cis3210
design and implementation of software applications I
spring 2015
lecture # II.5: graphics systems

topics:

- affine transformations

references:

- Oracle Java SE 2D Tutorial

affine transformations

- *affine transformations* are generalizations of *geometric transformations* that allows for manipulation of a graphic object easily by using mathematics to transform the coordinates of the points of the object
- three basic transformations will be discussed:
  - *translating* is shifting an object to a new location, without changing its size or shape
  - *scaling* is changing the size of an object
  - *rotation* is turning an object about a particular point, without changing the object's size or shape
- although we'll cover 2-dimensional transformations only, the same concepts apply for 3 (or more) dimensions
- for each type of transformation, there are a set of matrix equations that can be applied to ease the computation

basics

- every point in 2-dimensional space is represented as a *vector* $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ or in 3-dimensional space $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$
- a *matrix* is just like a vector with more than one column, e.g.: $\begin{bmatrix} 1 & 5 & 3 \\ 8 & 1 & 4 \\ 2 & 9 & 1 \end{bmatrix}$
- in order to be able to multiply two matrices together, the number of rows in one has to be the same as the number of columns in the other
- the general formula for matrix multiplication is:

$$C_{(row,col)} = \sum_{s=1}^{m} A_{(row,s)} * B_{(s,col)}$$

where A is an $n \times m$ matrix and B is an $m \times p$ matrix and C is an $n \times p$ matrix

- so if we want to multiply two 2-dimensional matrices together: $\begin{bmatrix} a_{11}a_{12} \\ a_{21}a_{22} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$
- applying the formula, and then substituting back the original matrix values, we get:

$C_{(1,1)} = \Sigma_{s=1}^{1} A_{(1,s)} * B_{(s,1)} = A_{(1,1)} * B_{(1,1)} + A_{(1,2)} * B_{(2,1)}$
$C_{(2,1)} = \Sigma_{s=1}^{1} A_{(2,s)} * B_{(s,1)} = A_{(2,1)} * B_{(1,1)} + A_{(2,2)} * B_{(2,1)}$

in other words:
$= \begin{bmatrix} a_{11} * b_1 + a_{12} * b_2 \\ a_{21} * b_1 + a_{22} * b_2 \end{bmatrix}$

translation

- *translating* is shifting an object to a new location, without changing its size or shape
- to translate (shift, move) the object in each direction use Δx and Δy values, respectively
- to translate point (x, y) by Δx units along the x axis and Δy units along the y axis
- because transformations are often done in conjunction with other transformations (which require matrix multiplication) translation is expressed in the following matrix form (2D):

$$T = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix}$$

- when applied:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x * 1 + y * 0 + 1 * \Delta x \\ x * 0 + y * 1 + 1 * \Delta y \\ x * 0 + y * 0 + 1 * 1 \end{bmatrix} = \begin{bmatrix} x + \Delta x \\ y + \Delta y \\ 1 \end{bmatrix}$$

scaling

- *scaling* is changing the size of an object
- this can be done *uniformly*, where the size of all dimensions change by the same amount
- or *non-uniformly*, where the size of each dimension changes by a different amount, s_x and s_y, which represent how much we want to scale the object in the x and y dimensions, respectively. Scaling is represented in matrix form:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

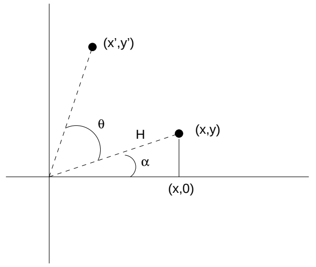
- when applied:

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

- this operation shifts the object if the object is not first moved to the origin. (e.g. imagine a line from point $(1, 2)$ to $(2, 4)$ scaled to $s_x = s_y = 2$. the result will be a line $(2, 4)$ to $(4, 8)$)

rotation

- *rotation* is turning an object about a particular point, without changing the object's size or shape
- typically, rotation is computed about the origin



- the amount of rotation is expressed in terms of an angle, θ

- the maxtrix form for rotating a point by θ about the origin:

$$R = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- when applied:

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

$$= \begin{bmatrix} x * cos(\theta) + y * sin(\theta) \\ y * cos(\theta) - x * sin(\theta) \\ 1 \end{bmatrix}$$

- in order to rotate an object about its center point, you first have to translate the object so that the center point is at the origin, perform the rotation and then translate the object back to where it came from

combining transformations

- complex graphics objects are composed of lots of points, therefore a common method to transform these objects is:
  - first multiply all the transformation matrices and
  - then use the result matrix to transform each point
- matrix multiplication is not *commutative*, in other words:
  $TS \neq ST$
- therefore the **order matters**. example:

$$TS = \begin{bmatrix} 1+0+\Delta x \\ 0+1+\Delta y \\ 0+0+1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & s_y & \delta x \\ 0 & s_y & \delta y \\ 0 & 0 & 1 \end{bmatrix}$$

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

Java Graphics2D transformations

- `Graphics2D` class contains a `transform` attribute, which is an instance of `AffineTransform` class in `java.awt.geom` package.
- by modifying `transform` attribute you can translate, scale and rotate graphics primitives when they are rendered.
- usual steps for performing transformations with Graphics2D
  - save the current state of the `transform` attribute:

    ```
    AffineTransform savedTransform = g2.getTransform();
    ```
  - change the `transform` by concatenating `translate`, `scale`, `rotate` operations, example:

    ```
    g2.translate(10,40);
    g2.rotate(Math.toRadians(50));
    g2.scale(1.3, 0.7);
    ```
  - restore the `transform` to its initial state:

    ```
    g2.setTransform(savedTransform);
    ```

Java Graphics2D transformation example

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class TransformDemo {
  public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
          TransformDemo demo = new TransformDemo();
        }
      });
  }

  public TransformDemo() {
    frame = new JFrame("Transformation Demo");
    DrawRegion drawRegion = new DrawRegion();
    frame.getContentPane().add(drawRegion);
```

```
    frame.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }

  private JFrame frame;
  private static final int WINDOW_WIDTH = 600;
  private static final int WINDOW_HEIGHT = 500;

  class DrawRegion extends JPanel {
    public void paintComponent(Graphics g) {
      super.paintComponent(g);
      setBackground(Color.WHITE);
      Graphics2D g2 = (Graphics2D) g;

      int rect_width = 50;
      int rect_height = 30;

      Ellipse2D oval = new Ellipse2D.Float(0, 0, rect_width, rect_height);
      Rectangle2D rect = new Rectangle2D.Float(0, 0, rect_width, rect_height);
```

```
Ellipse2D center = new Ellipse2D.Float((WINDOW_WIDTH - 5)/2,
                                       (WINDOW_HEIGHT - 5)/2,
                                       10,
                                       10);
g2.setPaint(Color.ORANGE);
g2.fill(center);

g2.setPaint(Color.RED);
g2.draw(rect);
g2.fill(oval);

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

// change the transformation to move the ellipse
g2.translate((WINDOW_WIDTH - rect_width)/2, (WINDOW_HEIGHT - rect_height)
g2.rotate(Math.toRadians(45));
g2.scale(2,2);
g2.setPaint(Color.BLUE);
```

```
   g2.draw(rect);
   g2.fill(oval);

   // reset
   g2.setTransform(savedTransform);

   // change the transformation to move the rectangle
   g2.translate(350,150);
   g2.rotate(Math.toRadians(-45));
   g2.scale(1.5,1);
   g2.setPaint(Color.GREEN);
   g2.draw(rect);
   g2.fill(oval);

   // reset
   g2.setTransform(savedTransform);
  }
 }
}
```