http://www.android.com/

Model–view–controller (MVC)

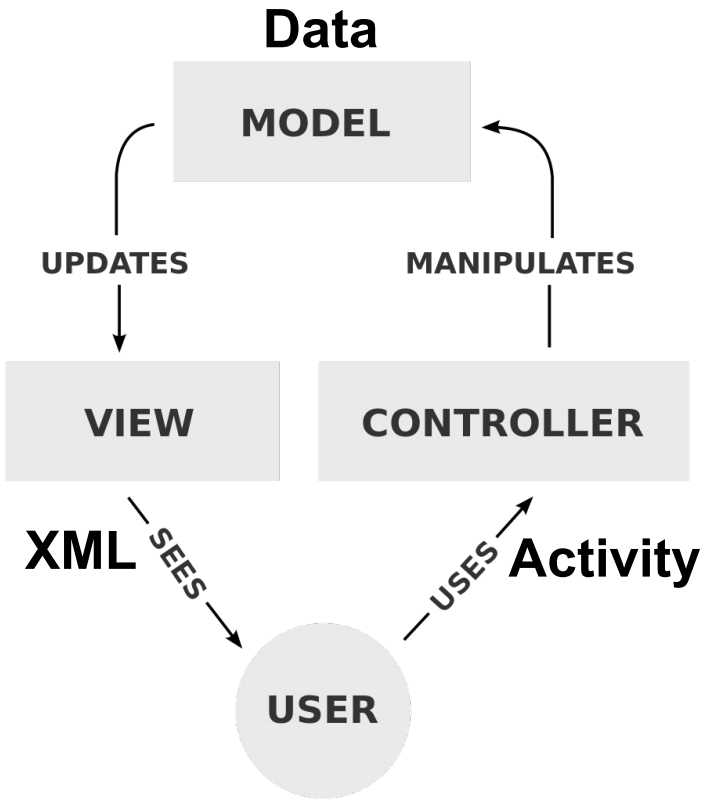
Animation and Graphics

Drawables

MVC in Android

- Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces
  - It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user
- To follow the MVC design pattern could not only boost development time, but improve the maintainability, extensibility and performance of the application
- Resources
  - http://teamtreehouse.com/library/build-a-blog-reader-android-app/exploring-the-masterdetail-template/the-modelviewcontroller-mvc-design-pattern-2
  - Paper: http://www.thinkmind.org/index.php?view=article&articleid=patterns_2013_1_20_70039

http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller



2D and 3D Graphics

- When writing an application, it's important to consider exactly what your graphical demands will be
  - Graphics and animations for a rather static application should be implemented much differently than for an interactive game
- Canvas and Drawables
  - You can do your own custom 2D rendering using the various drawing methods contained in the Canvas class or create Drawable objects for things such as textured buttons or frame-by-frame animations
- Hardware Acceleration
  - Beginning in Android 3.0, you can hardware accelerate the majority of the drawing done by the Canvas APIs to further increase their performance.
- OpenGL
  - Android supports OpenGL ES 1.0, 2.0 and 3.0, with Android framework APIs as well as natively with the Native Development Kit (NDK)
  - Using the framework APIs is desireable when you want to add a few graphical enhancements to your application that are not supported with the Canvas APIs, or if you desire platform independence and don't demand high performance. There is a small performance hit in using the framework APIs compared to using the NDK.

Animation

- The Android framework provides two animation systems
  - Property Animation (android.animation)
    - Introduced in Android 3.0 (API level 11), the property animation system lets you animate properties of any object, including ones that are not rendered to the screen. The system is extensible and lets you animate properties of custom types as well.
  - View Animation (android.view.animation)
    - View Animation is the older system and can **only** be used for Views with some limitations. It is relatively easy to setup and offers enough capabilities to meet many application's needs.
- Drawable Animation
  - Drawable animation involves displaying Drawable resources one after another, like a roll of film. This method of animation is useful if you want to animate things that are easier to represent with Drawable resources, such as a progression of bitmaps.

Drawables

- The **android.graphics.drawable** package is where you'll find the common classes used for drawing shapes and images in two-dimensions.
- A Drawable is a general abstraction for "something that can be drawn."
- Supported file types are PNG (preferred), JPG (acceptable) and GIF (discouraged)
- There are three ways to define and instantiate a Drawable
  - using an image saved in your project resources
  - using an XML file that defines the Drawable properties
  - or using the normal class constructors

```
ImageView iv = new ImageView(this); // Define and instantiate an ImageView
iv.setImageResource(R.drawable.my_image);
// or create a Drawable from resources
Drawable myImage = Context.getResources().getDrawable(R.drawable.my_image);
```

```
<ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/my_image"/>
```

Drawable & ShapeDrawable

- Shape drawable defined in res/drawable/ground.xml
- Shape in xml used in layout.xml file
- Draw a shape programmatically
- Draw shape from a XML layout, then the CustomDrawable class must override the View(Context, AttributeSet) constructor
- Set the view with SetContentView(mCustomDrawableView / layoutfile.xml);

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:dither="true"
    android:shape="rectangle" >
    <solid android:color="#339933" />
</shape>
```

```
<ImageView
    android:id="@+id/ground"
    android:layout_width="fill_parent"
    android:layout_height="200dp"
    android:layout_alignParentBottom="true"
    android:contentDescription="@string/ground"
    android:padding="40dp"
    android:src="@drawable/ground" />
```

```
// layout from xml pointing to custom class
<com.example.shapedrawable.CustomDrawableView
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

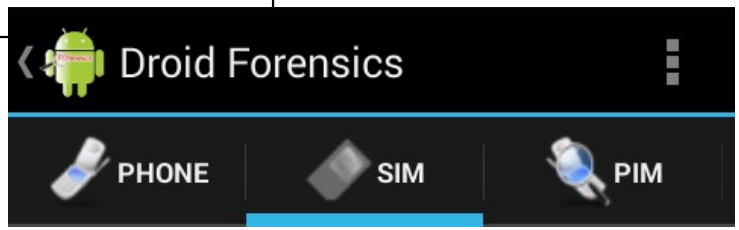
```
public class CustomDrawableView extends View {
   private ShapeDrawable mDrawable;

   public CustomDrawableView(Context context, AttributeSet attrs) {
      super(context, attrs);
      int x = 10, y = 10,  width = 300, height = 50;

      mDrawable = new ShapeDrawable(new OvalShape());
      mDrawable.getPaint().setColor(0xff74AC23);
      mDrawable.setBounds(x, y, x + width, y + height);
   }

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

State List Drawable

- A StateListDrawable is a drawable object defined in a XML file which uses several different images to represent the same graphic, depending on the state of the object (selected, pressed, focused, etc.)
- Each graphic is represented by an <item> element (describing the state) inside a single <selector> element
- During each state change, the state list is traversed top to bottom and the first item that matches the current state is used
- Example: XML file saved at res/drawable/button.xml and a new background is applied to the button depending on state

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/button_pressed" /> <!-- pressed -->
    <item android:state_focused="true"
          android:drawable="@drawable/button_focused" /> <!-- focused -->
    <item android:state_hovered="true"
          android:drawable="@drawable/button_focused" /> <!-- hovered -->
    <item android:drawable="@drawable/button_normal" /> <!-- default -->
</selector>
```

```
<Button
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:background="@drawable/button" />
```



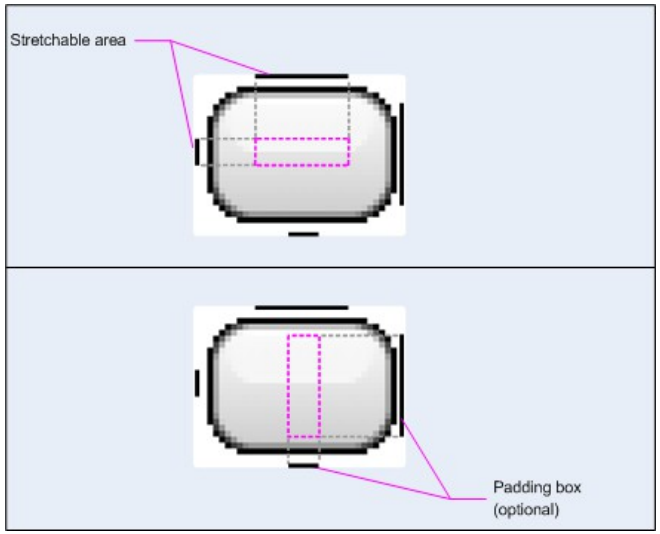
Custom color buttons

- There are three approaches
  - Set background color (bad) - button does not have rounded edges and does not support changes of the color to indicate focus or pressed state
  - Custom buttons via XML (not very good)
    - http://stackoverflow.com/questions/1521640/standard-android-button-with-a-different-color
  - Use different 9-patch images for the different colors (the right(eous) one)
- Procedure
  - Create images 9-patch images for the different colors that you need and put them into drawable-hdpi, drawable-mdpi, etc. (you will need several versions if you want the buttons to look good on different devices)
  - Create XML files with <selector> for each color (drawable/custom_button_*.xml)

http://ogrelab.ikratko.com/custom-color-buttons-for-android/

Nine-patch stretchable bitmap

- A NinePatchDrawable graphic is a stretchable bitmap image, which Android will automatically resize to accommodate the contents of the View in which you have placed it as the background
- The Draw 9-patch tool (SDK /tools directory) allows you to easily create a NinePatch graphic using a WYSIWYG editor
  - http://developer.android.com/guide/topics/graphics/2d-graphics.html#nine-patch
- From a terminal, launch the draw9patch application from your SDK /tools directory




```
<Button id="@+id/tiny"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerInParent="true"
      android:text="Tiny"
      android:textSize="8sp"
      android:background="@drawable/my_button_background"/>

<Button id="@+id/big"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:layout_centerInParent="true"
      android:text="Biiiiiiig text!"
      android:textSize="30sp"
      android:background="@drawable/my_button_background"/>
```

Drawable (Frame) Animation

- The android.graphics.drawable.AnimationDrawable class is the basis for Drawable animations
- A single XML file that lists the frames that compose the animation. The XML file is located in the res/drawable/ directory
- The XML file consists of an <animation-list> element as the root node and a series of child <item> nodes that each define a frame: a drawable resource for the frame and the frame duration
- This animation runs for just three frames. If it is set false then the animation will loop

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">
    <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3" android:duration="200" />
</animation-list>
```

```
// It's important to note that the start() method called on the AnimationDrawable cannot be
// called during the onCreate() method of your Activity, because the AnimationDrawable is
// not yet fully attached to the window. If you want to play the animation immediately, without
// requiring interaction, then you might want to call it from the onWindowFocusChanged() method
// in your Activity, which will get called when Android brings your window into focus.
ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
rocketImage.setBackgroundResource(R.drawable.rocket_thrust); // rocket_thrust.xml
rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
rocketAnimation.start();
```

View (Tween) Animation 1

- A disadvantage of the view animation system is that it only modify where the View was drawn, and not the actual View itself.
  - If you animated a button to move across the screen, the button draws correctly, but the actual location where you can click the button does not change!
- A tween animation calculates the animation with information such as the start point, end point, size, rotation, etc. and can perform a series of simple transformations (position, size, rotation, and transparency) on the contents of a View object
  - So, if you have a TextView object, you can move, rotate, grow, or shrink the text
- A sequence of animation instructions defines the tween animation, defined by either XML or Android code
- The animation XML file is located in the res/anim/ directory
  - The file must have a single root element: this will be either a single <alpha>, <scale>, <translate>, <rotate>, interpolator element, or <set> element that holds groups of these elements
  - By default, all animation instructions are applied simultaneously. To make them occur sequentially, you must specify the startOffset attribute

View (Tween) Animation 2

- The following XML from the example **AnimationSimpleTween** is used to stretch, then simultaneously spin and rotate a View object.
- Screen coordinates (not used in this example) are (0,0) at the upper left hand corner, and increase as you go down and to the right.
- With hyperspace_jump.xml in the res/anim/ directory of the project, the following code will reference it and apply it to an ImageView object from the layout.
- More examples in API Demos > Views > Animation

```
ImageView droidView =
     (ImageView) findViewById(
          R.id.droidView);

Animation jumpOut =
     AnimationUtils.loadAnimation(this,
          R.anim.hyperspace_jump);

droidView.startAnimation(jumpOut);
```

```
<scale
    android:interpolator="@android:anim/accelerate_decelerate_interpolator"
    android:duration="700"
    android:fillAfter="false"
    android:fromXScale="1.0"
    android:fromYScale="1.0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:toXScale="1.4"
    android:toYScale="0.6" />
<set android:interpolator="@android:anim/decelerate_interpolator" >
    <scale
        android:duration="500"
        android:fillBefore="false"
        android:fromXScale="1.4"
        android:fromYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="100"
        android:toXScale="0.0"
        android:toYScale="0.0" />
    <rotate
        android:duration="500"
        android:fromDegrees="0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="100"
        android:toDegrees="360"
        android:toYScale="0.0" />
</set>
...
```

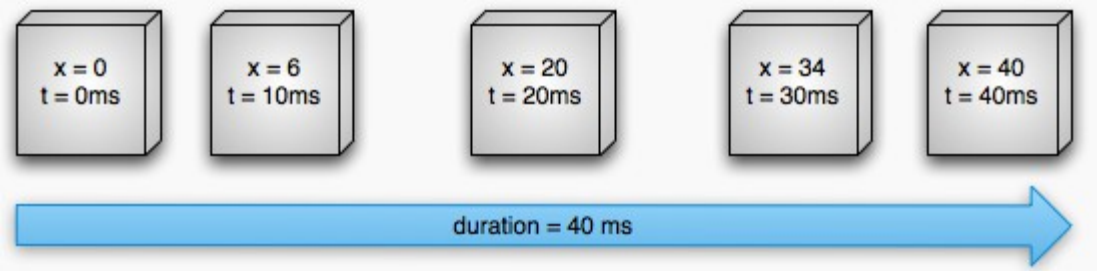
Property Animation API

- ValueAnimator
  - The main timing engine for property animation that also computes the values for the property to be animated. It has all of the core functionality that calculates animation values and contains the timing details of each animation, information about whether an animation repeats, listeners that receive update events, and the ability to set custom types to evaluate.
  - There are two pieces to animating properties: **calculating** the animated values and **setting** those values on the object and property that is being animated.
- ObjectAnimator
  - A subclass of ValueAnimator that allows you to set a target object and object property to animate. This class updates the property accordingly when it computes a new value for the animation.
- AnimatorSet
  - Provides a mechanism to group animations together so that they run in relation to one another.

Property Animation

- The property animation system lets you define the following characteristics of an animation
- **Duration:** You can specify the duration of an animation. The default length is 300 ms.
- **Time interpolation:** You can specify how the values for the property are calculated as a function of the animation's current elapsed time.
- **Repeat count and behavior:** You can specify whether or not to have an animation repeat when it reaches the end of a duration and how many times to repeat the animation. As well as play back in reverse.
- **Animator sets:** You can group animations into logical sets that play together or sequentially or after specified delays.
- **Frame refresh delay:** You can specify how often to refresh frames of your animation. The default is set to refresh every 10 ms, but it depends on the system (load and performance)
- Etc...

How Property Animation Work



- Linear and non-linear animations are supported

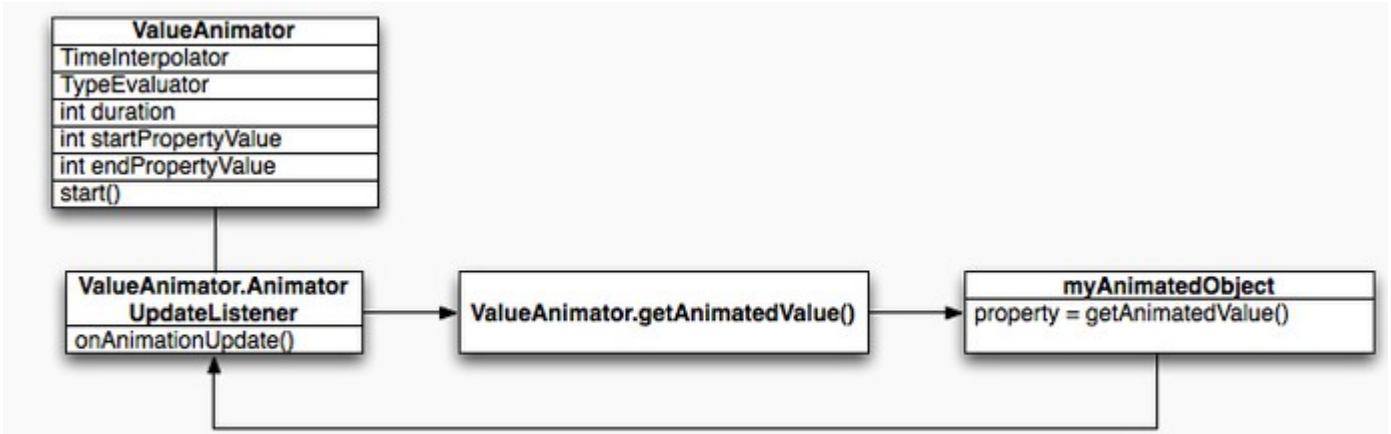


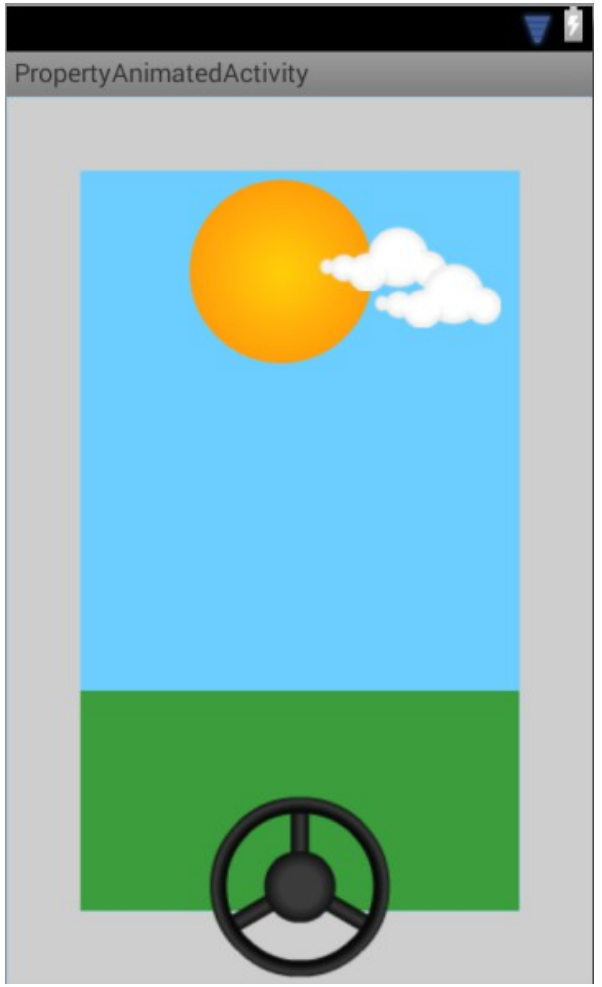
Figure 3. How animations are calculated

- The Animator class provides the basic structure for creating animations.
- The com.example.android.apis.animation package in the API Demos sample project provides many examples on how to use the property animation system.

Property Animation example 1

- PropertyAnimatedActivity example
  - http://mobile.tutsplus.com/tutorials/android/android-sdk-creating-a-simple-property-animation/
- Animation definition
  - res/animator/wheel_spin.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_decelerate_interpolator"
    android:ordering="sequentially" >

    <objectAnimator
        android:duration="3000"
        android:propertyName="rotation"
        android:repeatCount="infinite"
        android:repeatMode="reverse"
        android:valueTo="180"
        android:valueType="floatType" />
</set>
```



- Get the wheel to spin with inflated xml

```
//get the wheel view
ImageView wheel = (ImageView)findViewById(R.id.wheel);
//load the wheel spinning animation
Animator wheelSet =
      AnimatorInflater.loadAnimator(this, R.animator.wheel_spin);
//the the view as target
wheelSet.setTarget(wheel);
//start the animation
wheelSet.start();
```

Property Animation example 2

- ValueAnimator, ObjectAnimator and AnimatorSet
- car_layout is the layout id for the whole screen

```
//create a value animator to darken the sky as we move towards and away from the sun
//passing the view, property and to-from values, affects the whole layout
ValueAnimator skyAnim = ObjectAnimator.ofInt(findViewById(R.id.car_layout),
     "backgroundColor", Color.rgb(0x66, 0xcc, 0xff), Color.rgb(0x00, 0x66, 0x99));
//set same duration and animation properties as others
skyAnim.setDuration(3000);
skyAnim.setEvaluator(new ArgbEvaluator());
skyAnim.setRepeatCount(ValueAnimator.INFINITE);
skyAnim.setRepeatMode(ValueAnimator.REVERSE);

//create an object animator to move the cloud, passing the view, property and to value only
ObjectAnimator cloudAnim = ObjectAnimator.ofFloat(findViewById(R.id.cloud1), "x", -350);
//same duration and other properties as rest
cloudAnim.setDuration(3000);
cloudAnim.setRepeatCount(ValueAnimator.INFINITE);
cloudAnim.setRepeatMode(ValueAnimator.REVERSE);

//start the animations at the same time
AnimatorSet as = new AnimatorSet();
as.playTogether(skyAnim, cloudAnim, ...);
as.start();
```

Draw with a Canvas

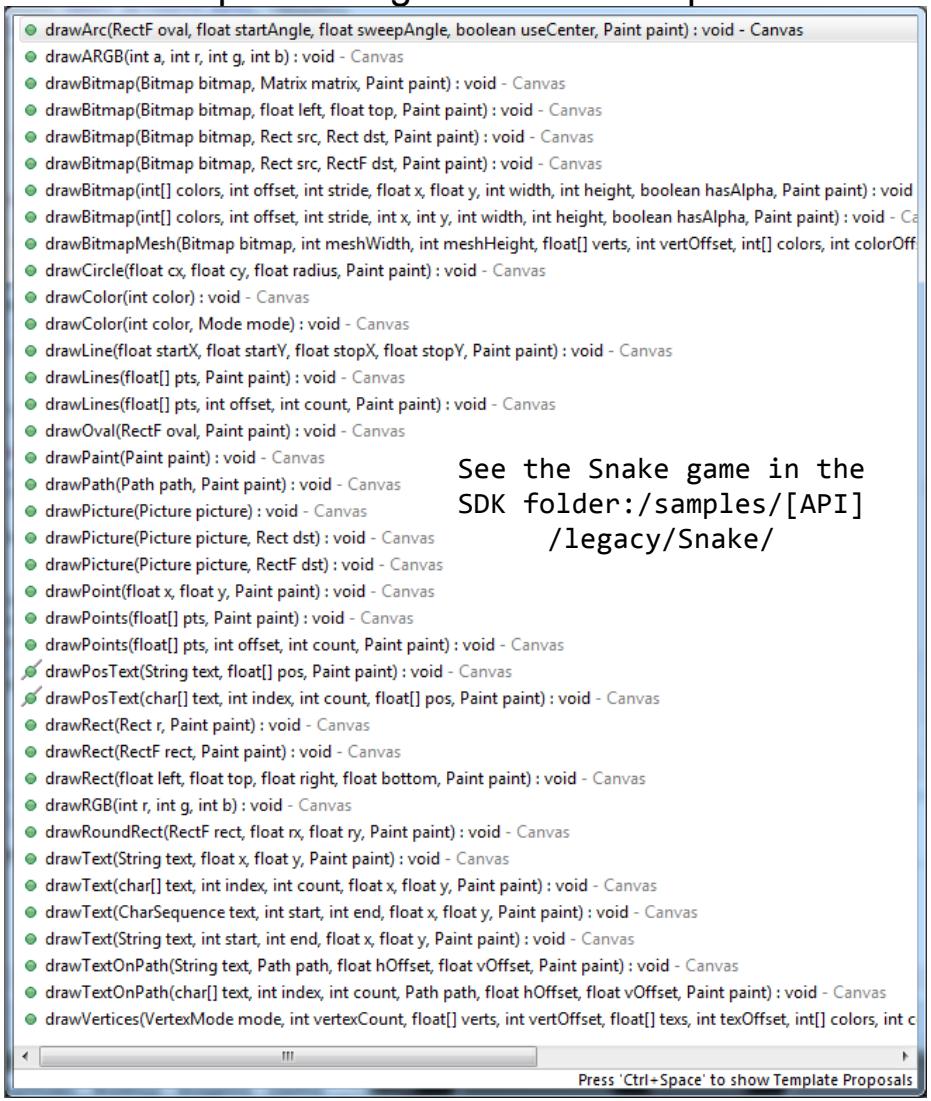
- Drawing to a Canvas is used when your application needs to regularly re-draw itself. Applications such as video games should be drawing to the Canvas on its own. There are two possibilites:
  - In the same thread as your UI Activity, wherein you create a **custom View** component in your layout, **call invalidate()** and then handle the onDraw() callback. Android framework will provide you with a pre-defined Canvas
  - Or, in a separate thread, wherein you manage a **SurfaceView** and perform draws to the Canvas as fast as your thread is capable (you do not need to request invalidate())
- A Canvas works for you as a interface, to the actual surface upon which your graphics will be drawn. If you're drawing within the onDraw() callback method, the Canvas is provided for you
- If you need to create a new Canvas, then you must define the Bitmap upon which drawing will actually be performed. The Bitmap is always required for a Canvas. You can set up a new Canvas like this:

```
// The Canvas class has its own set of drawing methods that you can use
Bitmap b1 = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);
Bitmap b2 = Bitmap.createBitmap(b1); // create a copy for offscreen use
Canvas canvas = new Canvas(b1), offScreen = new Canvas(b2);
offScreen.drawCircle(cx, cy, radius, paint); // paint describe the color and style
offScreen.drawRect(r, paint);
offScreen.drawLine(startX, startY, stopX, stopY, paint);
offScreen.drawText(text, x, y, paint);
canvas.drawBitmap(b2, left, top, paint); // copy the offscreen bitmap to canvas associated with screen
```

Canvas on a View

- A custom View component and drawing with a Canvas in View.onDraw()
- If your application does not require a significant amount of processing or frame-rate speed

```
public class AboutActivity extends Activity {
       @Override
       public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(new Animation(this));
       }

       public class Animation extends View implements Runnable   {
          Bitmap b1;
          Paint p = new Paint();
          int x=0, y=250;

          Animation(Context c){
              super(c);
              b1 = BitmapFactory.decodeResource(c.getResources(),
                     R.drawable.android_forensics);
              new Thread(this).start();
          }

          public void run(){
              try{
                 while(true){
                     x++;
                     postInvalidate(); // from other thread
                     Thread.sleep(50);
                 }
              }catch(Exception e){}
          }

          @Override
          public void onDraw(Canvas canvas){
              canvas.drawColor(Color.WHITE);
              canvas.drawBitmap(b1, x, y, p); // p can be null here
              // invalidate(); // from same thread
          }
       }
}
```



See the Snake game in the SDK folder:/samples/[API] /legacy/Snake/

Bouncing ball on a canvas

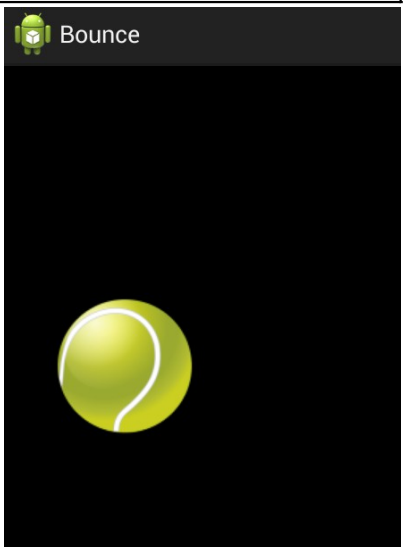
http://www.techrepublic.com/blog/android-app-builder/bouncing-a-ball-on-androids-canvas/

```
public class AnimatedView extends ImageView{
   private Context mContext;
   int x = -1, y = -1;
   private int xVelocity = 10;
   private int yVelocity = 5;
   private Handler h;
   private final int FRAME_RATE = 30;

   public AnimatedView(Context context, AttributeSet attrs) {
       super(context, attrs);
       mContext = context;
       h = new Handler();
   }
   private Runnable r = new Runnable() {
       @Override
       public void run() {
          invalidate();
       }
   };
   @Override
   protected void onDraw(Canvas c) {
       BitmapDrawable ball = (BitmapDrawable) mContext.getResources().getDrawable(R.drawable.ball);
       if (x<0 && y<0) {
          x = this.getWidth()/2;
          y = this.getHeight()/2;
       } else {
          x += xVelocity;
          y += yVelocity;
          if ((x > this.getWidth() - ball.getBitmap().getWidth()) || (x < 0)) {
              xVelocity = xVelocity*-1;
          }
          if ((y > this.getHeight() - ball.getBitmap().getHeight()) || (y < 0)) {
              yVelocity = yVelocity*-1;
          }
       }
       c.drawBitmap(ball.getBitmap(), x, y, null);
       h.postDelayed(r, FRAME_RATE);
   }
}
```

From your activity set the content to the layout xml pointing to your custom class AnimatedView

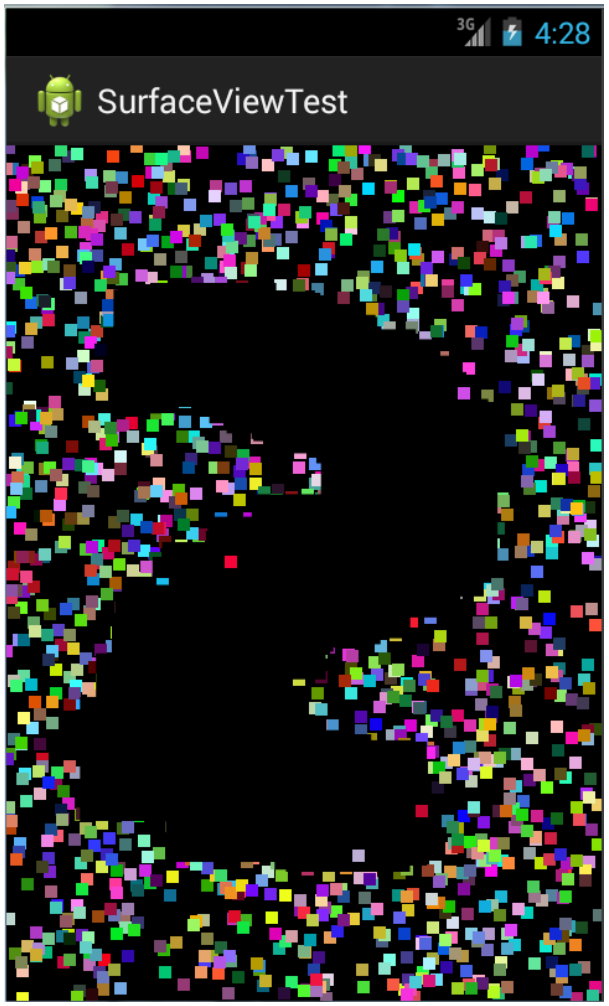
```
setContentView(R.layout.main);
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#000000"
    android:orientation="vertical" >
    <com.authorwjf.bounce.AnimatedView
        android:id="@+id/anim_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```



Canvas on a SurfaceView

- To begin, you need to create a new class that extends SurfaceView. The class should also **implement SurfaceHolder.Callback**
- SurfaceView provides a dedicated drawing surface embedded inside of a view hierarchy
- You can control the format of this surface and, if you like, its size; the SurfaceView takes care of placing the surface at the correct location on the screen
- Access to the underlying surface is provided via the SurfaceHolder interface, which can be retrieved by calling **getHolder()**
- To draw something on the SurfaceView, place the code in-between **surfaceHolder.lockCanvas()** and **surfaceHolder.unlockCanvasAndPost(canvas)**



- Re-paint the whole surface, otherwise previous state is remembered

SurfaceView with onTouchEvent()

- See the SurfaceViewTest example (some code is missing here!)

```
public class SurfaceViewTest extends Activity {
   MySurfaceView mySurfaceView;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      mySurfaceView = new MySurfaceView(this);
      setContentView(mySurfaceView);
   }

   @Override
   protected void onResume() {
      super.onResume();
      mySurfaceView.onResumeMySurfaceView();
   }

   @Override
   protected void onPause() {
      mySurfaceView.onPauseMySurfaceView();
      super.onPause();
   }
   class MySurfaceView extends SurfaceView
      implements Runnable, SurfaceHolder.Callback{
      Thread thread = null;
      SurfaceHolder surfaceHolder;

      public MySurfaceView(Context context) {
         super(context);
         surfaceHolder = getHolder();
         surfaceHolder.addCallback(this);
      }
      public void onResumeMySurfaceView(){
         running = true;
         thread = new Thread(this);
         thread.start();
      }
```

```
      @Override
      public void run() {
         while(running){
            if(surfaceHolder.getSurface().isValid()){
               Canvas canvas = surfaceHolder.lockCanvas();
               paint.setStyle(Paint.Style.STROKE);
               paint.setStrokeWidth(10);
               // a lot of drawPoint prepare code here...
               canvas.drawPoint(x, y, paint);
               if(touched){
                  paint.setStrokeWidth(100);
                  paint.setColor(Color.BLACK);
                  canvas.drawPoint(touched_x, touched_y, paint);
               }
               surfaceHolder.unlockCanvasAndPost(canvas);
            }
         }  // Implement to handle touch screen motion events
      }     // or implement View.OnTouchListener and override
      @Override  // onTouch(View v, MotionEvent event)
      public boolean onTouchEvent(MotionEvent event) {
         touched_x = event.getX();
         touched_y = event.getY();
         int action = event.getAction();
         switch(action){
            case MotionEvent.ACTION_DOWN:
            case MotionEvent.ACTION_MOVE:
               touched = true;
               break;
            case MotionEvent.ACTION_UP:
               touched = false;
               break;
            default:
         }
            return true;
      }
```

Also
See the Lunar Lander game in the SDK folder:/samples/[API] /legacy/LunarLander/

Hardware Acceleration

- Hardware acceleration is enabled by default if your Target API level is >=14, but can also be explicitly enabled. If your application uses only standard views and Drawables, turning it on globally should not cause any adverse drawing effects. However, because hardware acceleration is not supported for all of the 2D drawing operations, turning it on might affect some of your custom views or drawing calls.
- Problems usually manifest themselves as invisible elements, exceptions, or wrongly rendered pixels.
- You can control hardware acceleration at the following 4 levels

```
// Application
<application android:hardwareAccelerated="true" ...>
// Activity
<application android:hardwareAccelerated="true">
     <activity ... />
     <activity android:hardwareAccelerated="false" />
</application>
// Window
getWindow().setFlags(WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED,
    WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED);
// View
myView.setLayerType(View.LAYER_TYPE_SOFTWARE, null);
```

OpenGL ES I

- There are two foundational classes in the Android framework that let you create and manipulate graphics with the OpenGL ES API (javax.microedition.khronos.egl)
- GLSurfaceView
  - **This class is a View where you can draw and manipulate objects using OpenGL API calls and is similar in function to a SurfaceView.** You can use this class by creating an instance of GLSurfaceView and adding your Renderer to it. However, if you want to capture touch screen events, you should extend the GLSurfaceView class to implement - public boolean onTouchEvent(MotionEvent e)
- GLSurfaceView.Renderer
  - **This interface defines the methods required for drawing graphics in a GLSurfaceView**. You must provide an implementation of this interface as a separate class and attach it to your GLSurfaceView instance using GLSurfaceView.setRenderer().

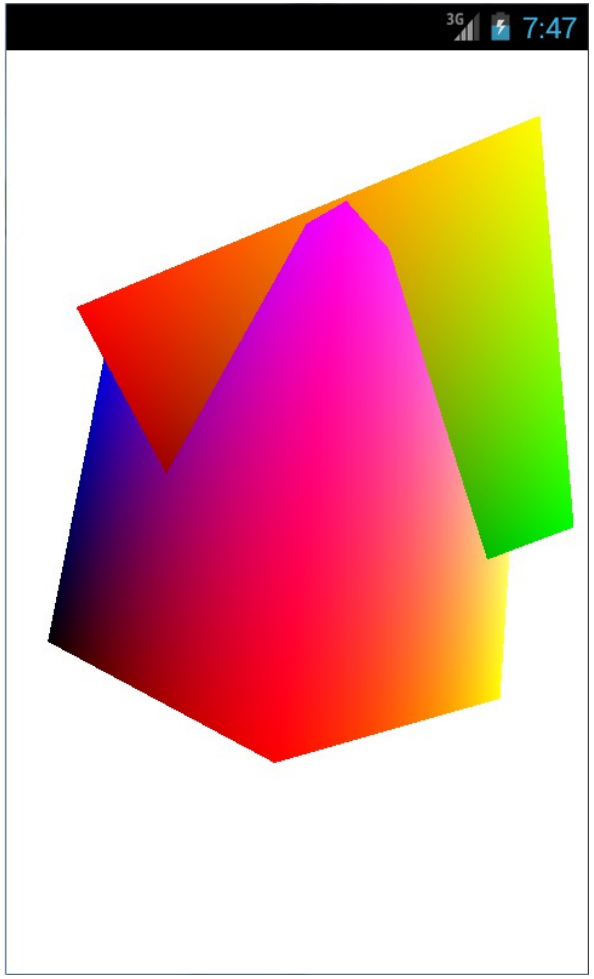
OpenGL ES II

- The GLSurfaceView.Renderer interface requires that you implement the following methods
  - onSurfaceCreated()
    - **The system calls this method once, when creating the GLSurfaceView.** Use this method to perform actions that need to happen only once, such as setting OpenGL environment parameters or initializing OpenGL graphic objects.
  - OnDrawFrame()
    - **The system calls this method on each redraw of the GLSurfaceView.** Use this method as the primary execution point for drawing (and re-drawing) graphic objects.
  - OnSurfaceChanged()
    - **The system calls this method when the GLSurfaceView geometry changes, including changes in size of the GLSurfaceView or orientation of the device screen.** For example, the system calls this method when the device changes from portrait to landscape orientation. Use this method to respond to changes in the GLSurfaceView container.
  - API Demos > Graphics > OpenGL ES

OpenGL ES III

- GLSurfaceView and TouchRotate are easy to understand

```
public class GLSurfaceViewActivity extends Activity {
    private GLSurfaceView mGLSurfaceView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

// Create our Preview view and set it as the content of our Activity
        mGLSurfaceView = new GLSurfaceView(this);
        mGLSurfaceView.setRenderer(new CubeRenderer(false));
        setContentView(mGLSurfaceView);
    }
    // Ideally a game should implement onResume() and onPause()
    // to take appropriate action when the activity looses focus
    @Override
    protected void onResume() {
        super.onResume();
        // Calling this method will recreate the OpenGL display and
        // resume the rendering thread
        mGLSurfaceView.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
        // Calling this method will pause the rendering thread
        mGLSurfaceView.onPause();
    }
}
```



OpenGL ES IV

```
public class CubeRenderer implements GLSurfaceView.Renderer {
    private boolean mTranslucentBackground;
    private Cube mCube;
    private float mAngle;
    public CubeRenderer(boolean useTranslucentBackground) { mTranslucentBackground = useTranslucentBackground;
        mCube = new Cube();
    }
    public void onDrawFrame(GL10 gl) {
       // clear the screen and draw some 3D objects
       gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
       gl.glMatrixMode(GL10.GL_MODELVIEW);
       gl.glLoadIdentity();
       gl.glTranslatef(0, 0, -3.0f); gl.glRotatef(mAngle,        0, 1, 0); gl.glRotatef(mAngle*0.25f,  1, 0, 0);
       gl.glEnableClientState(GL10.GL_VERTEX_ARRAY); gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
       mCube.draw(gl);
       gl.glRotatef(mAngle*2.0f, 0, 1, 1); gl.glTranslatef(0.5f, 0.5f, 0.5f);
       mCube.draw(gl);
       mAngle += 1.2f;
    }
    public void onSurfaceChanged(GL10 gl, int width, int height) {gl.glViewport(0, 0, width, height);
       // Set our projection matrix. This doesn't have to be done each time we draw
         float ratio = (float) width / height;
         gl.glMatrixMode(GL10.GL_PROJECTION);
         gl.glLoadIdentity();
         gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
    }
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
       // By default, OpenGL enables features that improve quality but reduce performance
        gl.glDisable(GL10.GL_DITHER);
       // Some one-time OpenGL initialization can be made here probably based on features of this particular context
         gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_FASTEST);
         if (mTranslucentBackground) {
             gl.glClearColor(0,0,0,0);
         } else {
             gl.glClearColor(1,1,1,1);
         }
         gl.glEnable(GL10.GL_CULL_FACE);
         gl.glShadeModel(GL10.GL_SMOOTH);
         gl.glEnable(GL10.GL_DEPTH_TEST);
    }
}
```

OpenGL ES V

- Declaring requirements in your AndroidManifest.xml file

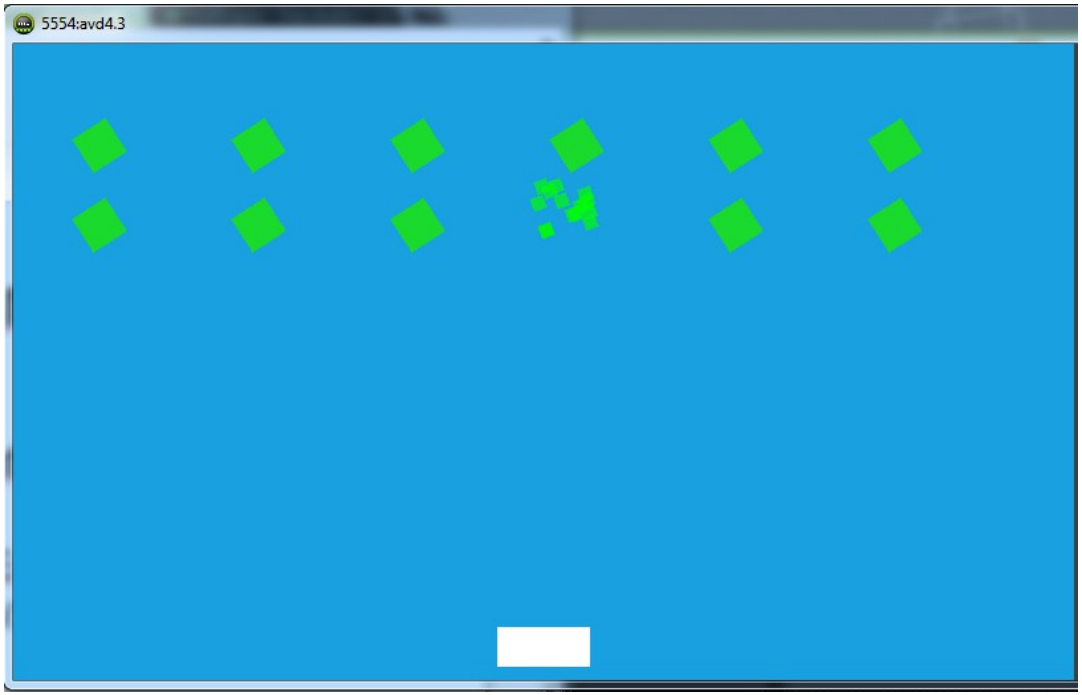
```
<!-- Tell the system this app requires OpenGL ES 3.0. -->
<uses-feature android:glEsVersion="0x00030000" android:required="true" />
```

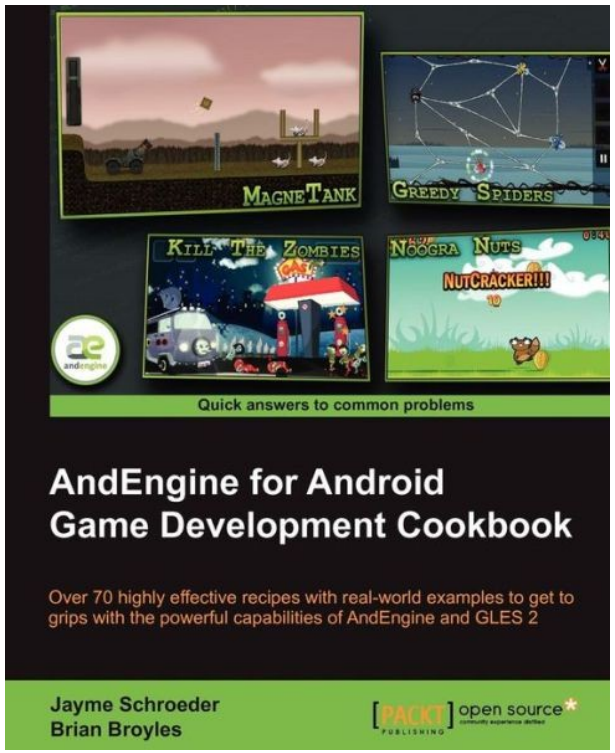
- Checking for OpenGL ES Version

```
private static double glVersion = 3.0;

private static class ContextFactory implements GLSurfaceView.EGLContextFactory {

     private static int EGL_CONTEXT_CLIENT_VERSION = 0x3098;

     public EGLContext createContext(
               EGL10 egl, EGLDisplay display, EGLConfig eglConfig) {

          Log.w(TAG, "creating OpenGL ES " + glVersion + " context");
          int[] attrib_list = {EGL_CONTEXT_CLIENT_VERSION, (int) glVersion,
                    EGL10.EGL_NONE };
          // attempt to create a OpenGL ES 3.0 context
          EGLContext context = egl.eglCreateContext(
                    display, eglConfig, EGL10.EGL_NO_CONTEXT, attrib_list);
          return context; // returns null if 3.0 is not supported;
     }
}
```

Graphic Libraries 1

- AndEngine and Simple Android Game/Github Tutorial
  - http://www.andengine.org/
  - https://github.com/nicolasgramlich
  - https://jimmaru.wordpress.com/2011/09/28/andengine-simple-android-game-tutorial/





Graphic Libraries 2

- Updated list of Game Engines for Android
  - http://software.intel.com/en-us/blogs/2012/03/13/game-engines-for-android
  - http://libgdx.badlogicgames.com/features.html

---

libGDX

News | Features | Download | Source | Documentation | Gallery | Community

Goals and Features

Libgdx is a Java game development framework that provides a unified API that works across all supported platforms.

The framework provides an environment for rapid prototyping and fast iterations. Instead of deploying to Android/iOS/Javascript after each code change, you can run and debug your game on the desktop, natively. Desktop JVM features like code hotswapping reduce your iteration times considerably.

Resources

- Animation and Graphics
  - http://developer.android.com/guide/topics/graphics/index.html
- Building Apps with Graphics & Animation
  - http://developer.android.com/training/building-graphics.html
- Resource Types
  - http://developer.android.com/guide/topics/resources/available-resources.html
- Android Advanced User Interface Development
  - http://www.vogella.com/android.html
- The API Demos sample projects
  - Android-SDK\samples\android-[API-#]\legacy
- Drawables
  - http://developer.android.com/guide/topics/resources/drawable-resource.html