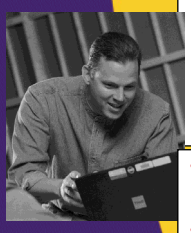
Android Programming: 2D Drawing Part 2: Coordinate Transformations and ShapeDrawable

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/android-tutorial/

© 2012 Marty Hall

For live Android training, please see courses at http://courses.coreservlets.com/.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this Android tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- **Canvas coordinate transformations**
  - translate
  - rotate
  - skew
  - scale
- **Using ShapeDrawable**
  - Motivation
  - Making a ShapeDrawable
  - Drawing a ShapeDrawable

Coordinate Transformations

Summary

- **Idea**
  - Instead of computing new coordinates, move the coordinate system itself.
    - For example, to draw text on clock face, don't rotate and position each string. Instead, translate to center, rotate Canvas 22.5 degrees, then draw string in same place.
- **Java Syntax**

```
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    canvas.translate(…);
    canvas.rotate(…);
    canvas.scale(…);
    canvas.skew(…);
    canvas.drawBlah(…);
}
```


Coordinate Transformations: Available Operations

- **translate**
  - Move drawing. Positive x means to move drawing to the right
- **rotate**
  - Spin around current center, which can be changed by translate. In degrees, unlike Java 2D (desktop Java) which uses radians
- **scale**
  - Stretch evenly.
- **skew**
  - Stretch more as points get further from origin.
    - Called "shear" in some other libraries.
- **concat/setMatrix**
  - Do several transformations to Matrix, then apply Matrix. Usually use "concat" to add Matrix to existing one, rather than "setMatrix" to reset.
- **save/restore**
  - Save/restore the current transformation Matrix. Rather than reversing a long series of transformations, just save, do the transformations, then restore.

Example: Rotated Text

Example Summary

- **Idea**
  - Make View that draws text rotated around a central point
    - Uses “translate” to shift the origin to the center of the View
    - Then uses “rotate” to spin around that point
    - Draws the strings with drawText
    - Uses anti-aliasing in the Paint
  - Uses a fixed-size font
    - So, just making a guess about what text size will fit. In section on Custom Components, We will discuss several improvements to this approach
      - Change the font size to fit the window
      - Let the XML file specify the font size
      - Use onMeasure to compute desired sizes, so that the View does not have to be in fixed-size region (i.e., it can use wrap_content instead of match_parent)

View: General Class Structure

```
public class RotateTextView extends View {
    private Paint mPaint = makePaint(Color.BLUE);
    private String mMessage = "Android";

    public RotateTextView(Context context) {
        super(context);
    }

    public RotateTextView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        ...
    }
    ...
}
```


View: onDraw

```
public class RotateTextView extends View {
    ...

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        int viewWidth = getWidth();
        int viewHeight = getHeight();
        canvas.translate(viewWidth/2, viewHeight/2);
        for(int i=0; i<10; i++) {
            canvas.drawText(mMessage, 0, 0, mPaint);
            canvas.rotate(36);
        }
    }

    ...
}
```

Remember that rotate uses degrees, whereas Java 2D in desktop Java uses radians.

View: Making the Paint

```
public class RotateTextView extends View {
    private Paint mPaint = makePaint(Color.BLUE);
    private String mMessage = "Android";

    ...

    private Paint makePaint(int color) {
        Paint p = new Paint(Paint.ANTI_ALIAS_FLAG);
        p.setColor(color);
        p.setTextSize(50);
        return(p);
    }
}
```

Angled lines look better with anti-aliasing.

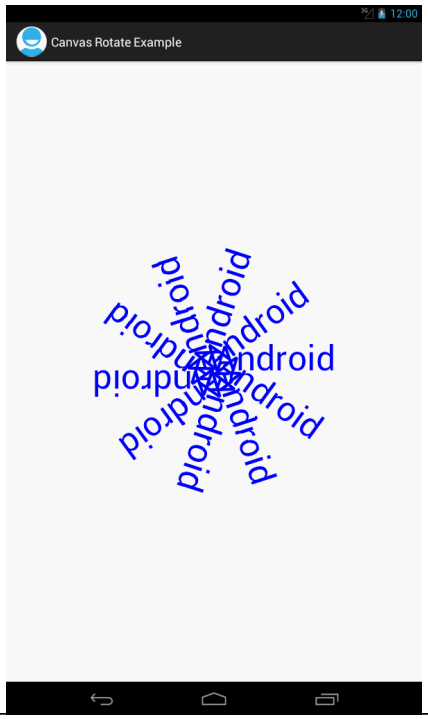
Layout File for Example (activity_rotate.xml)

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <view
        class="com.coreservlets.drawing.RotateTextView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Activity for Example

```
public class RotateActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_rotate);
    }
}
```


Results



Example: Skew

Example Summary

- **Idea**
  - Make View that draws rectangles with various amounts of x skew
- **Meaning of skew**
  - X skew
    If you specify a non-zero x skew, then x values will be more and more shifted to the right the farther they go in the y direction. For example, an x skew of 0.1 means that the x value will be shifted 10% of the distance the point is away from the x axis.
  - Y skew
    Points are shifted down in proportion to the distance they are to the right (x direction) from the y axis.

View: General Class Structure

```
public class SkewShapeView extends View {
    private Paint mBlackPaint = makePaint(Color.BLUE);

    public SkewShapeView(Context context) {
        super(context);
    }

    public SkewShapeView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        ...
    }
    ...
}
```


View: onDraw

```
    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        int viewWidth = getWidth();
        int viewHeight = getHeight();
        int deltaW = viewWidth/36;
        int rectWidth = 4*deltaW;
        int deltaH = 5;
        int rectHeight = viewHeight-2*deltaH;
        int left = deltaW;
        int top = deltaH;
        for(int i=0; i<5; i++) {
            canvas.drawRect(left, top, left+rectWidth,
                            top+rectHeight, mBlackPaint);
            left = left+6*deltaW;
            canvas.skew(0.1f, 0);
        }
    }
```

View: Making the Paint

```
public class SkewShapeView extends View {
    private Paint mBlackPaint = makePaint(Color.BLUE);

    ...

    private Paint makePaint(int color) {
        Paint p = new Paint(Paint.ANTI_ALIAS_FLAG);
        p.setColor(color);
        p.setTextSize(50);
        return(p);
    }
}
```

Again, angled lines look better with anti-aliasing.

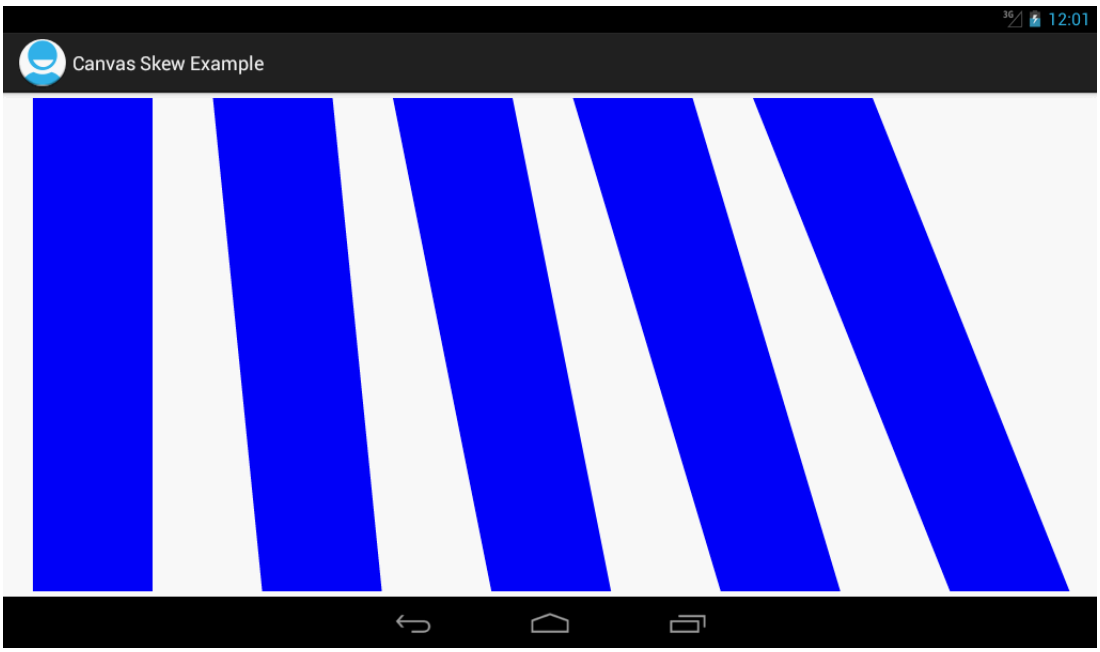
Layout File for Example (activity_skew.xml)

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <view
        class="com.coreservlets.drawing.SkewShapeView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Activity for Example

```
public class SkewActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_skew);
    }
}
```


Results



With s skew, points are shifted to the *right* the further they are *down* in the y direction. There is no y skew in this example.

Using ShapeDrawable

Summary: Idea

- **Idea**
  - Instead of directly using canvas.draw*Blah*, make a ShapeDrawable object, then tell it to draw itself on the Canvas.
- **Advantages**
  - By keeping a List of ShapeDrawable objects and having onDraw draw them all, you can
    - "Remember" the drawings when onDraw is re-invoked
      - E.g., after invalidate
    - Remove some of the drawings later
      - E.g., based on user actions. Note that ShapeDrawable has a getBounds() method, and Bounds has a contains(x,y) method.
    - Change the characteristics of the drawings later
      - E.g., keep same shape but change colors

Summary: Java Syntax

- **Making shapes**

```
public boolean onSomeUserEvent(BlahEvent event) {
    Shape shape = new OvalShape(…);  // Or other Shape
    ShapeDrawable shapeD = new ShapeDrawable(shape);
    shapeD.setBounds(…);
    shapeD.getPaint().setColor(…);
    shapeList.add(shapeD);
    invalidate();
    return(true);
}
```

- **Drawing shapes**

```
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    for(ShapeDrawable shape: shapes) {
      shape.draw(canvas);
    }
  }
```


Example: Pressing to Add or Delete Shapes

Example Summary

- **Idea**
  - When user touches screen on empty space
    - Make a ShapeDrawable
    - Add it to List
    - Call invalidate (triggers onDraw)
  - When user touches screen on top of existing shape
    - Loop down List of ShapeDrawables
    - Check if touch position is inside bounds
      - If so, remove shape from List
      - If not, add a new shape as above
    - Call invalidate (triggers onDraw)
  - onDraw
    - Loops down List of ShapeDrawables and draws them all

View: General Class Structure and onDraw

```
public class ShapeDrawableView extends View {
    private List<ShapeDrawable> shapes =
        new ArrayList<ShapeDrawable>();
    private Integer[] mColors =
        { Color.BLACK, Color.BLUE, Color.GREEN, Color.RED };

    public ShapeDrawableView(Context context) {
        super(context);
    }
    public ShapeDrawableView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        for(ShapeDrawable shape: shapes) {
            shape.draw(canvas);
        }
    }
    ...
```

View: Handling Touch Events

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        int x = (int)event.getX();
        int y = (int)event.getY();
        if (!isDeletingExistingShape(x, y)) {
            shapes.add(makeShapeDrawable(x, y));
        }
        invalidate();
        return (true);  // Handled touch event
    } else {
        return (false); // Did not handle touch event
    }
}
```


View: Removing Existing Shapes

```
private boolean isDeletingExistingShape(int x, int y) {
    for(ShapeDrawable shape: shapes) {
        Rect bounds = shape.getBounds();
        if (bounds.contains(x, y)) {
            shapes.remove(shape);
            return(true);
        }
    }
    return(false);
}
```

View: Making ShapeDrawable Objects

```
private ShapeDrawable makeShapeDrawable(int x, int y) {
    int maxWidth = getWidth()/10;
    int maxHeight = getHeight()/10;
    Shape shape;
    if (Math.random() < 0.5) {
        shape = new OvalShape();
    } else {
        shape = new RectShape();
    }
    ShapeDrawable shapeD = new ShapeDrawable(shape);
    int width = RandomUtils.randomInt(maxWidth)+5;
    int height = RandomUtils.randomInt(maxHeight)+5;
    shapeD.setBounds(x-width/2, y-height/2,
                     x+width/2, y+height/2);
    shapeD.getPaint().setColor
                (RandomUtils.randomElement(mColors));
    return(shapeD);
}
```

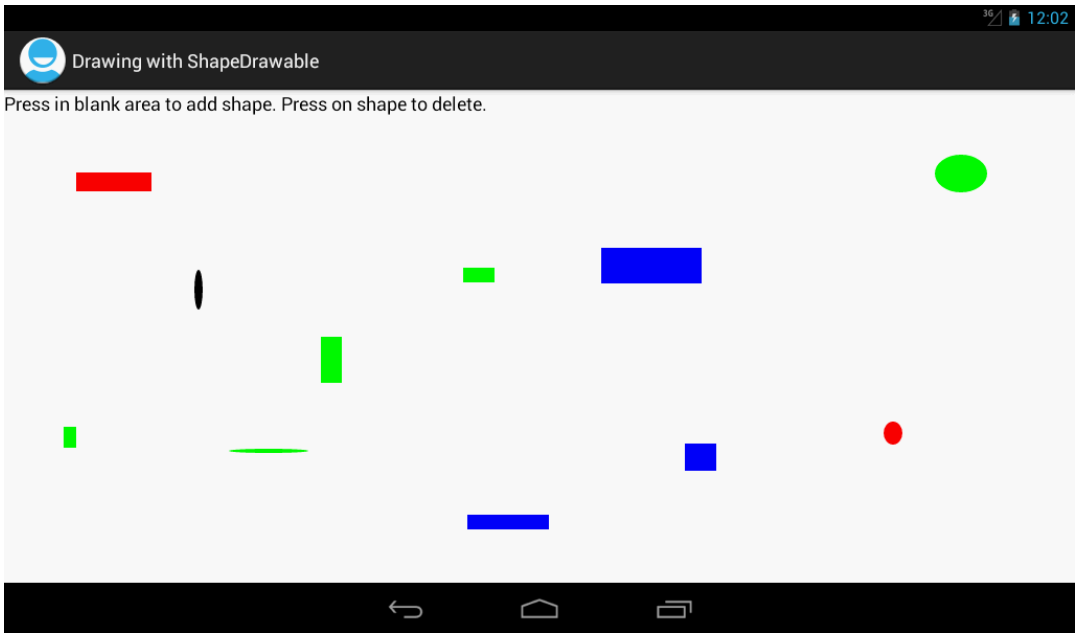

Layout File for Example (activity__draw_shapes2.xml)

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
   <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance=
            "?android:attr/textAppearanceMedium"
        android:text="@string/heading_activity_draw_shapes2" />
    <view
        class="com.coreservlets.drawing.ShapeDrawableView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

Activity for Example

```
public class DrawShapes2 extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_draw_shapes2);
    }
}
```


Results



Touching the screen (clicking if using emulator) on an empty area results in a new shape appearing under where you touched.
Touching an existing shape deletes it.

Wrap-Up

Summary

- **Coordinate transformations**
  - canvas.translate(…)
  - canvas.rotate(…)
  - canvas.scale(…)
  - canvas.skew(…)
- **ShapeDrawable**
  - Event handler
    - Make a ShapeDrawable based on a Shape
    - Adjust its characteristics
    - Add it to a List
    - Call invalidate
  - onDraw
    - Loop down List and draw each ShapeDrawable

Questions?