

CS378 - Mobile Computing

2D Graphics

Android Graphics

- Does not use the Java awt or swing packages
- Whole set of custom classes
- Canvas: class that holds code for various "draw" methods
- Paint: Controls the drawing. A whole host of properties. Similar to Java Graphics object
- Bitmap: the things drawn on
- Drawable: the thing to draw. (rectangles, images, lines, etc.)

Common Methods for Drawing

- Two approaches
- draw graphics or animations into a View object that is part of layout
 - define graphics that go into View
 - the simple way
- Draw graphics directly to a Canvas
 - the complex way

Simple Graphics

- Use Drawables in Views
- Create a folder res/drawable
- Add images
 - png (preferred)
 - jpg (acceptable)
 - gif (discouraged)

Name	Date	Type	Size
 Picture1.png	2/21/2012 5:44 PM	PNG Image	147 KB
 Picture2.jpg	2/21/2012 5:45 PM	JPEG Image	24 KB
 Picture3.gif	2/21/2012 5:45 PM	GIF Image	78 KB

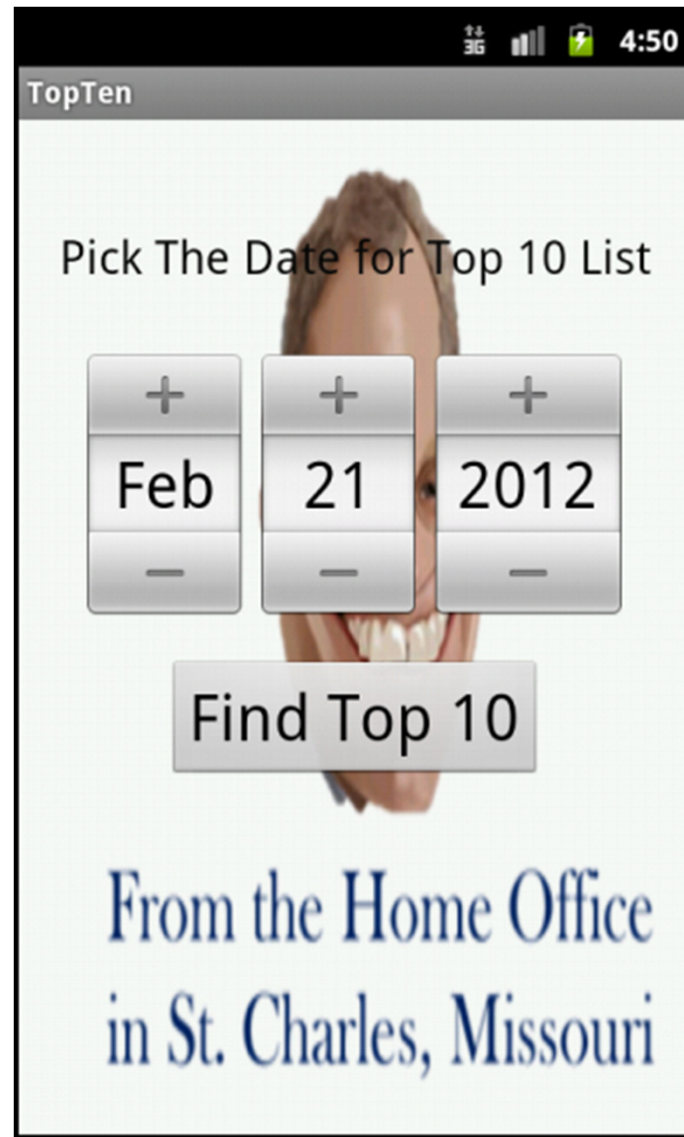
- Images can be added as background for Views

Simple Graphics

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.andro
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:orientation="vertical"
6     android:background="@drawable/home_office" >
```

- Change background to an image
 - previously used background colors

Top Ten With Image Background



Add ImageView to Layout

- In the main.xml for top ten

```
<ImageView  
    android:id="@+id/imageView1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/home_office" />
```



ImageView Attributes

- `scaleType`: how image should be moved or resized in the `ImageView`
- `tint`: affects color of image
- more to position image in `ImageView`



Changing ImageView Programmatically

- Randomly set the alpha (transparency of the image)
- Or pick an image randomly
- Or set image based on month (Season)

```
private void changeImage() {  
    ImageView iv = (ImageView) findViewById(R.id.imageView1);  
    Drawable image = getResources()  
        .getDrawable(R.drawable.home_office2);  
    image.setAlpha( (int) (Math.random() * 100) + 50);  
    iv.setImageDrawable(image);  
}
```



Using a Canvas

- Simple way -> Create a custom View and override the onDraw method
- The Canvas is sent as a parameter
- Create a class that extends View
 - override the 2 parameter constructor
 - override the onDraw method
 - perform custom drawing in the onDraw method
 - add the View to the proper layout

Simple Example - Graphics View

```
public class GraphicsView extends View {  
  
    private static final String TAG = "GraphicView";  
  
    public GraphicsView(Context context, AttributeSet attrs) {  
        super(context, attrs);  
        Log.d(TAG, "in 2 param constructor");  
    }  
  
    public GraphicsView(Context context) {  
        super(context);  
        Log.d(TAG, "in 1 param constructor");  
    }  
}
```


GraphicsView - onDraw

```
@Override
protected void onDraw(Canvas canvas) {
    Log.d(TAG, "in ondraw");
    Paint p = new Paint();
    p.setColor(Color.WHITE);
    int w = getWidth();
    int h = getHeight();
    canvas.drawRect(0, 0, w, h, p);

    p.setColor(getResources().getColor(R.color.burnt_orange));
    // Log.d(TAG, getWidth() + " " + getHeight() + " " + canvas
    int circleRadius = Math.min(w, h) / 3;
    canvas.drawCircle(w/ 2, h/ 2, circleRadius, p);
}
```


Add CustomView to XML

- in main.xml
- add custom View as last element in LinearLayout

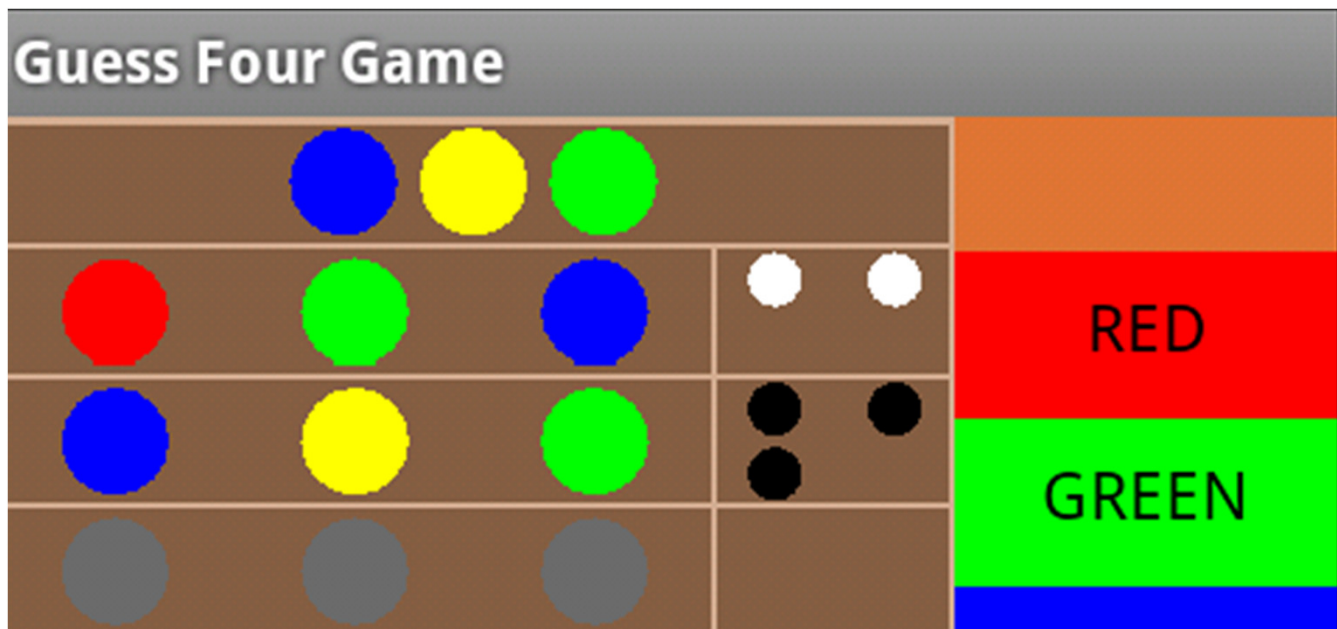
```
<scottm.examples.GraphicsView  
    android:id="@+id/graphicsView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:padding="5dp" />
```

Canvas Class

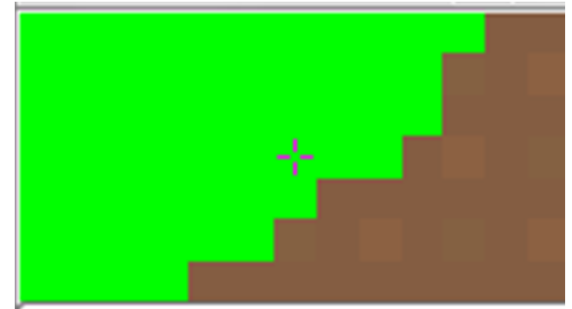
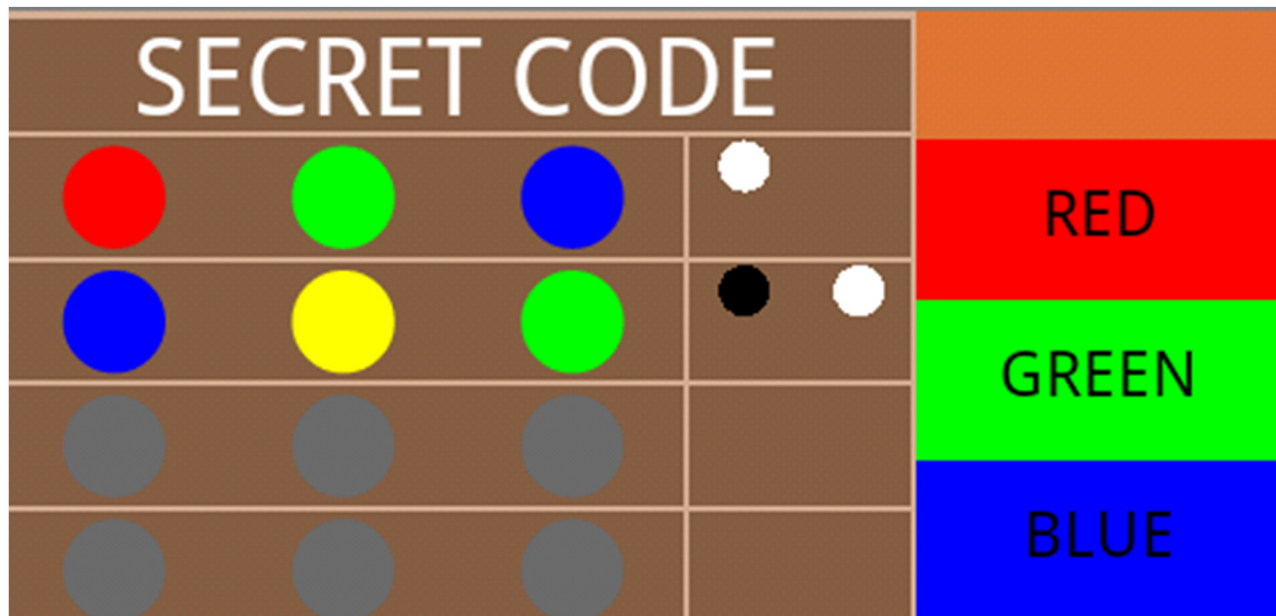
- methods to draw
 - lines
 - arcs
 - paths
 - images
 - circles
 - ovals
 - points
 - text
 - and a few I missed

Paint

- typically create Paint with anti aliasing
- `Paint p =
new Paint(Paint.ANTI_ALIAS_FLAG);`



Anti Aliasing



Paint Object

- many, many attributes and properties including:
 - current color to draw with
 - whether to fill or outline shapes
 - size of stroke when drawing
 - text attributes including size, style (e.g. underline, bold), alignment,
 - gradients

Gradients

- 3 kinds of gradients
- LinearGradient
- RadialGradient
- SweepGradient
- at least 2 color, but possibly more
- flows from one color to another

Linear Gradient

```
public LinearGradient (float x0, float y0, float x1, float y1, int color0, int color1, Shader.TileMode tile)
```

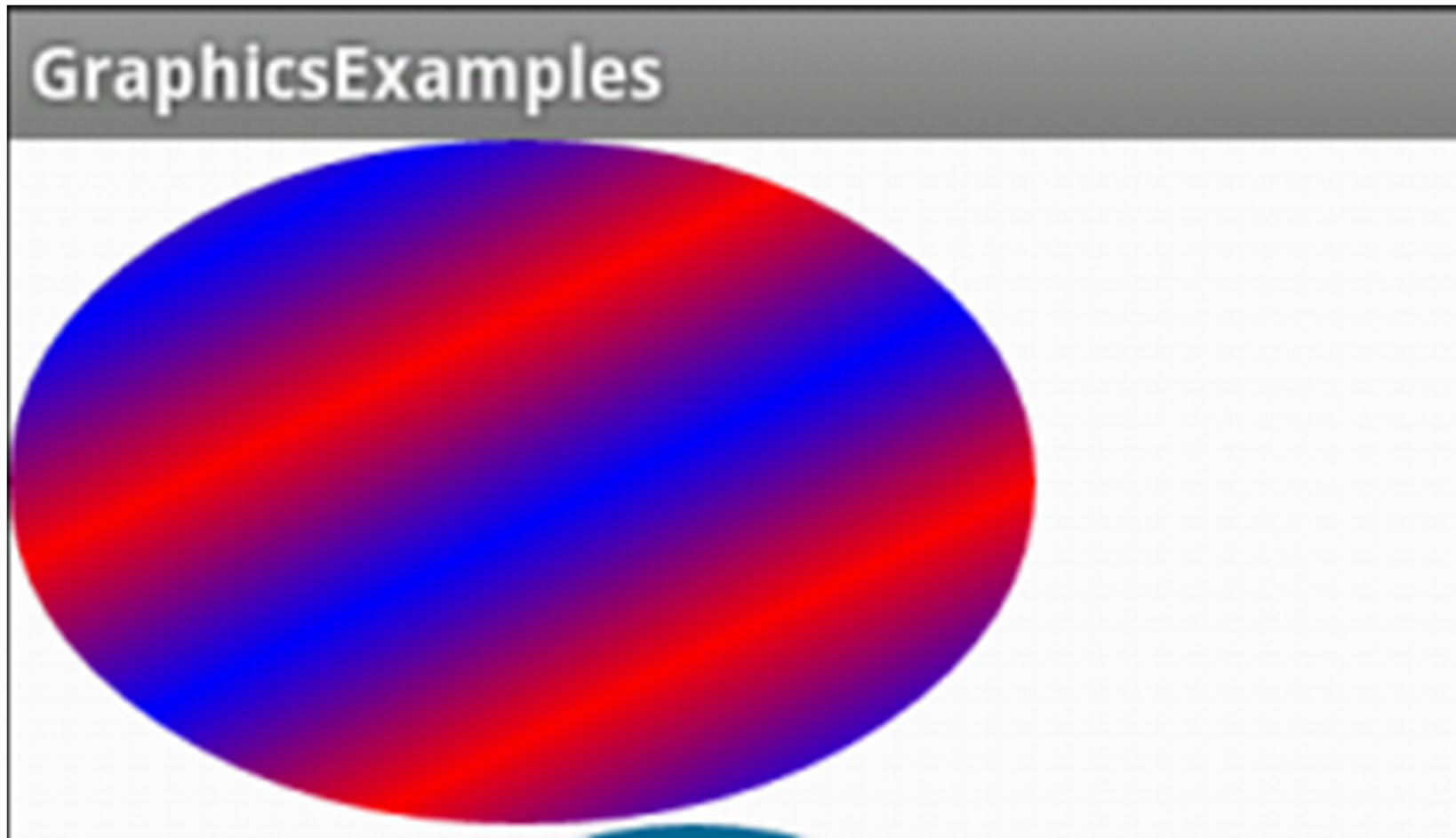
Create a shader that draws a linear gradient along a line.

Parameters

<i>x0</i>	The x-coordinate for the start of the gradient line
<i>y0</i>	The y-coordinate for the start of the gradient line
<i>x1</i>	The x-coordinate for the end of the gradient line
<i>y1</i>	The y-coordinate for the end of the gradient line
<i>color0</i>	The color at the start of the gradient line.
<i>color1</i>	The color at the end of the gradient line.
<i>tile</i>	The Shader tiling mode

```
// linear gradient
Paint p = new Paint(Paint.ANTI_ALIAS_FLAG);
LinearGradient lg = new LinearGradient(0, 0, 25, 50,
    Color.RED, Color.BLUE, Shader.TileMode.MIRROR);
p.setShader(lg);
canvas.drawOval(new RectF(0, 0, 300, 200), p);
```

LinearGradient



RadialGradient

```
public RadialGradient (float x, float y, float radius, int color0, int color1, Shader.TileMode tile)
```

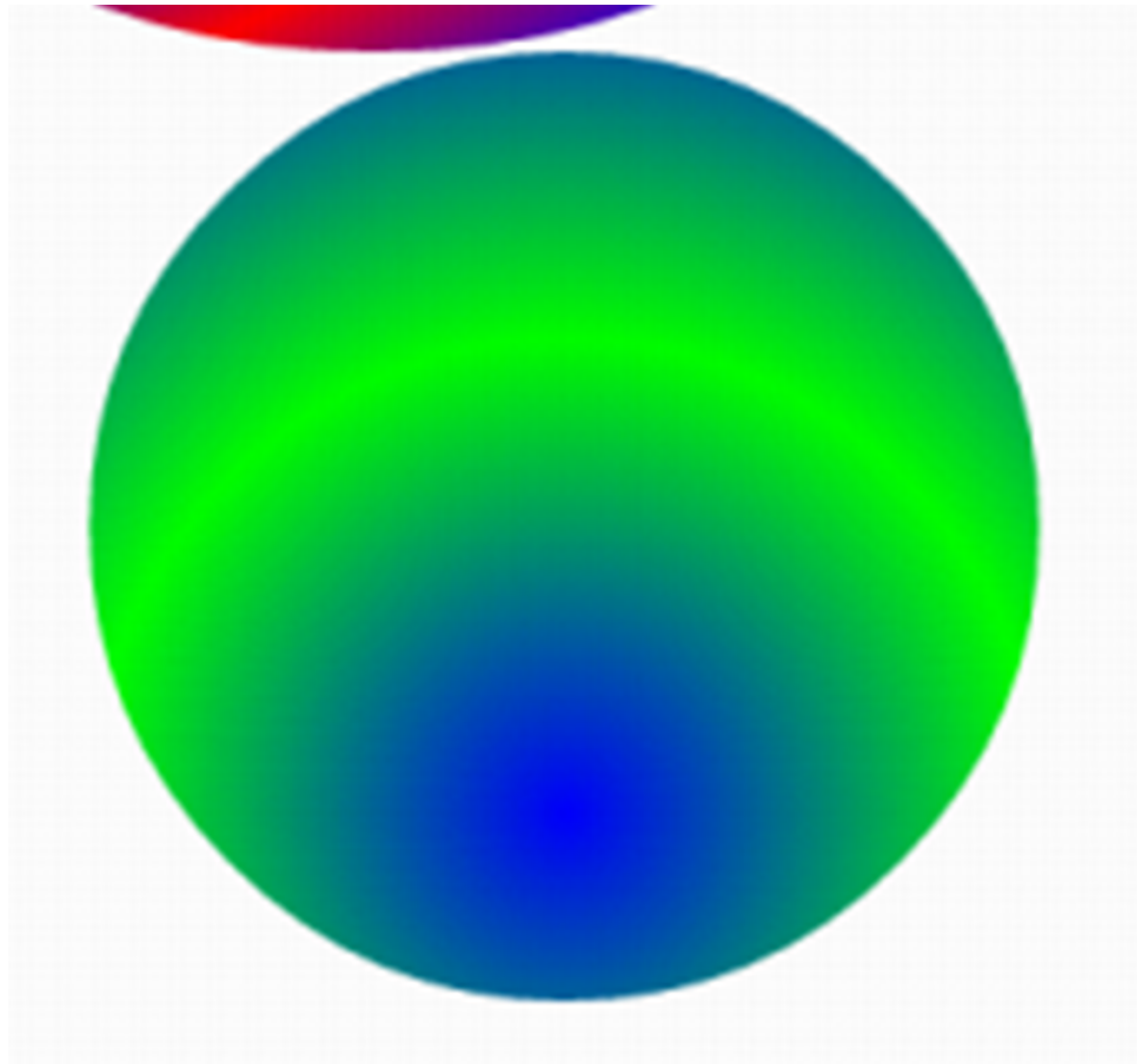
Create a shader that draws a radial gradient given the center and radius.

Parameters

<i>x</i>	The x-coordinate of the center of the radius
<i>y</i>	The y-coordinate of the center of the radius
<i>radius</i>	Must be positive. The radius of the circle for this gradient
<i>color0</i>	The color at the center of the circle.
<i>color1</i>	The color at the edge of the circle.
<i>tile</i>	The Shader tiling mode

```
// radial gradient  
RadialGradient rg = new RadialGradient(200, 400, 125,  
    Color.BLUE, Color.GREEN, Shader.TileMode.MIRROR);  
p.setShader(rg);  
canvas.drawCircle(200, 325, 125, p);
```

RadialGradient



SweepGradient

```
public SweepGradient (float cx, float cy, int[] colors, float[] positions) Since: API Level 1
```

A subclass of Shader that draws a sweep gradient around a center point.

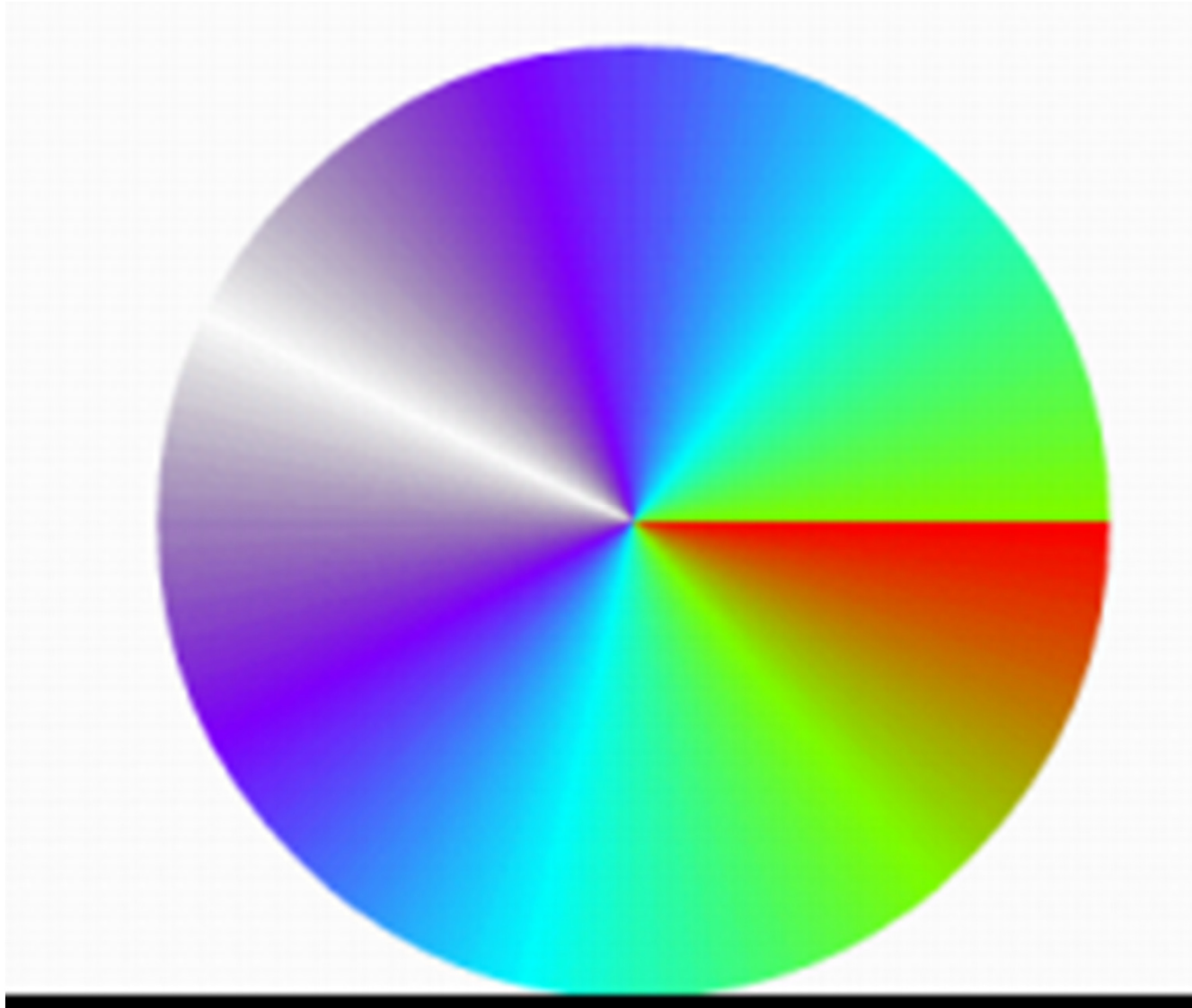
Parameters

<i>cx</i>	The x-coordinate of the center
<i>cy</i>	The y-coordinate of the center
<i>colors</i>	The colors to be distributed between around the center. There must be at least 2 colors in the array.
<i>positions</i>	May be NULL. The relative position of each corresponding color in the colors array, beginning with 0 and ending with 1.0. If the values are not monotonic, the drawing may produce unexpected results. If positions is NULL, then the colors are automatically spaced evenly.

SweepGradient

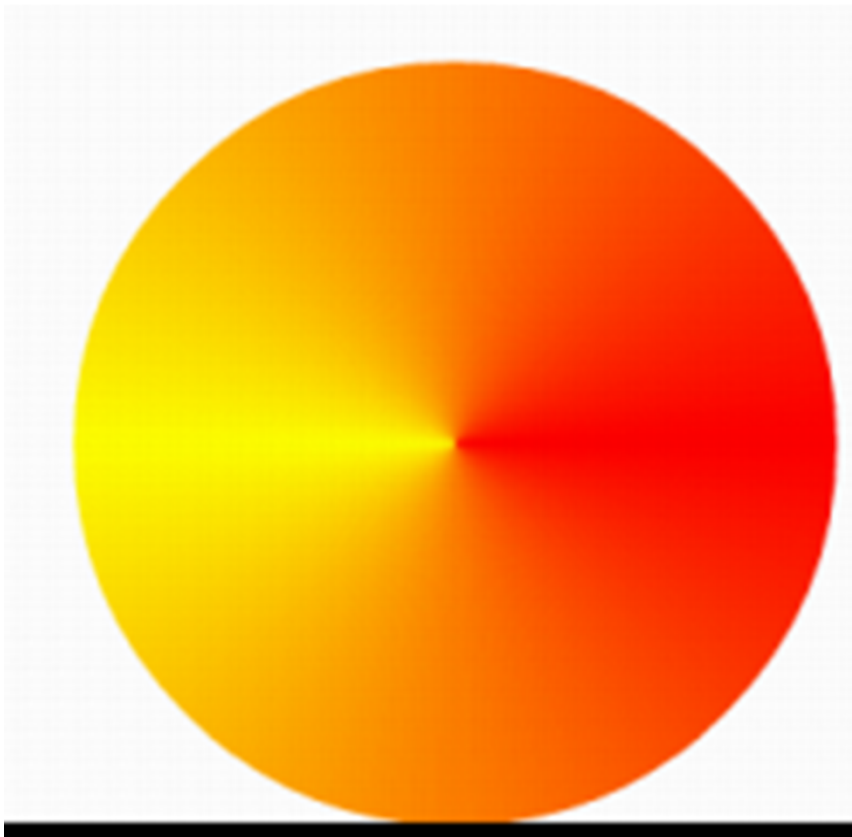
```
// sweep gradient
int numColors = 4;
int angleIncrement = 360 / numColors;
int[] rainbow = new int[numColors * 2];
float[] hsv = {0, 1, 1};
for(int i = 0; i < rainbow.length / 2; i++) {
    rainbow[i] = Color.HSVToColor(hsv);
    hsv[0] += angleIncrement;
}
for(int i = rainbow.length / 2; i < rainbow.length; i++) {
    rainbow[i] = rainbow[rainbow.length - i];
}
SweepGradient sg = new SweepGradient(300, 600, rainbow, null);
p.setShader(sg);
canvas.drawCircle(300, 600, 125, p);
```

SweepGradient



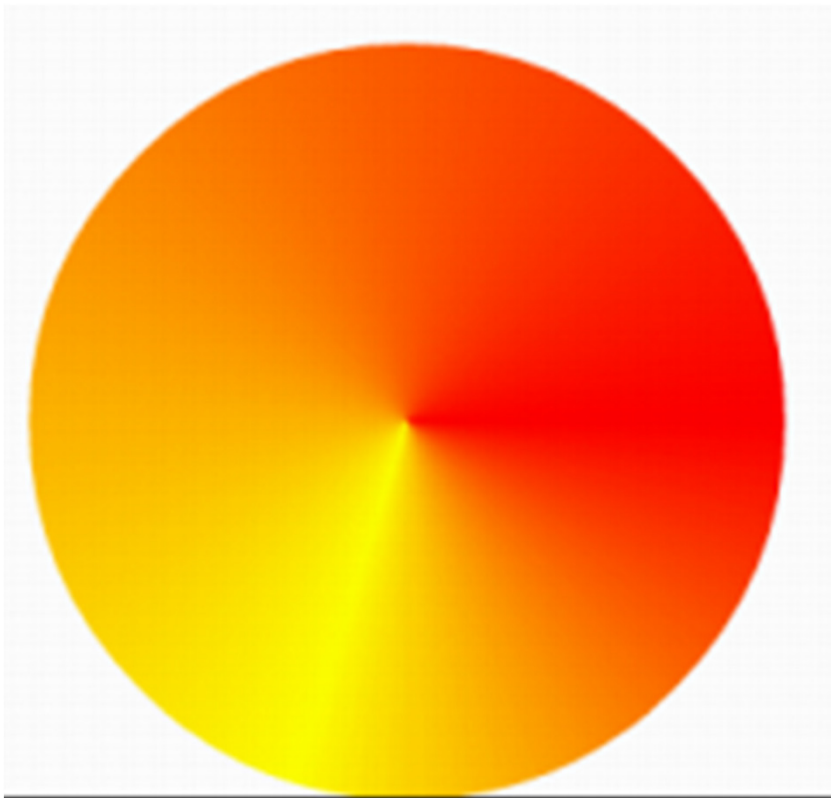
SweepGradient

```
SweepGradient sg = new SweepGradient(300, 600,  
    new int[] {Color.RED, Color.YELLOW, Color.RED},  
    null);  
p.setShader(sg);  
canvas.drawCircle(300, 600, 125, p);
```



SweepGradient

```
SweepGradient sg = new SweepGradient(300, 600,  
    new int[] {Color.RED, Color.YELLOW, Color.RED},  
    new float[]{0, 0.3f, 1});  
p.setShader(sg);  
canvas.drawCircle(300, 600, 125, p);  
|
```



GuessFour

Simple Animations

- Tweened Animations
- provide a way to perform simple animations on Views, Bitmaps, TextViews, Drawables
- provide start point, end point, size, rotation, transparency, other properties
- Can set up tweened animation in XML or programmatically

GuessFour Example

- On error board shakes back and forth
- On win board shakes up and down
- From BoardView in GuessFour

```
public void shakeLeftRight() {  
    Log.d(TAG, "in shake! Trying to start animation!");  
    startAnimation(AnimationUtils.loadAnimation(game, R.anim.shake));  
}  
  
public void shakeUpDown() {  
    Log.d(TAG, "in shake! Trying to start animation!");  
    startAnimation(AnimationUtils.loadAnimation(game, R.anim.shake_up_down));  
}
```

res/anim

- shake up down

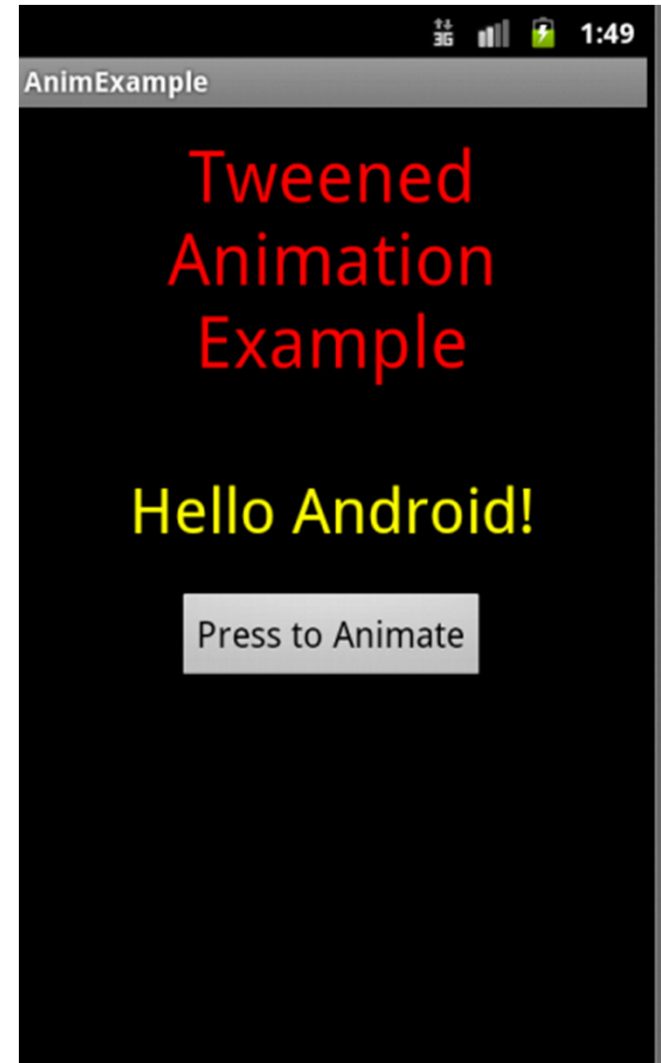
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <translate
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     android:fromYDelta="0"
5     android:toYDelta="25"
6     android:duration="2000"
7     android:interpolator="@anim/cycle_7" />
8
```

- shake left right

```
<?xml version="1.0" encoding="utf-8"?>
<translate
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromXDelta="0"
    android:toXDelta="25"
    android:duration="1000"
    android:interpolator="@anim/cycle_7" />
```

More Tweened Examples

- hyperspace example from android dev site
- rotate and change alpha
- animation types:
 - alpha
 - scale
 - translate
 - rotate



<http://developer.android.com/guide/topics/resources/animation-resource.html>

More Complex Graphics

- Don't want apps to become unresponsive
- If complex graphics or animation use SurfaceView class
- Main view not waiting on onDraw to finish
- secondary thread with reference to SurfaceView
- SurfaceView draws and when done display result

Using a SurfaceView

- extend SurfaceView
- implement SurfaceHolder.Callback
 - methods to notify main View when SurfaceView is created, changed or destroyed

Simple Example

- Static Screen
- continuously draw several hundred small rectangles (points, with stroke = 10)
 - slowly fill screen and then keep changing

```
public class StaticView extends SurfaceView
    implements SurfaceHolder.Callback {

    private static final String TAG = "Static";

    private StaticThread thread;

    public StaticView(Context context, AttributeSet attrs) {
        super(context, attrs);

        // register our interest in hearing about changes to
        SurfaceHolder holder = getHolder();
        holder.addCallback(this);
    }
}
```



Implement SurfaceHolder.Callback methods

```
// called when surface changes size
@Override
public void surfaceChanged(SurfaceHolder holder, int format,
    int width, int height) {

}
```

```
// called when surface is first created
@Override
public void surfaceCreated(SurfaceHolder holder)
{
    thread = new StaticThread(holder);
    thread.setRunning(true);
    thread.start(); //start the animation
}
```


Prevent Runaway Threads!

```
// called when the surface is destroyed
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    // ensure that thread terminates properly
    boolean retry = true;
    thread.setRunning(false);

    while (retry) {
        try {
            thread.join();
            Log.d(TAG, "Thread stopped! " + thread);
            retry = false;
        }
        catch (InterruptedException e) {}
    }
}
```

Inner Class for Thread

```
private class StaticThread extends Thread {

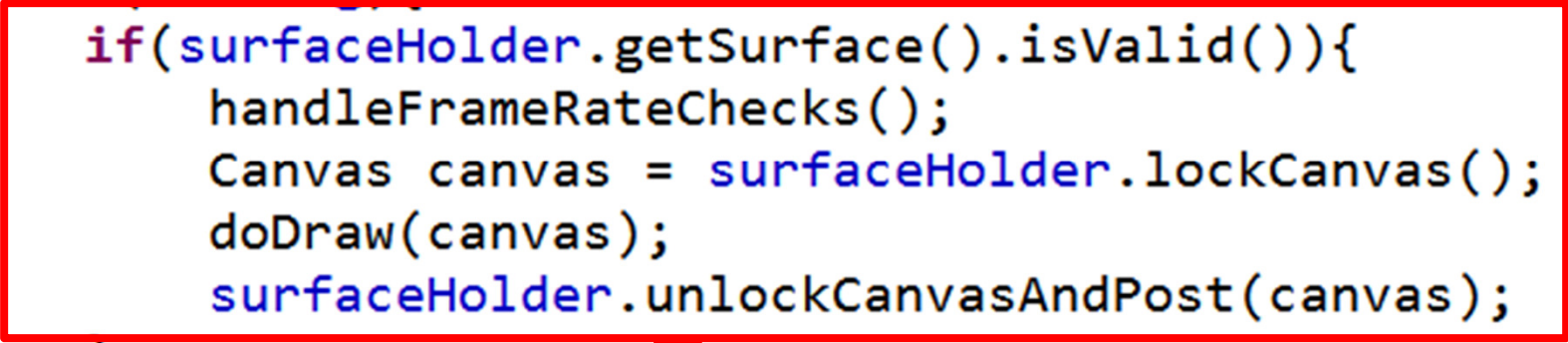
    private boolean running;
    private SurfaceHolder surfaceHolder;
    private Bitmap image;
    private Random random;
    private Paint paint;

    public StaticThread(SurfaceHolder sh) {
        surfaceHolder = sh;
        random = new Random();
        paint = new Paint();
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeWidth(10);
        image = Bitmap.createBitmap(getWidth(), getHeight(),
            Bitmap.Config.ARGB_8888);
        Log.d(TAG, "width: " + image.getWidth());
        Log.d(TAG, "height: " + image.getHeight());
        Log.d(TAG, "image: " + image);
    }

    public void setRunning(boolean status) {
        running = status;
    }
}
```

Run Method in StaticThread

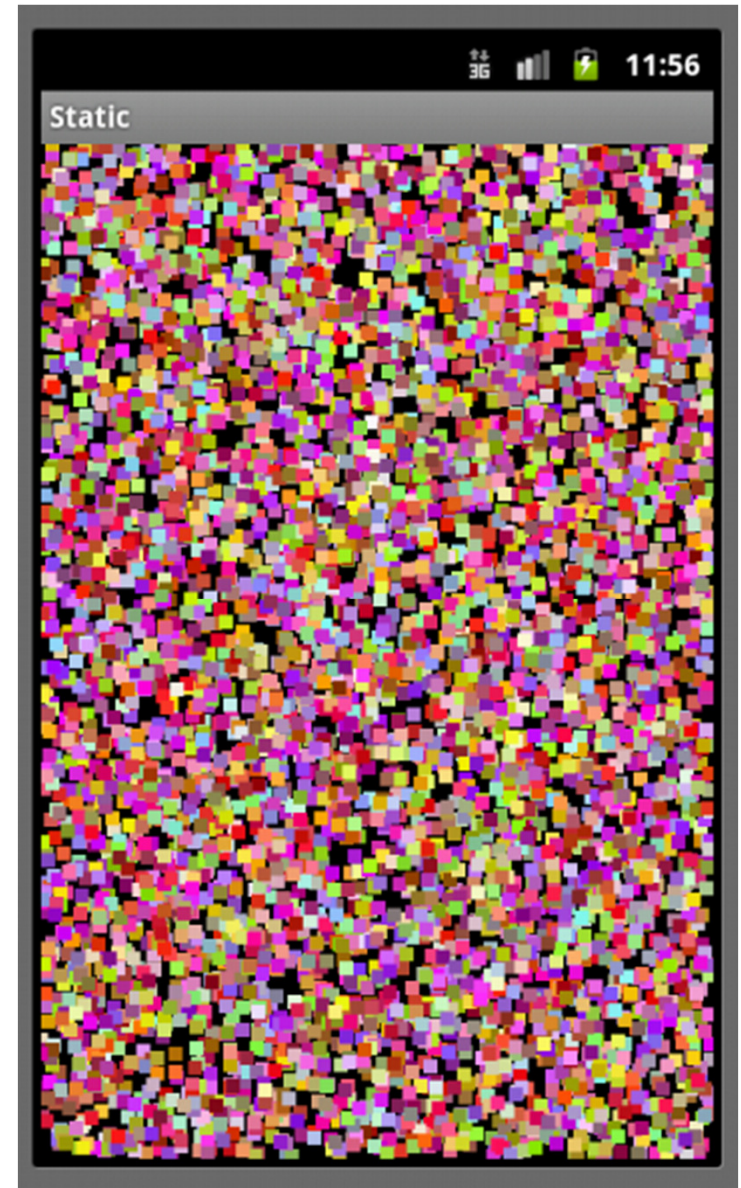
```
@Override
public void run() {
    while(running){
        if(surfaceHolder.getSurface().isValid()){
            handleFrameRateChecks();
            Canvas canvas = surfaceHolder.lockCanvas();
            doDraw(canvas);
            surfaceHolder.unlockCanvasAndPost(canvas);
        }
    }
}
```



Standard Approach for
Drawing on SurfaceView

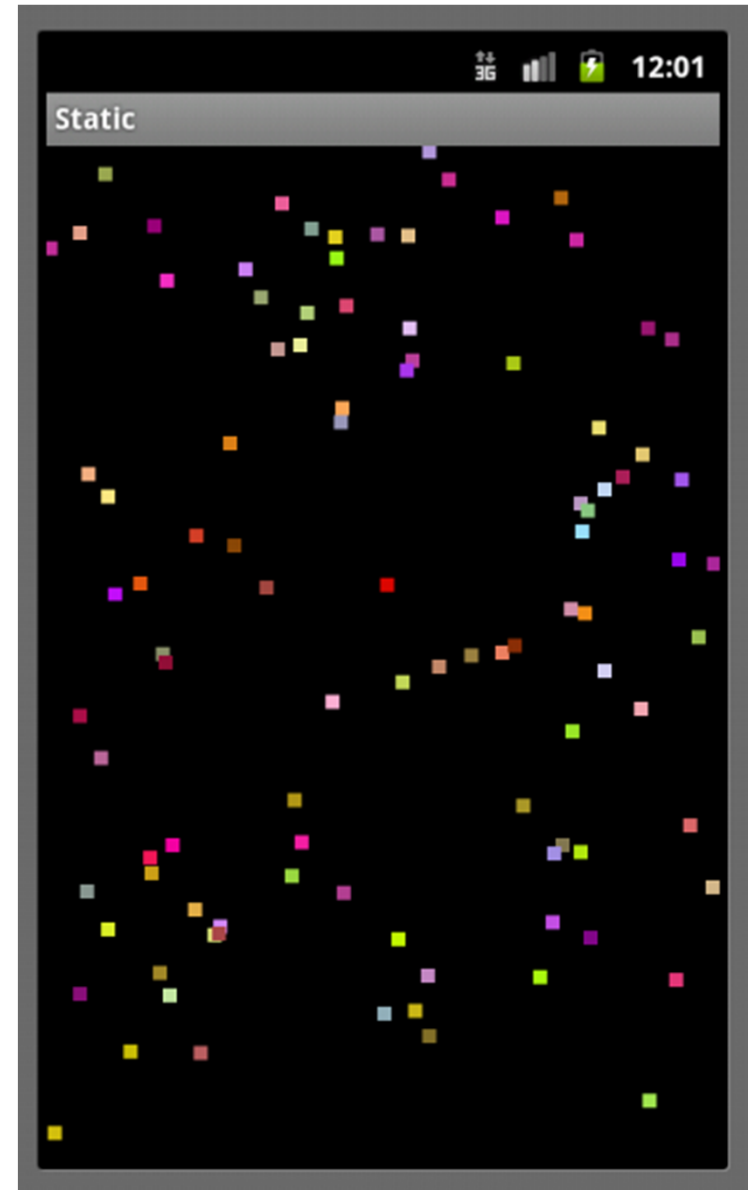
Demo run()

- Pronounced flicker and jitter
- Double buffer under the hood
- We are drawing on two different Bitmaps
- Canvas does drawing onto Bitmap



Remove Flicker / Jitter

- If we draw background each "frame" then we don't redraw previous rectangles
- How about "saving" all the data?
 - points, colors



Alternative

- Recall two approaches:
 - draw on UI thread by overriding onDraw
 - create custom View (tutorial 4)
 - okay if not a lot of drawing
 - must keep UI thread responsive
 - complex drawing or animations using SurfaceView
- Third approach, possible variation on the above two approaches
 - maintain a separate Bitmap

Separate Bitmap

- StaticThread has a Bitmap instance var
- Initialize in constructor

```
public StaticThread(SurfaceHolder sh) {  
    surfaceHolder = sh;  
    random = new Random();  
    paint = new Paint();  
    paint.setStyle(Paint.Style.STROKE);  
    paint.setStrokeWidth(10);  
    |  
    image = Bitmap.createBitmap(getWidth(), getHeight(),  
                                Bitmap.Config.ARGB_8888);  
}
```

Updates to Bitmap

```
@Override
public void run() {
    while(running){
        if(surfaceHolder.getSurface().isValid()){
            handleFrameRateChecks();

            Canvas c = new Canvas(image);
            doDraw(c);

            Canvas canvas = surfaceHolder.lockCanvas();
            canvas.drawBitmap(image, new Matrix(), pt);
            surfaceHolder.unlockCanvasAndPost(canvas);

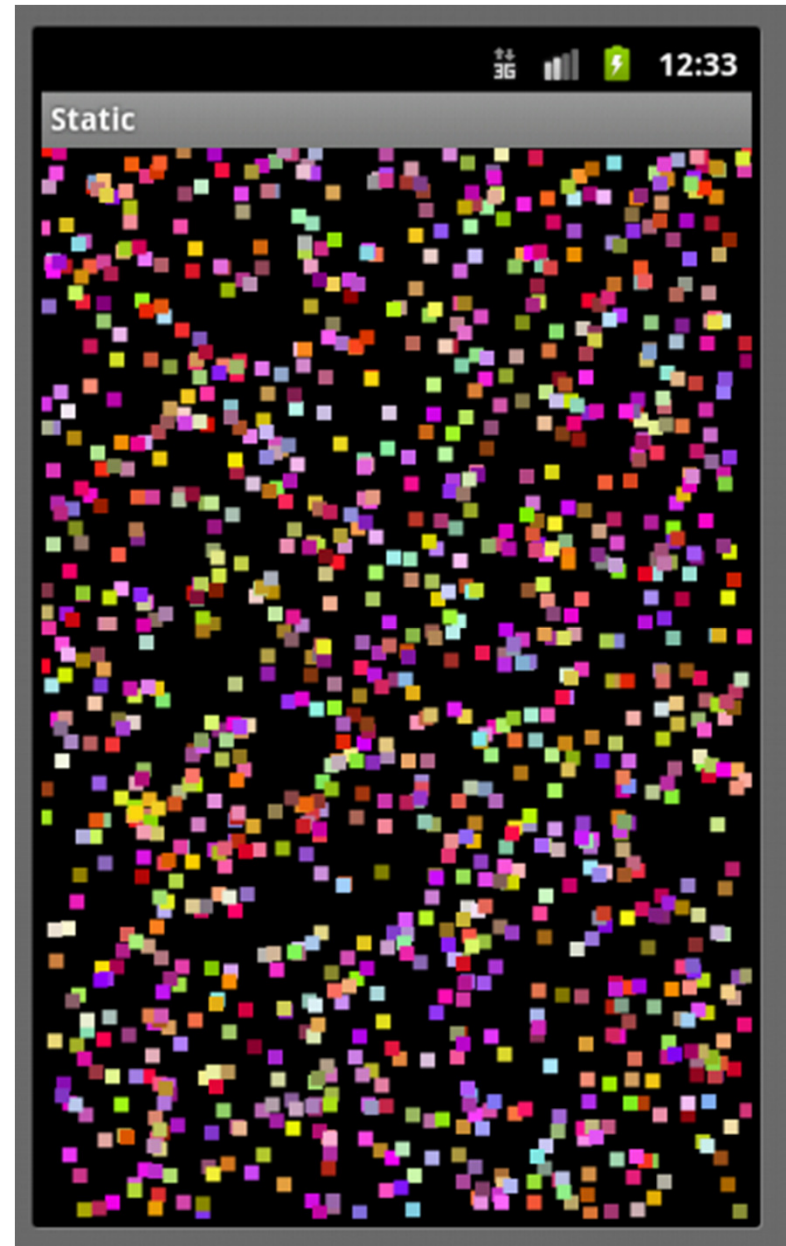
            try{
                Thread.sleep(1000);
            }
            catch(Exception e) {}
        }
    }
}
```

Create a Canvas to draw on the Bitmap we are saving

When done drawing to Bitmap use SurfaceView Canvas to draw

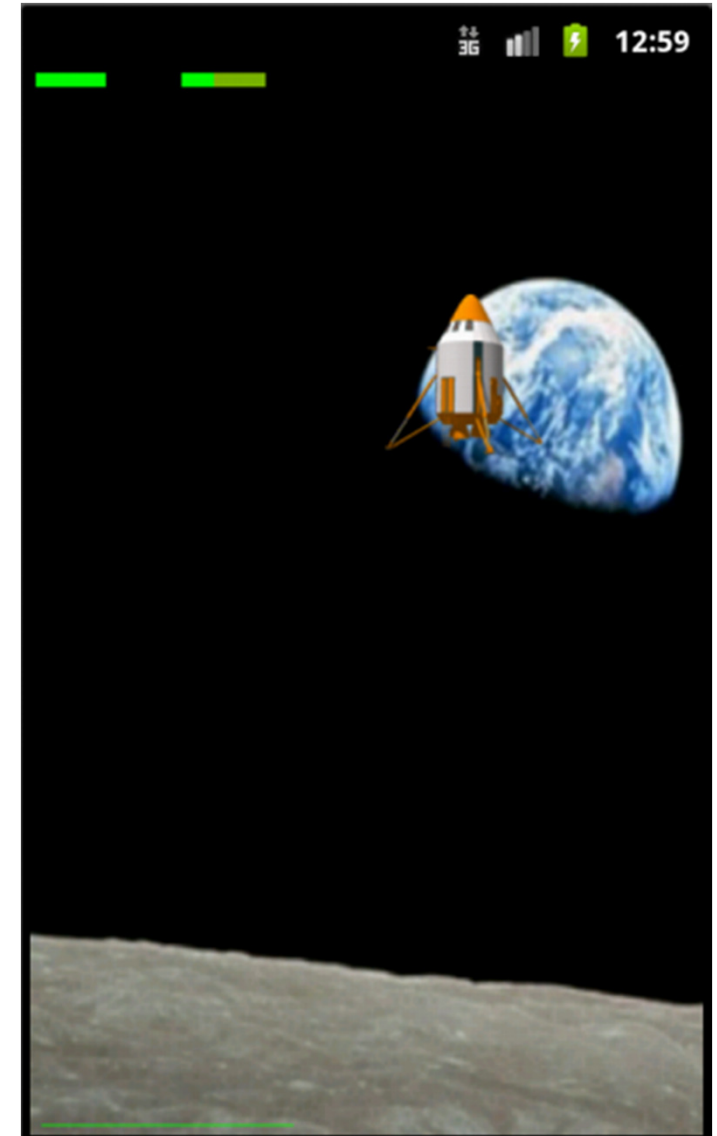
Demo Alt Version of run()

- Flicker and jitter?
- Also possible to save Bitmap to file for later use



Animations

- Frame based vs. Time based
- Frame based:
 - update every frame
 - simple, but difference in frame rates
- Time based
 - update every frame but based on time elapsed since last frame
 - more work, more accurate
 - sdk example lunar lander



Checking Frame Rate

- From StaticView
- Emulator 6-7 fps, dev phone 40 -45 fps

```
private void handleFrameRateChecks() {  
    long currTime = System.currentTimeMillis();  
    long diff = currTime - prevTime;  
    prevTime = currTime;  
    // Log.d("Static", "time diff: " + diff);  
  
    if(frameCount < 30) {  
        frameCount++;  
    }  
    else {  
        frameCount = 0;  
        long timeDiff = currTime - startTime;  
        startTime = currTime;  
        double frameRate = 1000.0 / (timeDiff / 30.0) ;  
        Log.d("Static", "frame rate: " + (int) frameRate);  
        Log.d("Static", "timediff: " + timeDiff);  
    }  
}
```

Controlling Frame Rate

- Sleep after completing work in loop of run
- More complex than shown, use previous time and current time

```
try{  
    Thread.sleep(1000);  
}  
catch(Exception e) {}
```