

Introduction to Computation and Problem Solving

Class 18: *Lab*
Transformations in the 2D API

Prof. Steven R. Lerman
and
Dr. V. Judson Harward

1

Affine Transformations

- The 2D API provides strong support for *affine transformations*.
 - Affine means linear
- An affine transformation maps 2D coordinates so that the straightness and parallelism of lines are preserved.
- All affine transformations can be represented by a 3x3 floating point matrix.
- There are a number of “primitive” affine transformations that can be combined: scaling, rotation, and translation.

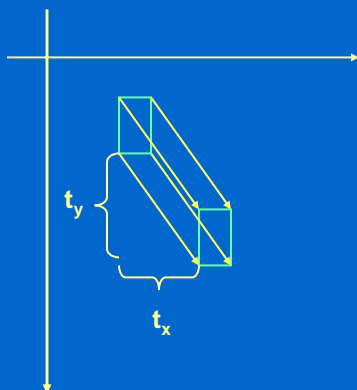
2

Transformations in the 2D API

- Transformations are represented by instances of the AffineTransform class in the java.awt.geom package.
- You can create a new AffineTransform object with its no argument constructor.
 - `AffineTransform at = new AffineTransform();`
- You can invoke any of the following methods on an AffineTransform object:
 - `at.scale(double sx, double sy)`
 - `at.translate(double tx, double ty)`
 - `at.rotate(double theta)`
 - `at.rotate(double theta, double x, double y)`
- These methods can be combined to build complex transformations.

3

Translation



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

4

Translation Example, 1

```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
public class TransformPanel extends JPanel {
    Rectangle2D.Double rect;
    public TransformPanel() {
        rect = new Rectangle2D.Double(0, 0, 50, 100);
    }
    public void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        g2.setPaint(Color.BLUE);
        AffineTransform baseXf = new AffineTransform();
        // Shift to the right 50 pixels, down 50 pixels
        baseXf.translate(50,50);
        Shape s = baseXf.createTransformedShape( rect );
        g2.fill(rect);
    }
}
```

5

Applying Transforms

- You can apply a transformation to one or more Shapes and draw the result:

```
AffineTransform baseXf = new AffineTransform();
baseXf.translate( 50, 50 );
rect = new Rectangle2D.Double(0, 0, 50, 100);
Shape transformed =
    baseXf.createTransformedShape( rect );
g2.fill(transformed);
```

- Or you can transform the coordinate space and everything in it:

```
g2.transform(baseXf);
g2.fill(rect);
```

6

Translation Example, 2

```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
public class TransformPanel extends JPanel {
    Rectangle2D.Double rect;
    public TransformPanel() {
        rect = new Rectangle2D.Double(0, 0, 50, 100);
    }
    public void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        g2.setPaint(Color.BLUE);
        AffineTransform baseXf = new AffineTransform();
        // Shift to the right 50 pixels, down 50 pixels
        baseXf.translate(50,50);
        g2.transform(baseXf);
        g2.fill(rect);
    }
}
```

7

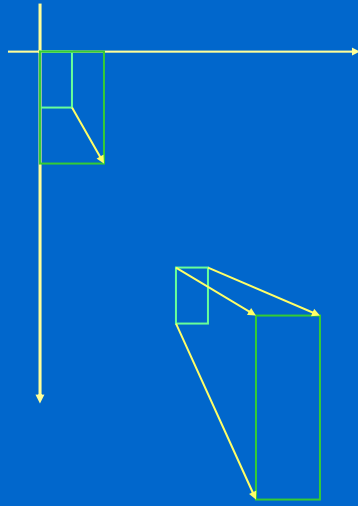
Translation Example, 3

We could use this main method to create and display a TransformPanel in a JFrame:

```
import javax.swing.*;
public class TransformMain {
    public static void main(String[] args) {
        JFrame f = new JFrame("Transform");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel p = new TransformPanel();
        f.getContentPane().add(p);
        f.setSize(700,700);
        f.setVisible(true);
    }
}
```

8

Scaling



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

9

Scaling Exercise

- First, write code to scale `rect` at the origin using `TransformPanel` as a basis.
- Follow the same steps you saw in the translation exercise. Instead of `translate`, invoke the `scale` method. It takes two arguments: the first for scaling `x`, the second for `y`.
- Invoke `scale` with numbers that are > 1 . Try numbers < 1 and > 0 . Finally, try numbers that are < 0 .
- Next, modify `rect` so that it is not at the origin. How does `scale` act on shapes that aren't at the origin?

10

Scaling Solution

```
public class TransformPanel extends JPanel {
    Rectangle2D.Double rect;
    public TransformPanel() {
        rect = new Rectangle2D.Double(0, 0, 50, 100);
        // or, not at origin:
        // rect = new Rectangle2D.Double(100,200,50,100);
    }
    public void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        g2.setPaint(Color.BLUE);
        AffineTransform baseXf = new AffineTransform();
        baseXf.scale(1.25,1.25);
        g2.transform(baseXf);
        g2.fill(rect);
    }
}
```

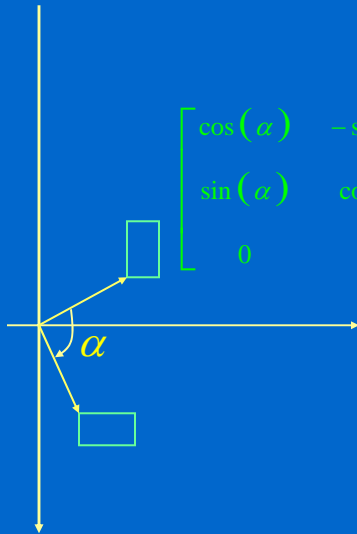
11

Scaling Notes

- Basic scaling operations take place with respect to the origin. If the shape is at the origin, it grows. If it is anywhere else, it grows and moves.
- s_x , scaling along the x dimension, does not have to equal s_y , scaling along the y.
- For instance, to flip a figure vertically about the x-axis, scale by $s_x=1$, $s_y=-1$.

12

Rotation



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

13

Rotation Exercise

- Write code to rotate `rect` as described in the prior slide using `TransformPanel` as a basis.
- Follow the same steps as you did in the scaling exercise. Invoke `basexf.rotate` with a single argument: the angle, in radians, to rotate the rectangle. This method will appear to both rotate and move the rectangle with respect to the origin.
- You might find `Math.PI` or `Math.toRadians(double degrees)` useful.
- To avoid rotating `rect` completely out of view, rotate by only a small amount.
- How does rotating `rect` change when `rect` is at the origin? When it isn't?

14

Rotation Solution

```
public void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    g2.setPaint(Color.BLUE);
    AffineTransform baseXf = new AffineTransform();
    // Rotate counter clockwise with
    // respect to the origin
    baseXf.rotate(-Math.PI/12);
    g2.transform(baseXf);
    g2.fill(rect);
}
```

15

Composing Transformations

- Suppose we want to scale point (x, y) by 2 and then rotate by 90 degrees.

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right)$$

rotate

scale

16

Composing Transformations, 2

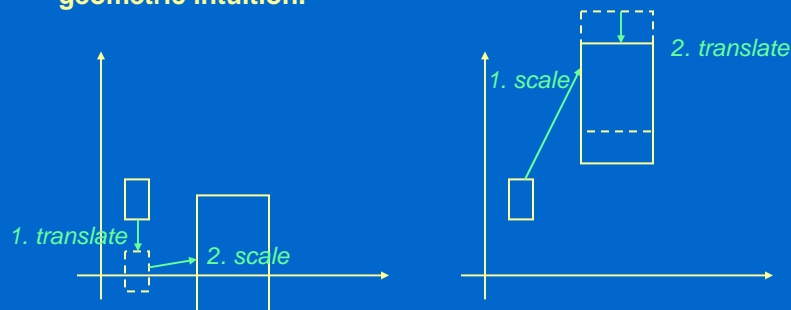
Because matrix multiplication is associative, we can rewrite this as

$$\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\ = \begin{pmatrix} 0 & -2 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

17

Composing Transformations, 3

Because matrix multiplication does not regularly commute, the order of transformations matters. This squares with our geometric intuition.



If we invert the matrix, we reverse the transformation.

18

Maximally Confusing Point!!

- When you are using transforms to transform the whole coordinate space, they compose as you would expect.
- When you are using transforms to transform **an object**, they compose in the **reverse** order to that in which you applied them.
- The following will scale first, then translate.

```
AffineTransform xf = new AffineTransform();  
xf.translate( 100, 50 );  
xf.scale( 2, 2 );  
xf.createTransformedShape( rect );
```

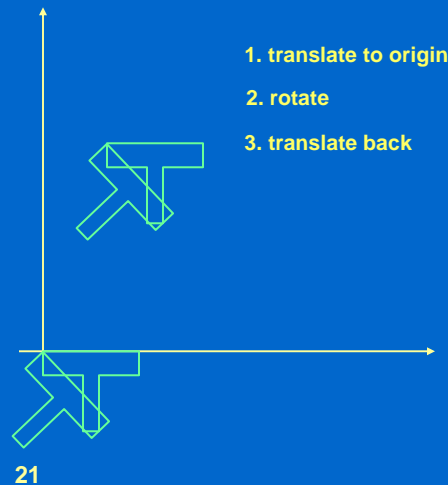
19

Transformations and the Origin

- When we transform a shape, we transform each of the defining points of the shape, and then redraw it.
- If we scale or rotate a shape that is not anchored at the origin, it will translate as well.
- If we just want to scale or rotate, then we should translate back to the origin, scale or rotate, and then translate back.

20

Transformations and the Origin, 2



Transformation Shortcuts

- Every `Graphics2D` object has an associated `AffineTransform`. When we invoke `g2.transform(baseXf)`, we are changing this associated transform by composing it with `baseXf`.
- There are methods in the `Graphics2D` class that allow us to directly affect the transform associated with `g2`:
 - `g2.translate(double tx, double ty);`
 - `g2.scale(double sx, double sy);`
 - `g2.rotate(double theta);`
 - `g2.rotate(double theta, double x, double y)`
- These methods represent an alternative, simple way to apply transforms.

22

Compass Exercise

- Download and review `Compass.java`
- This program listens for mouse clicks and when they occur, it points the red arrow in the direction of the click.
- In its current form, the program is not functional: the arrow never moves.
- Note that we calculate `theta` (the angle of the mouse click with respect to the origin) for you already.

23

Compass Hints

- Update `paintComponent` so the arrow points in the direction of each mouse click.
- To do this, you only need to add a single line of code. This line will invoke `rotate` on `g2` so that it properly draws the arrow when `g2.fill(arrow)` is invoked.
- When should `rotate` be invoked? Before, after, or between the `translate` and `scale`?
- What argument should you pass to `rotate`? Remember to take the orientation of the arrow right before you `rotate`.

24