Graphics Transformations

Translate, Scale, Rotate

Homogeneous Coordinates

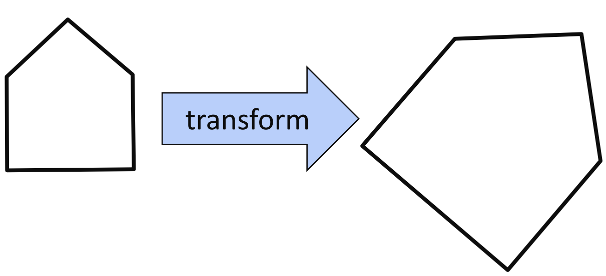
Affine Transformation Matrices

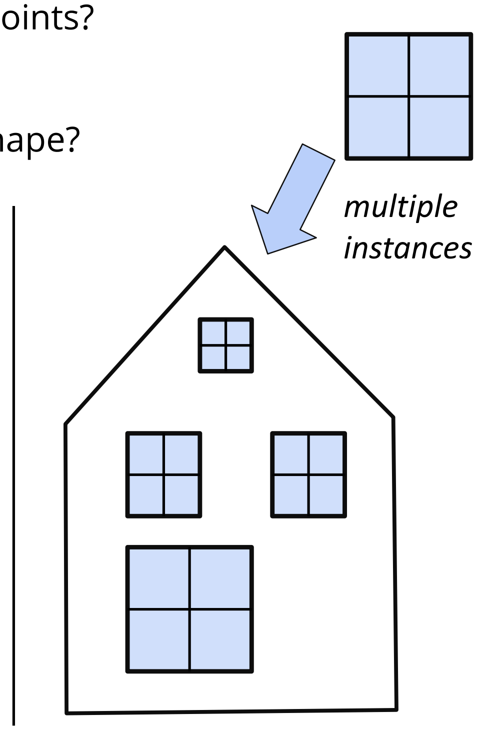
Combining Transformations

Shape Model Class

Positioning Shapes

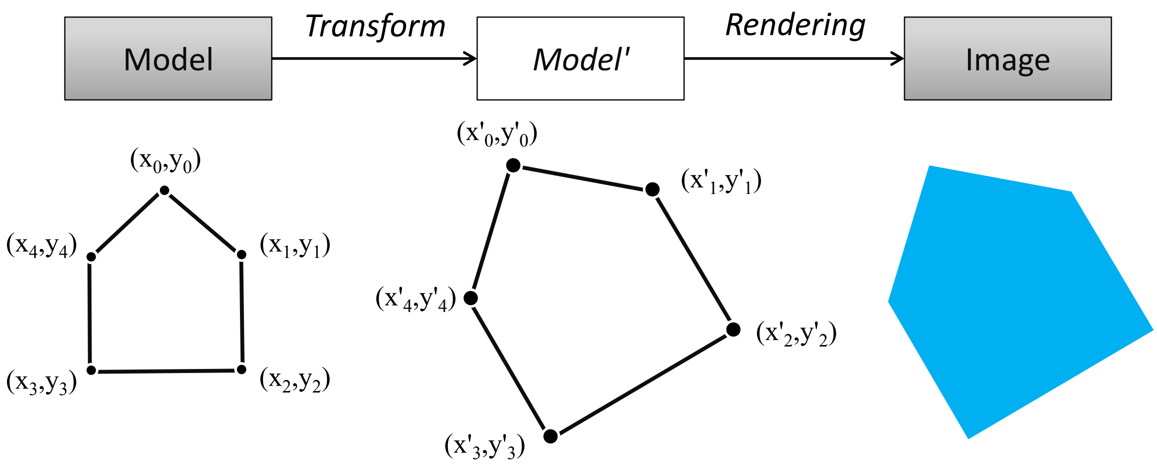
- Translate by adding offset to shape points?
- What about rotation (or scaling)?
- What if multiple instances of same shape?





Transforming Shape Models

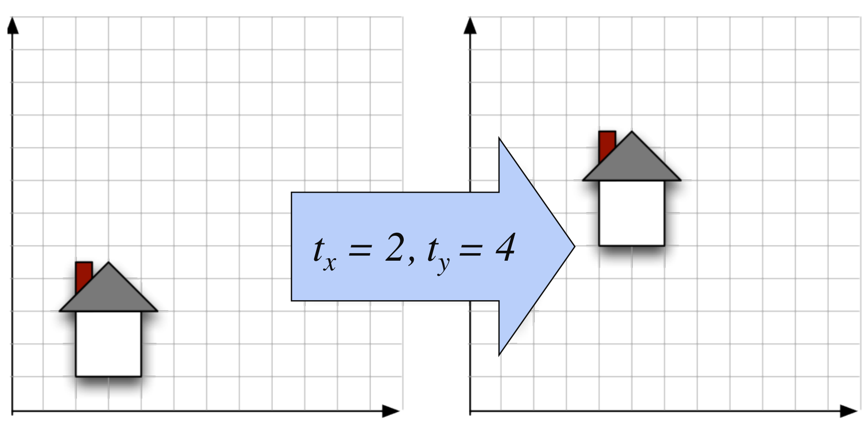
- Shape model is in a base coordinate frame
- The model is transformed to a location before rendering



$$(x'_i,y'_i) = f(x_i,y_i)$$

Translation

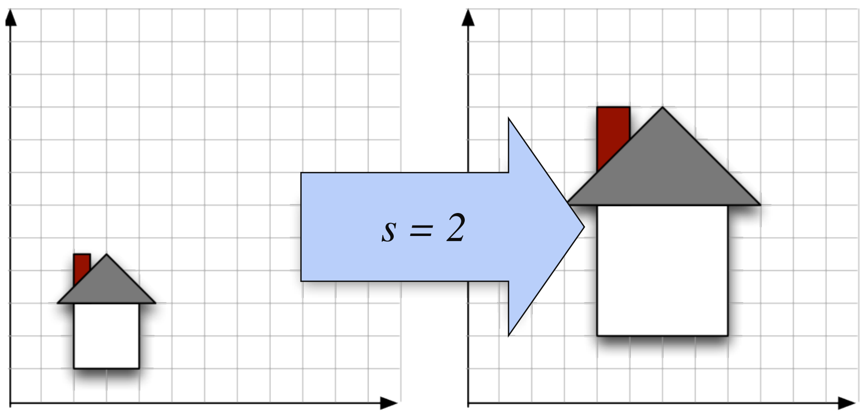
- translate: add a scalar to coordinates of each component



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

Uniform Scaling

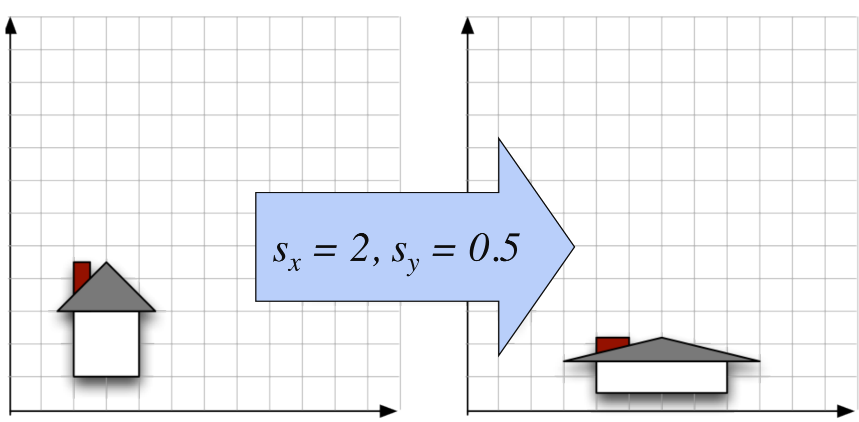
- uniform scale: multiply each component by same scalar



$$x' = x \times s$$
$$y' = y \times s$$

Non-Uniform Scaling

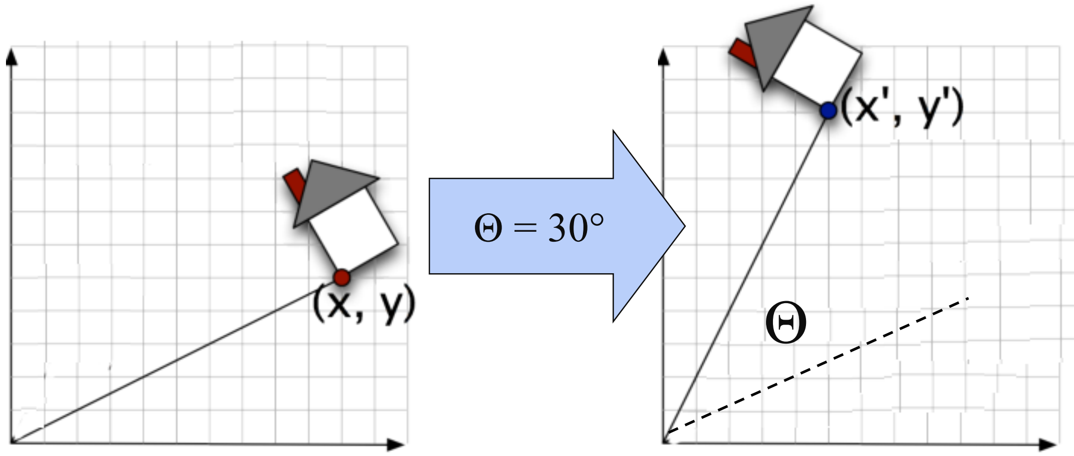
- scale: multiply each component by different scalar



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

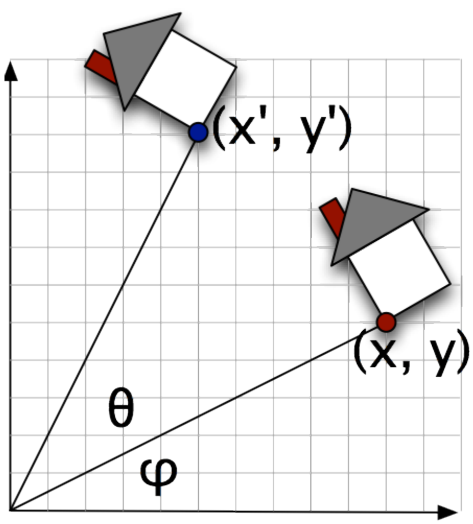
Rotation

- rotate: component is some function of x, y, Θ



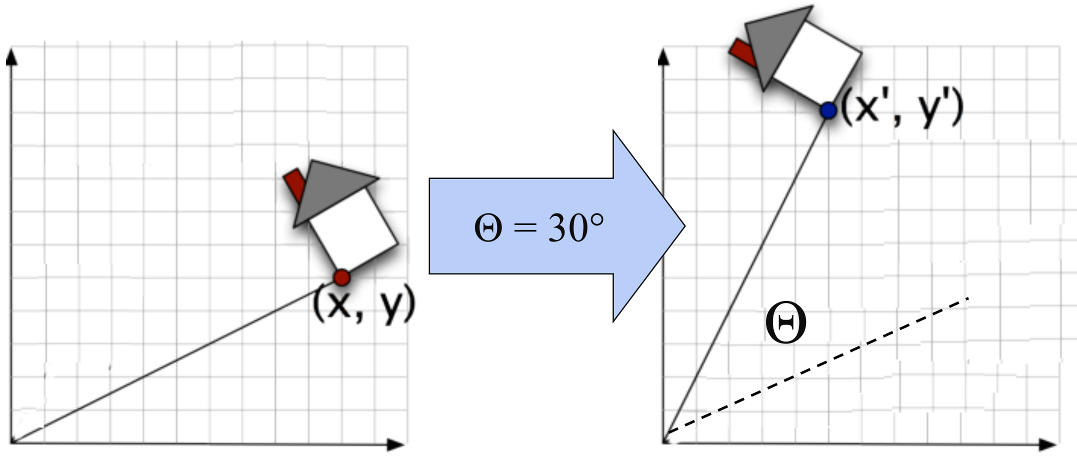
$$x' = f(x, y, \Theta)$$
$$y' = f(x, y, \Theta)$$

Deriving Rotation Function



Rotation

- rotate: component is some function of x, y, Θ



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

derivation:
https://www.student.cs.uwaterloo.ca/~cs349/f17/lectures/rotation-derivation.pdf

Combining Transformations

- Rotate:

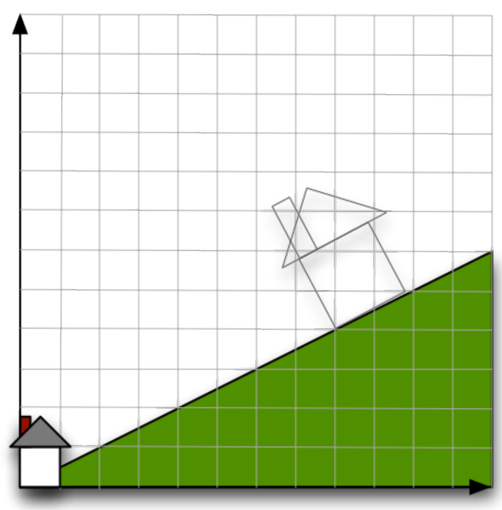
$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

- Translate:

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

- Scale:

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



Combining Transformations: Step 1 - Scale

- Rotate:

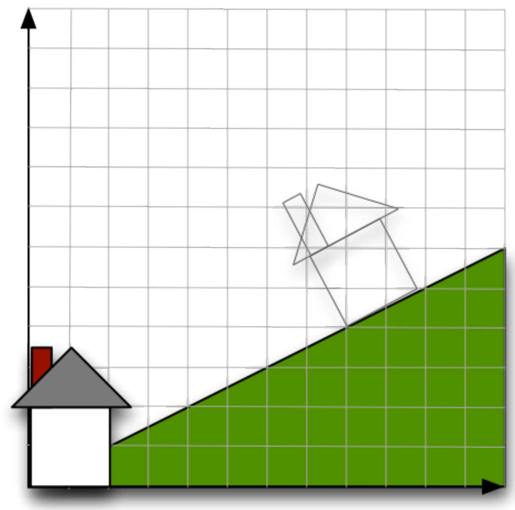
$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

- Translate:

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

- Scale:

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



$$x' = 2x$$
$$y' = 2y$$

Combining Transformations: Step 2 - Rotate

- Rotate:

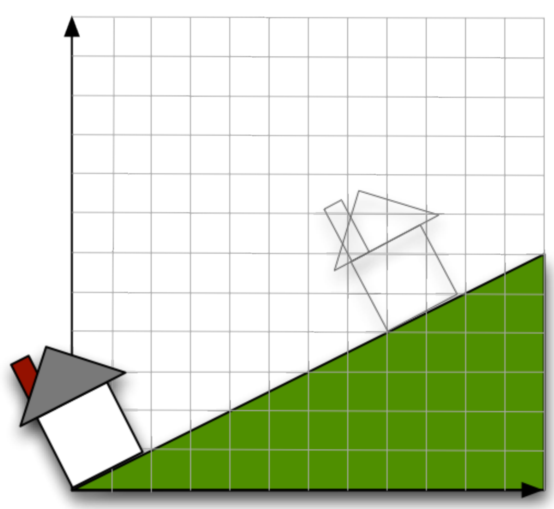
$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

- Translate:

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

- Scale:

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



$$x' = 2(x\cos(30) - y\sin(30))$$
$$y' = 2(x\sin(30) + y\cos(30))$$

Combining Transformations: Step 3 - Translate

- Rotate:

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

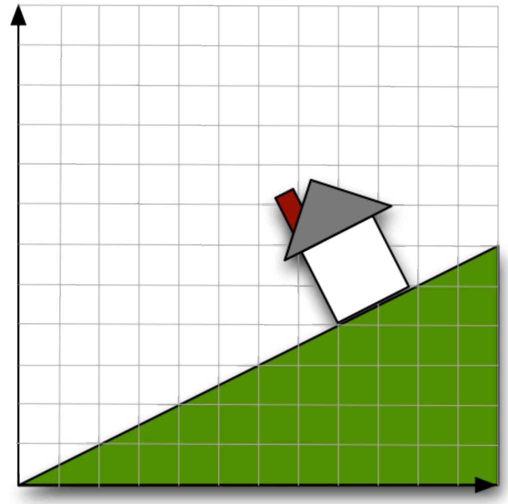
- Translate:

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

- Scale:

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

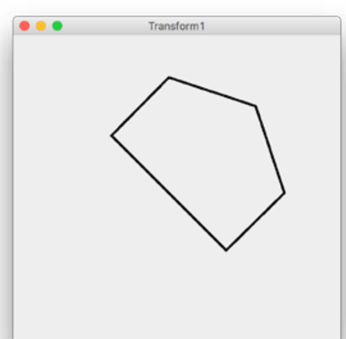
Note: Order of operations is important. What if you translate first?



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

Transform1.java

```
// the house shape model (centred at top left corner)
private Polygon s= new Polygon(new int[] {-50, 50,  50, 0, -50},
                               new int[] {75, 75, -25, -75, -25}, 5);
...

// get copy of shape
Polygon ts = new Polygon(s.xpoints, s.ypoints, s.npoints);

// transform by hand
scale(ts, 2, 1);
rotate(ts, 45);
translate(ts, M.x, M.y);

g2.setStroke(new BasicStroke(3));
g2.drawPolygon(ts.xpoints, ts.ypoints, ...);
```



NOTE: Doing transformations in this way is not the optimal.

Matrix Representation

- Goal: Represent each 2D transformation with a matri

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

- Multiply matrix by column vector ⇔ apply transformation to point

$$\begin{aligned} x' &= ax + by \\ y' &= cx + dy \end{aligned} \quad \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Matrix Representation

- Transformations can be combined by multiplication
  - transformations are associative

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

- 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 sent to a GPU to speed the process

Can a 2 x 2 Matrix Represent All 2D Transformations?

- 2D Scale around (0,0)?

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

- 2D Rotate around (0,0)?

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

- 2D Mirror about Y axis?

$$\begin{aligned} x' &= -x \\ y' &= y \end{aligned} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Can a 2 x 2 Matrix Represent All 2D Transformations? No.

- 2D Translation?

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

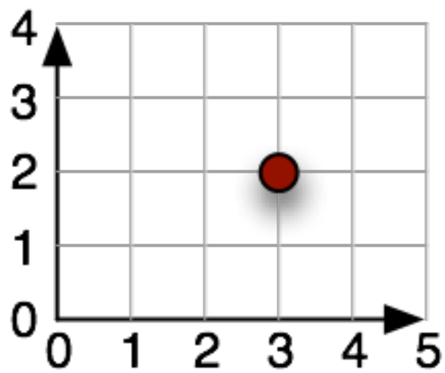
- Maybe this?

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

Problem: Only works for a specific point. Can't create a general 2x2 matrix to transform a model

Homogeneous Coordinates

- Solution: add an extra component **w** to each coordinate
- **[x, y, w]^T** represents a point at location **[x/w, y/w]^T**
- many Homogeneous points for same Cartesian point



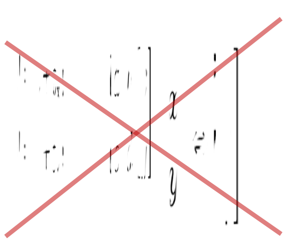
$$\begin{bmatrix}3\\2\\1\end{bmatrix} \quad \begin{bmatrix}6\\4\\2\end{bmatrix} \quad \begin{bmatrix}7.5\\5.0\\2.5\end{bmatrix} \quad \ldots$$

(divide by *w* to get Cartesian coords)

Homogeneous Coordinates

- represent coordinates in 2 dimensions with a 3-tuple (as a 3 × 1 column matrix)
- why? …

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



… need 3 columns in our transformation matrix

3 x 3 Translation Matrix

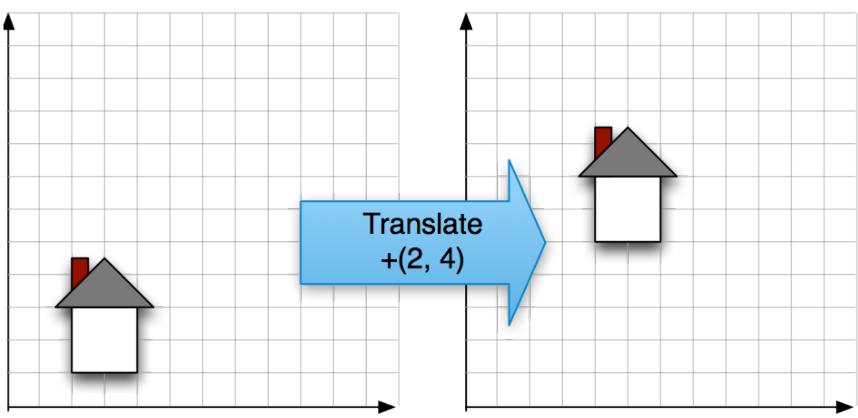
- Now we can represent 2D translation with a 3x3 matrix

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

- This 3 x 3 matrix is an Affine Transformation matrix
  – it can express any combination of translate, rotate, and scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A & B & C \\ D & E & F \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} Ax + By + C \\ Dx + Ey + F \\ 1 \end{bmatrix}$$

3 x 3 Translation Matrix Example



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

Rotation and Scale

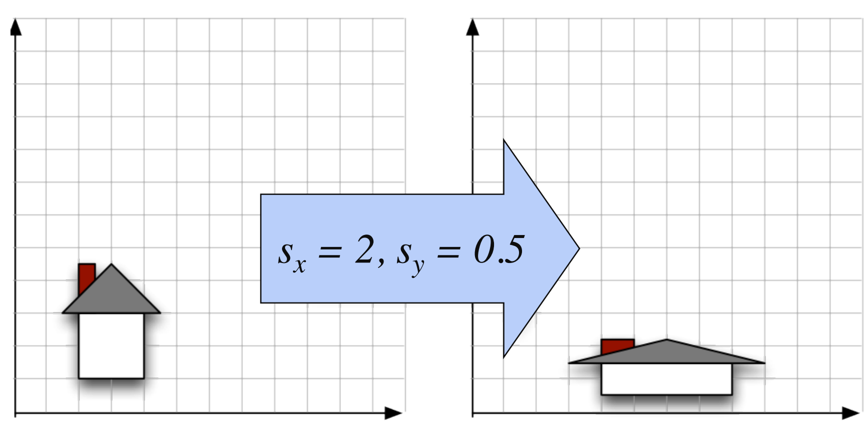
- 3 x 3 Scale Matrix

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

- 3 x 3 Rotation Matrix

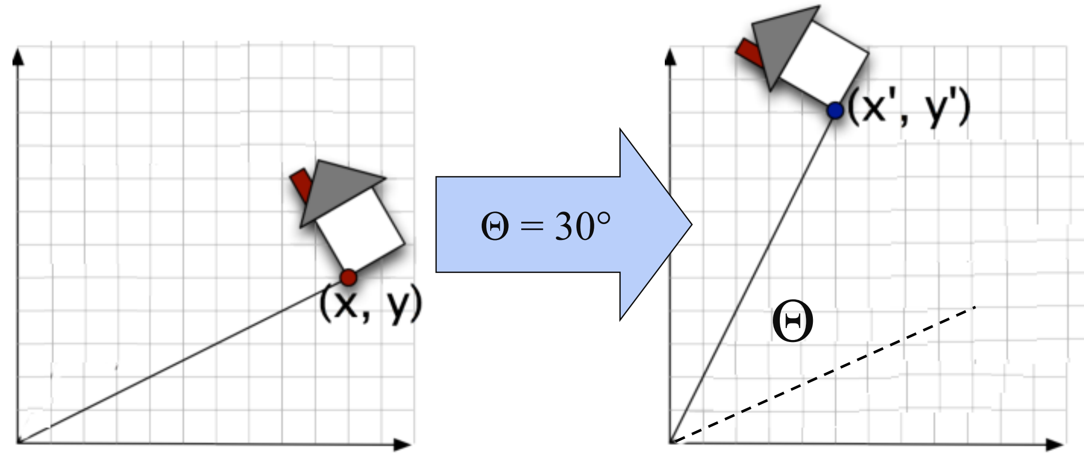
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & a \\ \sin(\theta) & \cos(\theta) & b \\ c & d & e \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos(\theta) - y\sin(\theta) \\ x\sin(\theta) + y\cos(\theta) \\ 1 \end{bmatrix}$$

3 x 3 Scale Matrix Example



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 2x \\ 0.5y \\ 1 \end{bmatrix}$$

3 x 3 Rotation Matrix Example



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(30) & -\sin(30) & 0 \\ \sin(30) & \cos(30) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(30)x - \sin(30)y \\ \sin(30)\,x + \cos(30)y \\ 1 \end{bmatrix}$$

Vector and Point Homogeneous Coordinates

$$\vec{v} + \vec{w} = \begin{bmatrix} v_{\mathrm{x}} \\ v_{\mathrm{y}} \\ 0 \end{bmatrix} + \begin{bmatrix} w_{\mathrm{x}} \\ w_{\mathrm{y}} \\ 0 \end{bmatrix} = \begin{bmatrix} v_{\mathrm{x}} + w_{\mathrm{x}} \\ v_{\mathrm{y}} + w_{\mathrm{x}} \\ 0 \end{bmatrix}$$

Add Vectors

$$\vec{v} \times s = \begin{bmatrix} v_{\mathrm{x}} \\ v_{\mathrm{y}} \\ 0 \end{bmatrix} \times s = \begin{bmatrix} v_{\mathrm{x}} \times s \\ v_{\mathrm{y}} \times s \\ 0 \end{bmatrix}$$

Scalar Multiply

$$p - q = \begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}} \\ 1 \end{bmatrix} - \begin{bmatrix} q_{\mathrm{x}} \\ q_{\mathrm{y}} \\ 1 \end{bmatrix} = \begin{bmatrix} p_{\mathrm{x}} - q_{\mathrm{x}} \\ p_{\mathrm{y}} - q_{\mathrm{x}} \\ 0 \end{bmatrix}$$

Subtract Points

$$p + \vec{v} = \begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}} \\ 1 \end{bmatrix} + \begin{bmatrix} v_{\mathrm{x}} \\ v_{\mathrm{y}} \\ 0 \end{bmatrix} = \begin{bmatrix} p_{\mathrm{x}} + v_{\mathrm{x}} \\ p_{\mathrm{y}} + v_{\mathrm{x}} \\ 1 \end{bmatrix}$$

Add Vector to Point

Translating Vectors

- Vectors have no position, so translating shouldn't change anything

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

Matrix Composition

- Transformations can be combined by matrix multiplication

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

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

transformations are applied from "right-to-left",
(call post multiplication)

$$p' = A \cdot B \cdot C \cdot p$$

$$p' = (A \cdot (B \cdot (C \cdot p)))$$

Matrix Multiplication

- Associative: A(BC) = (AB)C
- **Not Commutative:** AB ≠ BA
- Order of transformations matters (but not always …)

(blackboard …)

$$A = T(4,5) \cdot S(2,3) = \begin{bmatrix} 2 & 0 & 4 \\ 0 & 3 & 5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$B = S(2,3) \cdot T(4,5) = \begin{bmatrix} 2 & 0 & 8 \\ 0 & 3 & 15 \\ 0 & 0 & 1 \end{bmatrix}$$

"Reading" an Affine Transformation Matrix

- You can get some idea of transformations in a matrix,
  … "reading" the translation, rotation, and scale is not always easy

$$C = \begin{bmatrix} cos\Theta & -sin\Theta & 4 \\ sin\Theta & cos\Theta & 5 \\ 0 & 0 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 2cos\Theta & -3sin\Theta & 4 \\ 2sin\Theta & 3cos\Theta & 5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$E = \begin{bmatrix} 2cos\Theta & -3sin\Theta & 4cos\Theta - 5sin\Theta \\ 2sin\Theta & 3cos\Theta & 4sin\Theta - 5cos\Theta \\ 0 & 0 & 1 \end{bmatrix}$$

$$F = \begin{bmatrix} cos\Theta & -sin\Theta & 4cos\Theta - 5sin\Theta - 4 \\ sin\Theta & cos\Theta & 4sin\Theta - 5cos\Theta - 5 \\ 0 & 0 & 1 \end{bmatrix}$$

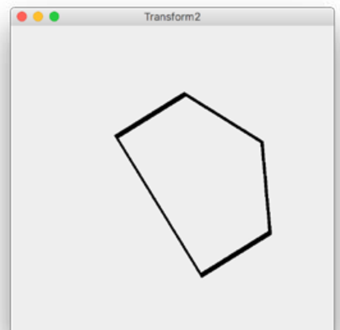
$$G = \begin{bmatrix} 2cos\Theta & sin\Theta & 4 \\ sin\Theta & 3cos\Theta & 5 \\ 0 & 0 & 1 \end{bmatrix}$$

NOTE: G is NOT a concatenation of only R(Θ), T(4,5), and S(2,3) in any order.

Transform2.java

```java
// the house shape model (centred at top left corner)
private Polygon s = new Polygon(new int[] {-50, 50, 50, 0, -50},
                                new int[] {75, 75, -25, -75, -25}, 5);
...


// the shape will get transformed when rendered
g2.translate(M.x, M.y); // T
g2.rotate(45); // R
g2.scale(2, 1); // S

g2.setStroke(new BasicStroke(3));
g2.drawPolygon(s.xpoints, s.ypoints, ...);
```

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

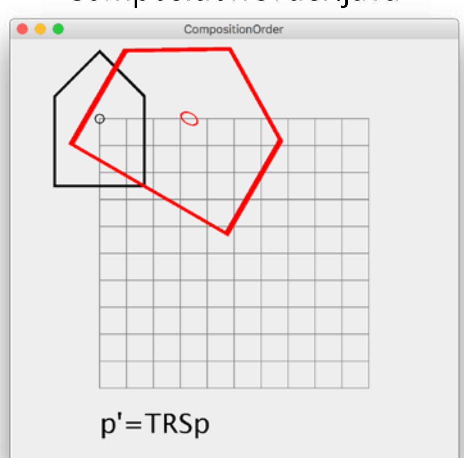


Transformation Composition Order

- Example:

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

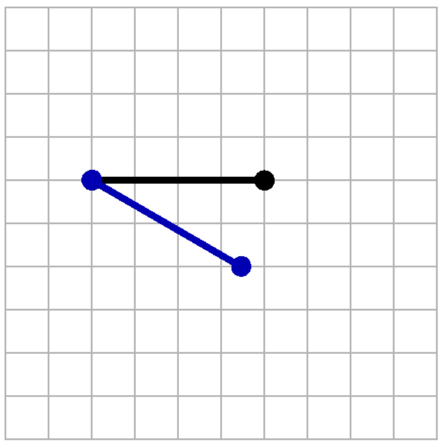
CompositionOrder.java



BarExercise.java

- Rotate the black bar about it’s left end by 30°
  - (after rotating, it should be in the blue bar position)

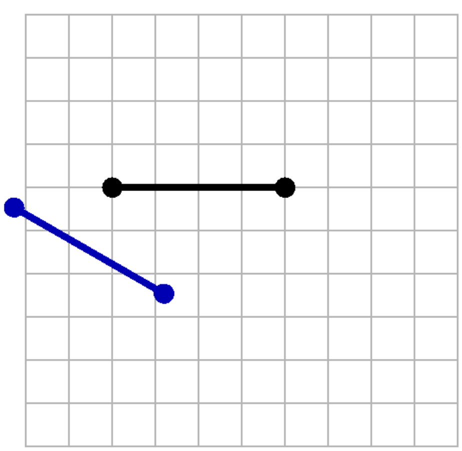
```
// left end (50, 100), right end (150, 100)
drawBar(g2, 50, 100, 150, 100);
```



Exercise: Attempt 1 (Wrong)

- Just rotate it?

```
g2.rotate(Math.toRadians(30));
drawBar(g2, 50, 100, 150, 100);
```

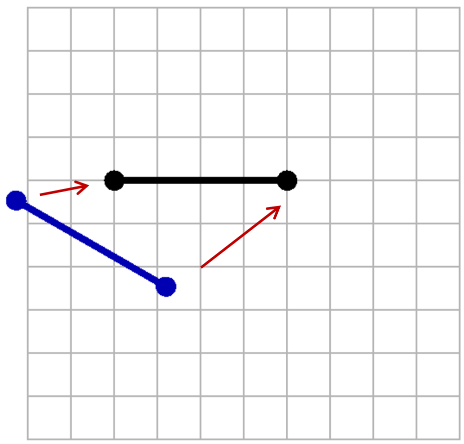


… why didn’t this work?

Exercise: Fix Attempt 1 (Wrong)

- Rotate it but fix with translations …

```
// add g2.translate(x, y) HERE?
g2.rotate(Math.toRadians(30));
// or maybe add g2.translate(x, y) HERE?
drawBar(g2, 50, 100, 150, 100);
```

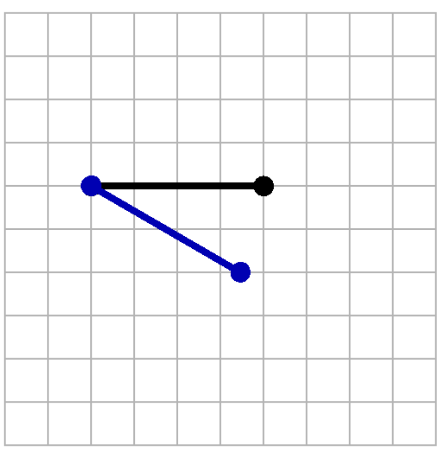


… try to correct with a translation?
but how much?

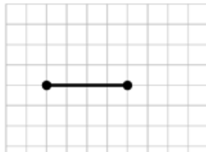
Exercise: Answer

- Scaling and rotation are matrices *are always about (0,0)*
- Need to translate to origin, rotate it, then translate back

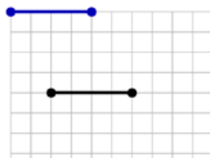
```
g2.translate(50, 100);
g2.rotate(Math.toRadians(30));
g2.translate(-50, -100);
drawBar(g2, 50, 100, 150, 100);
```



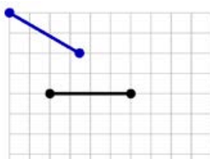
Illustration

1 

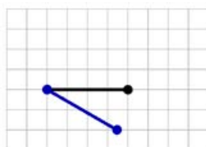
$$p' = I \cdot p$$

2 

$$p' = T_{(-50,-100)} \cdot p$$

3 

$$p' = R_{(30)} T_{(-50,-100)} \cdot p$$

4 

$$p' = T_{(50,100)} R_{(30)} T_{(-50,-100)} \cdot p$$

Shape.java

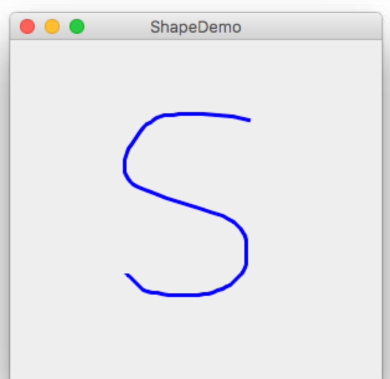
```java
// simple shape model class
class Shape {

   // shape points
   ArrayList<Point2d> points;

   // shape type
   Boolean isClosed = true;
   Boolean isFilled = true;

   // drawing attributes
   Color colour = Color.BLACK;
   float strokeThickness = 3.0f;

   // shape’s transform
   ...
```

Shape.java

```java
    ...

    public void draw(Graphics2D g2) {

       // call drawing functions
       g2.setColor(colour);
       if (isFilled) {
          g2.fillPolygon( ... );
       } else {
          g2.setStroke(new BasicStroke(strokeThickness));
          if (isClosed)
             g2.drawPolygon( ... );
          else
             g2.drawPolyline( ... );
          }
    }

    // let shape handle it's own hit testing
    public boolean hittest(double x, double y) { ... }
}
```


ShapeDemo.java

- using Shape.java
- setting attributes and drawing
- setting transform for shape



Shape.java (with scale transform)

```
// save the current g2 transform matrix
AffineTransform M = g2.getTransform();

// multiply in this shape's transform
g2.scale(scale, scale);

// call drawing functions
g2.setColor(colour);
if (isFilled) {
    g2.fillPolygon(xpoints, ypoints, npoints);
} else {
   // adjust stroke size using scale
   g2.setStroke(new BasicStroke(strokeThickness / scale));
   if (isClosed)
        g2.drawPolygon(xpoints, ypoints, npoints);
    else
        g2.drawPolyline(xpoints, ypoints, npoints);
}

// reset the transform to what it was before we drew the shape
g2.setTransform(M);
```


Useful Graphics2D methods

- Returns/sets a copy of the current Transform in the Graphics2D context.
  `AffineTransform getTransform(),`
  `void setTransform(AffineTransform Tx)`
- Concatenates the current Graphics2D Transform with a rotation transform.
  `void rotate(double theta)`
- Translates origin to (x,y), rotates, and translates origin (-x, -y).
  `void rotate(double theta, double x, double y)`
- 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 scale(double sx, double sy)`
- Concatenates the current Graphics2D Transform with a translation transform.
  `void translate(double tx, double ty)`

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)
```


Java2D AffineTransform Class

- Concatenation methods

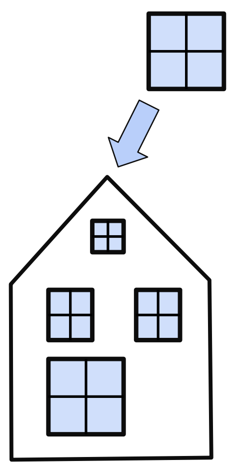
```
void rotate(double theta),
void rotate(double theta, double anchorx,
            double anchory)
void scale(double sx, double sy)
void translate(double tx, double ty)
void concatenate(AffineTransform Tx)
```

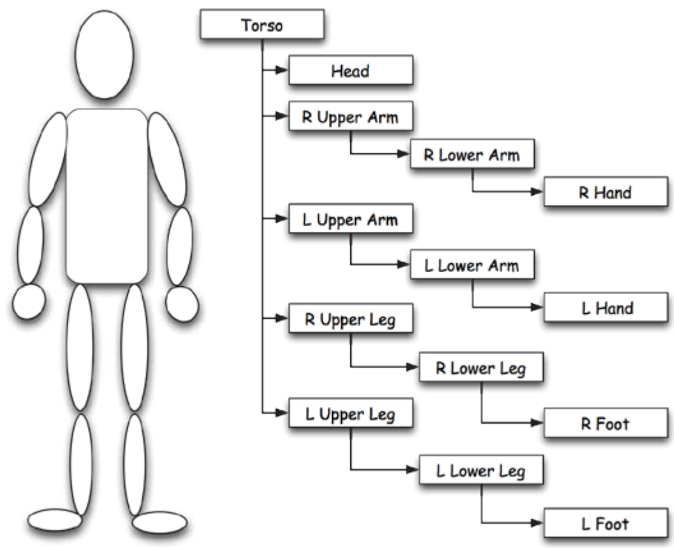
- Other Methods

```
AffineTransform createInverse() *
void transform(Point2D[] ptSrc, int srcOff,
         Point2D[] ptDst, int dstOff, int numPts)
```

Scene Graphs

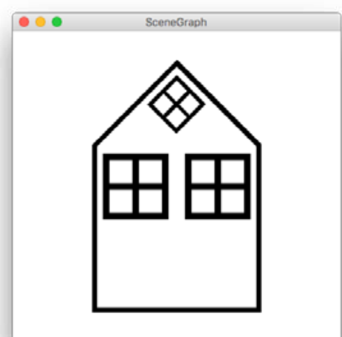
- Each part has a transform matrix
- Each part draws its children relative to itself

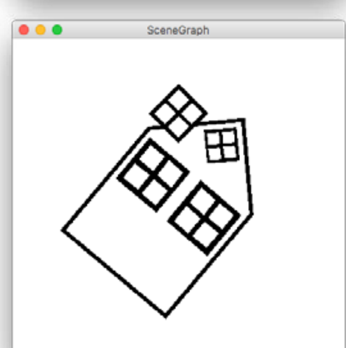




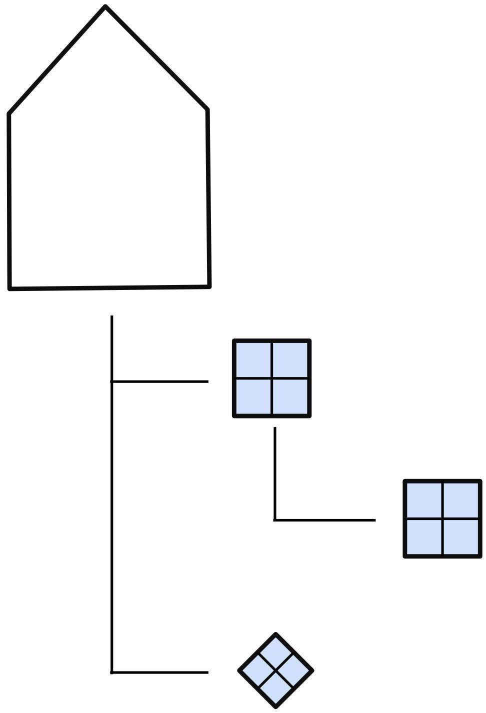
SceneGraph.java

```
// draw the house in centre of screen
g2.translate(getWidth() / 2, getHeight()/ 2);
g2.rotate(Math.toRadians(rotateBy));
g2.scale(scaleBy, scaleBy);

g2.drawPolygon(houseShape.xpoints, ... );

// save transform for later
AffineTransform save = g2.getTransform();

// in "house coordinates"
g2.translate(-25, 0); // centred 25 px
g2.scale(0.4, 0.4); // 40% house width
drawWindow(g2);

// translate to right 50 px
// (relative to last window)
g2.translate(50 / 0.4, 0);
drawWindow(g2);

// return to "House" coordinates
g2.setTransform(save);
drawWindow(g2, 0, -50, 45, 0.25);
```





Transformations in SceneGraph.java



SceneGraph.java

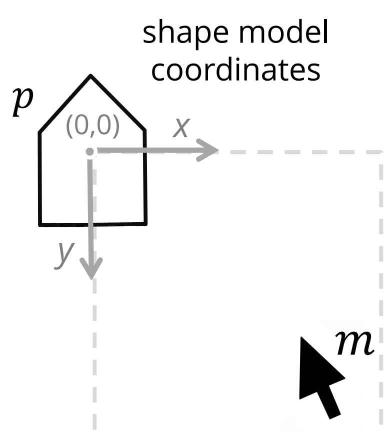
```
// draws 100 x 100 window shape centred at 0,0
void drawWindow(Graphics2D g2, double x, double y,
        double theta, double s) {

    // save the current g2 transform matrix
    AffineTransform save = g2.getTransform();

    // do the model to world transformation
    g2.translate(x, y);                  // T
    g2.rotate(Math.toRadians(theta)); // R
    g2.scale(s, s);                      // S

    // draws 100 x 100 window centred at 0,0
    drawWindow(g2);
    // reset transform to what it was before we drew the shape
    g2.setTransform(save);
}
```

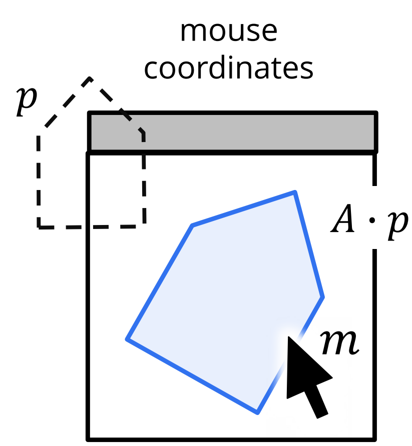
Benefits of Using Transformations

- 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

Hit-testing with Transformed Shapes

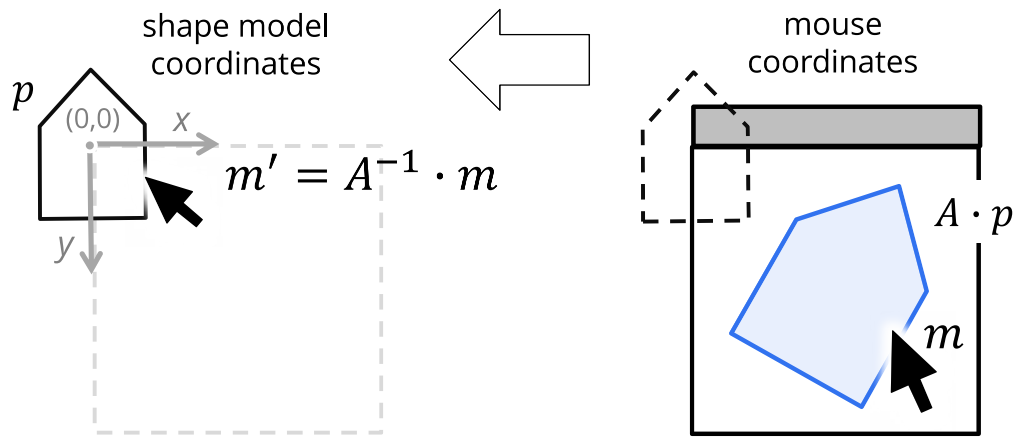
- Mouse and shape model are in different coordinate systems
- Two options for hit testing:
  1. Transform mouse to shape model coordinates
  2. Transform shape model to mouse coordinates





Transform Mouse to Shape Model Coordinates

- Only one point to transform
- Need to adjust hit-test threshold …
  - e.g. 3 pixels in mouse coords. is how far in model coords?
  - what if non-uniform scale?
- Computing inverse can be costly …



Transform Shape Model to Mouse Coordinates

- Have to transform every point or parameter in shape model within hit-test algorithm
  - extra memory, lots of extra calculation