y\sin(30) + 8$$
$$y_3 = 2x\sin(30) - 2y\sin(30) + 4$$

16

Matrix Representation

- Goal: Represent each 2D transformation with a matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- Multiply matrix by column vector
  ⇔ apply transformation to point

$$\left.\begin{matrix} x' = ax + by \\ y' = cx + dy \end{matrix}\right\} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

17

Matrix Representation

- Why? Transformations can be combined by multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- We can multiply transformation matrices together

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} aei + bgi + afk + bhk & aej + bgj + ael + bgl \\ cei + dgi + cfk + dhk & cej + dgj + cfl + dhl \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

  – This single matrix can then be used to transform many points
  – Can be downloaded to a GPU to speed the process

18

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Scale around (0,0)?

$$\left.\begin{aligned} x' &= x \times s_x \\ y' &= y \times s_y \end{aligned}\right\} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

19

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Rotate around (0,0)?

$$\left.\begin{aligned} x' &= x\cos(\theta) - y\sin(\theta) \\ y' &= x\sin(\theta) + y\cos(\theta) \end{aligned}\right\} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

20

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Mirror about Y axis?

$$\left.\begin{array}{c} x' = -x \\ y' = y \end{array}\right\} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

21

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Translation?

$$\left.\begin{array}{c} x' = x + t_x \\ y' = y + t_y \end{array}\right\} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

No! Only linear 2D transformations
can be represented with a 2x2 matrix

22

Homogeneous Coordinates

- Homogeneous coordinates
  - represent coordinates in 2 dimensions with a 3-vector

$$\begin{bmatrix} x \\ y \end{bmatrix} \Leftrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Homogeneous coordinates simplify 2D transformations

23

Homogeneous Coordinates

- Q: Can we represent translation as a 3x3 matrix?

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} \Leftrightarrow \begin{matrix} Ax + By + C = x + t_x \\ Dx + Ey + F = y + t_y \\ Gx + Hy + I = 1 \end{matrix}$$

24

Homogeneous Coordinates

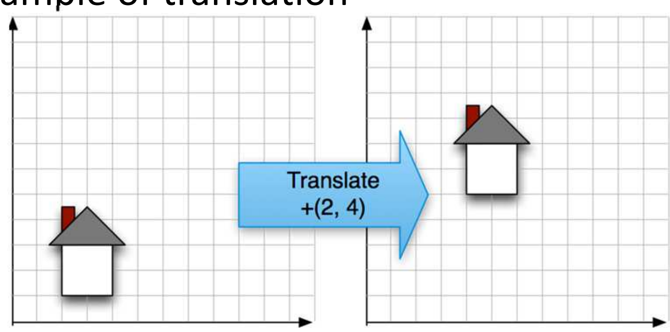
- Q: Can we represent translation as a 3x3 matrix?

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

25

Translation

- Example of translation

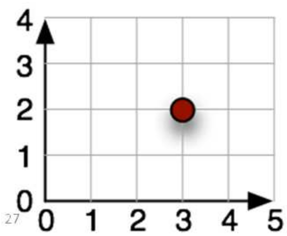


$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + 2 \\ y + 4 \\ 1 \end{bmatrix}$$

26

Homogeneous Coordinates

- Add a 3rd coordinate to every 2D point
  - (x, y, w) represents a point at location (x/w, y/w)
  - assume w > 0
- Convenient coordinate system to represent many useful transformations



(3, 2, 1) or
(6, 4, 2) or
(7.5, 5, 2.5) or
...

27

Vectors?

- Points: represent a position
- Vectors: represent direction and magnitude
- Operations:
  - v + v = v
  - v x s = v
  - p - p = v
  - p + v = p

28

Representing Vectors

$$\vec{v}+\vec{w}=\begin{bmatrix}v_x\\v_y\\0\end{bmatrix}+\begin{bmatrix}w_x\\w_y\\0\end{bmatrix}=\begin{bmatrix}v_x+w_x\\v_y+w_x\\0\end{bmatrix}$$

Add vectors

$$\vec{v}\times s=\begin{bmatrix}v_x\\v_y\\0\end{bmatrix}\times s=\begin{bmatrix}v_x\times s\\v_y\times s\\0\end{bmatrix}$$

Scalar Multiply

$$p-q=\begin{bmatrix}p_x\\p_y\\1\end{bmatrix}-\begin{bmatrix}q_x\\q_y\\1\end{bmatrix}=\begin{bmatrix}p_x-q_x\\p_y-q_x\\0\end{bmatrix}$$

Subtract points

$$p+\vec{v}=\begin{bmatrix}p_x\\p_y\\1\end{bmatrix}+\begin{bmatrix}v_x\\v_y\\0\end{bmatrix}=\begin{bmatrix}p_x+v_x\\p_y+v_x\\1\end{bmatrix}$$

Point + Vector

29

Translating Vectors

- A vector has no position, so translating it shouldn't change anything.

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

30

Rotation Matrix

- Vectors:

$$\begin{bmatrix} x' \\ y' \\ 0 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & a \\ \sin(\theta) & \cos(\theta) & b \\ c & d & e \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x\cos(\theta) - y\sin(\theta) \\ x\sin(\theta) + y\cos(\theta) \\ 0 \end{bmatrix}$$

- Points

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & a \\ \sin(\theta) & \cos(\theta) & b \\ c & d & e \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos(\theta) - y\sin(\theta) \\ x\sin(\theta) + y\cos(\theta) \\ 1 \end{bmatrix}$$

31

Scaling Matrix

- Vectors:

$$\begin{bmatrix} x' \\ y' \\ 0 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x \cdot s_x \\ y \cdot s_y \\ 0 \end{bmatrix}$$

- Points $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \cdot s_x \\ y \cdot s_y \\ 1 \end{bmatrix}$

32

Matrix Composition

- Transformations can be combined by matrix multiplication

$$p' = T(t_x, t_y) \cdot R(\theta) \cdot S(s_x, s_y) \cdot p$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

33

Matrix Composition

- Review: Properties of Matrix Multiplication
  - Associative: A(BC) = (AB)C
  - Not Commutative: AB ≠BA
    - Order of transformations matters!

$$p' = T \cdot R \cdot S \cdot p$$

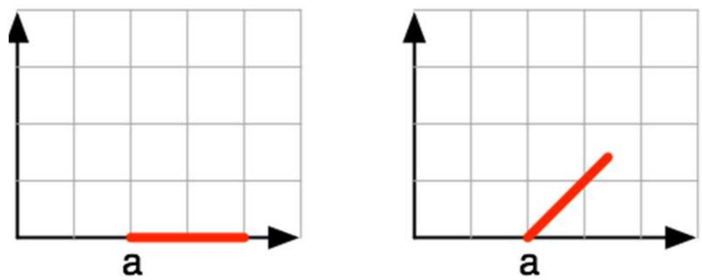
“Global” $$p' = (T \cdot (R \cdot (S \cdot p)))$$ “Local”

$$p' = (T \cdot R \cdot S) \cdot p$$

34

Matrix Composition

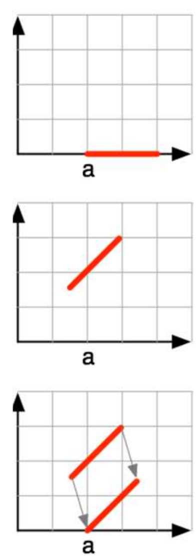
- What if we want to rotate and translate?
  - Ex: Rotate line segment by 45 degrees about endpoint **a**



35

Multiplication Order – Wrong Way

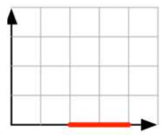
- Beginning situation
- Rotate 45 degrees, R(45)
  - Affects both endpoints
  - Oops
- Could try translating both endpoints to return **a** to its original position
  - But by how much?



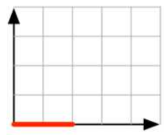
36

Multiplication Order

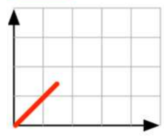
- Scaling and rotation are both about the origin
- Process:
  - Translate shape to the origin
  - rotate
  - translate back to where you want it



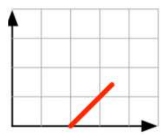
$$p' = I \cdot p$$



$$p' = T_{(-2,0)} \cdot p$$



$$p' = R_{(45)} T_{(-2,0)} \cdot p$$



$$p' = T_{(2,0)} R_{(45)} T_{(-2,0)} \cdot p$$


```
package affineRotateLine;
import javax.swing.*;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.BasicStroke;
public class RotateLine extends JComponent {
	public static void main(String[] args) {
		RotateLine canvas = new RotateLine();
		JFrame f = new JFrame("Rotate Line");
		f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		f.setSize(400, 400);
		f.setContentPane(canvas);
		f.setVisible(true);
	}
```

```
public void paintComponent(Graphics g) {
      Graphics2D g2 = (Graphics2D) g;
      g2.translate(20, 240);
      g2.setStroke(new BasicStroke(3));
      g2.drawLine(0, 0, 0, -200); // vertical axis
      g2.drawLine(0, 0, 200, 0);  // horizontal axis
      g2.setStroke(new BasicStroke(5));   // line
      g2.setColor(Color.RED);
     g2.drawLine(40, 0, 120, 0);
     g2.drawOval(40-4, -4, 8, 8);
      g2.drawOval(120-4, -4, 8, 8);
       // Copy last 4 lines.  Change color to GREEN.
      // What transformations to include to have it rotate
      // 45 degrees about the left-most endpoint?
}}
```

Java2D Intro

- Check out the Graphics class and Graphics2D, a subclass
  - paint methods specify a Graphics object (to be backward compatible)
  - The object passed is actually a Graphics2D object; cast it
- Graphics2D contains an affine transform that is applied to shapes before they are drawn

Useful Graphics2D methods

- AffineTransform getTransform(),
  void setTransform(AffineTransform Tx)
  - Returns/sets a copy of the current Transform in the Graphics2D context.
- void rotate(double theta),
  void rotate(double theta, double x, double y)
  - Concatenates the current Graphics2D Transform with a rotation transform.
  - Second variant translates origin to (x,y), rotates, and translates origin (-x, -y).
- void scale(double sx, double sy)
  - Concatenates the current Graphics2D Transform with a scaling transformation. Subsequent rendering is resized according to the specified scaling factors relative to the previous scaling.
- void translate(double tx, double ty)
  - Concatenates the current Graphics2D Transform with a translation transform.

41

Java2D AffineTransform Class

- AffineTransform handles all matrix manipulations
  - A bit more control than Graphics2D
- Static Methods
  - static AffineTransform getRotateInstance(double theta)
  - static AffineTransform getRotateInstance(double theta, double anchorx, double anchory)
  - static AffineTransform getScaleInstance(double sx, double sy)
  - static AffineTransform getTranslateInstance(double tx, double ty)

42

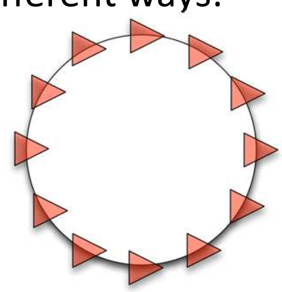
Java2D AffineTransform Class

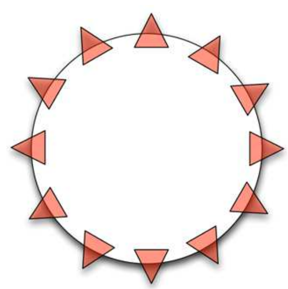
- Concatenation methods
  - void rotate(double theta),
    void rotate(double theta, double anchorx, double anchory)
  - void scale(double sx, double sy)
  - void translate(double tx, double ty)
  - void concatenate(AffineTransform Tx)
- Other Methods
  - AffineTransform createInverse()
  - void transform(Point2D[] ptSrc, int srcOff, Point2D[] ptDst, int dstOff, int numPts)

43

Class Exercise

- Develop the transformations to animate a triangle (drawn at the origin) in a circle in two different ways:

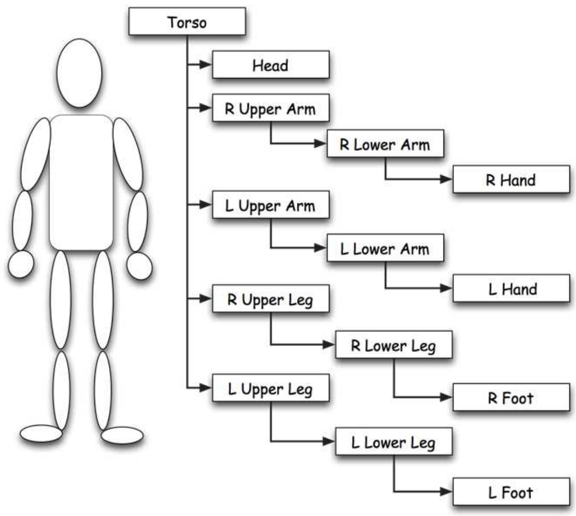




44

Scene Graphs

- Each part has a transform matrix
- Each part draws its children relative to itself



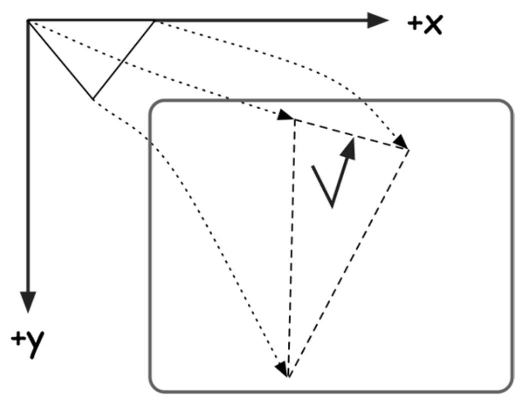
Benefits of Geometrical Manipulations

- Allow reuse of objects in scenes
  - Can create multiple instances by translating model of object and re-rendering
- Allows specification of object in its own coordinate system
  - Don't need to define object in terms of its screen location or orientation
- Simplifies remapping of models after a change
  - E.g. animation

46

Inside Tests

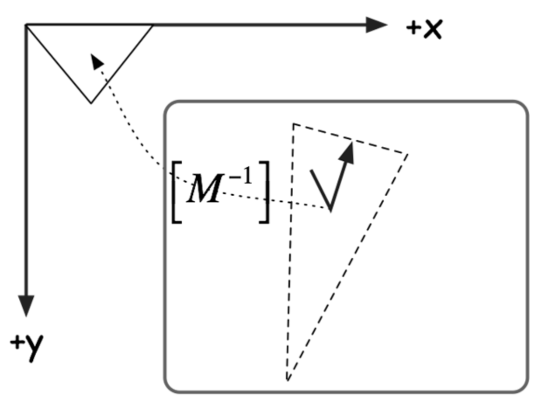
- Mouse and model must use the same coordinate system
- Two options:
  - Transform mouse
  - Transform shapes



47

Transform Mouse

- Only one transformation
- Within 3 pixels of a line in screen coordinates is how far in model coordinates?
- Uniform scaling…
- Maintaining the inverse



48

Transform Model

- Many transformations
- Manipulations (e.g. dragging) must be transformed back into model coordinates