UCD School of Computer Science

Advanced Techniques: Graphics – Animation - Sounds

Dr. Abraham(Abey) Campbell

Learning Objectives

- Customize a View in order to draw shapes, text, and images
- Animate shapes and images on a View
- Play a sound
- Create a simple game with animated characters and user interaction

Outline Graphic options on Android

- Graphics Options beyond using View/ Widget Objects
  - Drawable
  - Canvas
    - SurfaceView
- 3D Graphics option
  - GLSurface View
    - OpenGL ES
  - GoogleCardboard / Daydream API
  - AR library Artoolkit
  - AR Unity plugin Vuforia
- Video on Android

Remember everything is just a bitmap

- In Android , your screen is just a large buffer that the device draws onto
- Every view is just an underlying bitmap, and its canvas is attached to it.
- This change of the graphic buffer needs to happen sufficiently for the user to believe the screen is animating.

Drawable approach

- Android custom 2D graphics library
- Primary library : Android.graphics.drawable
- A drawable is an abstraction of anything that can be drawn
- As these are so important and can be unique to each phone/region, they use the res/drawable/ folder, using typical Android annotations
- There are three approach's the use an image or XML description or class constructors.
- For this course I will recommend just using the standard image, if you want to use animations, I will recommend the Canvas option for now.

Creating from resource images

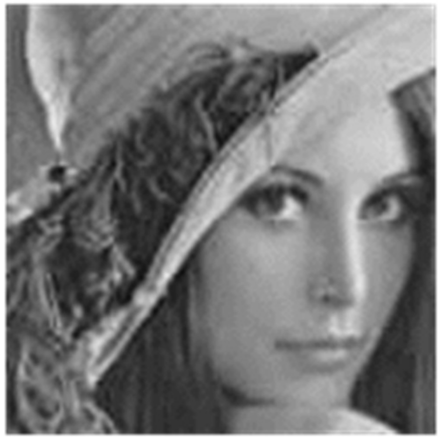
- Adding image files (normally PNG files are recommend)
- Remember PNG are Lossless compressed.
- JPG do work but they lossy compressed

- Lossless > lossy , as Lossless do not change the image but are reduced size due to the compression from a bitmap
- Lossy compression works by removing and changing information from an original bitmap, this compression is very smart as the human eye can be tricked easily.

Classic Lena example

Example of Lossy Compression

Original Lena Image (12KB size)


Lena Image, Compressed (85% less information, 1.8KB)

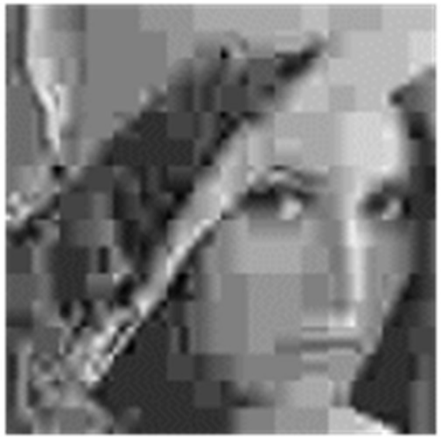

Lena Image, Highly Compressed (96% less information, 0.56KB)

Image Formats Supported

Image	JPEG	•	•	Base+progressive	JPEG (.jpg)
	GIF		•		GIF (.gif)
	PNG	•	•		PNG (.png)
	BMP		•		BMP (.bmp)
	WEBP	• (Android 4.0+)	• (Android 4.0+)		WebP (.webp)

Code example from Android Deveopler Doc’s

```
LinearLayout mLinearLayout;

protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  // Create a LinearLayout in which to add the ImageView
  mLinearLayout = new LinearLayout(this);

  // Instantiate an ImageView and define its properties
  ImageView i = new ImageView(this);
  i.setImageResource(R.drawable.my_image);
  i.setAdjustViewBounds(true); // set the ImageView bounds to match the Drawable's dimensions
  i.setLayoutParams(new Gallery.LayoutParams(LayoutParams.WRAP_CONTENT,
      LayoutParams.WRAP_CONTENT));

  // Add the ImageView to the layout and set the layout as the content view
  mLinearLayout.addView(i);
  setContentView(mLinearLayout);
}
```

Canvas approach

- You can direct add a canvas to your activity by extending a view
- Example in android-sdk\samples\android-15\Snake
- A better way would be to extend a Surfaceview object.
- Technically it is just extending a View but the SurfaceView is dedicated to drawing on a Canvas.
- Remember each view technical has its own canvas which is just a bitmap.
- Best example I cold find was actually not in Android SDK,
- https://github.com/MrBlackk/KillThemAll-Training
  - I have re-written this and adding in touch examples , this version will be put up on the moodle later so you do not have to take pictures ☺
- Remember GITHUB is a great source for tutorial, but learn from it , do not copy .

Canvas with Surface view

- First we need to setup our activity as usually but we add our custom Surface view called GameView

```
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(new GameView(this));
    }
}
```

Thread to keep updating the SurfaceView

```
public class GameLoop extends Thread {
private GameView view;
private boolean running = false;

public GameLoop(GameView view) {
this.view = view;
} public void setRunning(boolean run) {
running = run;
}
@Override
public void run() {
while (running) {
Canvas c = null;
try {
c = view.getHolder().lockCanvas();
synchronized (view.getHolder()) {

view.draw(c);
}
} finally {
if (c != null) {
view.getHolder().unlockCanvasAndPost(c);
}}}}}
```

GameView our extended SurfaceView

```
…
public class GameView extends SurfaceView {
…
public GameView(Context context) {
super(context);
gameLoop = new GameLoop(this);
holder = getHolder();
holder.addCallback(new SurfaceHolder.Callback() {
```

GameView our extended SurfaceView

```
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
        boolean retry = true;
        gameLoop.setRunning(false);
                while (retry) {
                        try {
                                gameLoop.join();
                                retry = false;
                            } catch (InterruptedException e) {
                                }}}
@Override
public void surfaceCreated(SurfaceHolder holder) {
        gameLoop.setRunning(true);
        gameLoop.start();
}
```

GameView our extended SurfaceView

```
@Override
public void surfaceChanged(SurfaceHolder holder, int format, int
width, int height) {
}
});
// need to load in our textures once and not on every redraw
Character = BitmapFactory.decodeResource(getResources(),
R.drawable.mario);
Background = BitmapFactory.decodeResource(getResources(),
R.drawable.background);
}
```

GameView our extended SurfaceView

```
 @Override
public void draw(Canvas canvas) {
          if (canvas != null)
                     {
                    ………………
                    canvas.drawBitmap(Background, 0, 0, null);
                    canvas.drawBitmap(Character, x, y +CurrentJumpHeight, null);
                    }
}
```

3D Graphics option /OpenGL ES

- Several version of OpenGL ES
- Version 1 is most similar to OpenGl 2
- Version 2 – 3 Is more like Current OpenGL 4.2
- Powerful but does require more battery life
- New Android phones have dedicated GPU chips

OpenGL ES activity

```
public class MainActivity extends Activity {

   private BasicGLSurfaceView mView;

   @Override
   protected void onCreate(Bundle icicle) {
      super.onCreate(icicle);
      mView = new
BasicGLSurfaceView(getApplication());
    setContentView(mView);
   }

   @Override
   protected void onPause() {
      super.onPause();
      mView.onPause();
   }

   @Override
   protected void onResume() {
      super.onResume();
      mView.onResume();
   }
}
```

OpenGL ES

- To use you will need to make a specialised SurfaceView

```
import android.opengl.GLSurfaceView;

class BasicGLSurfaceView extends GLSurfaceView {
        public BasicGLSurfaceView(Context context) {
        super(context);
        setEGLContextClientVersion(2);
        setRenderer(new MYTriangleRenderer(context));
    }
}
```

MyTriangleRenderer

- As OpenGL ES 2 , requires the programmer to setup the environment with far greater detail than lower version OpenGL
- To see full code
  - android-sdk\samples\android-15\BasicGLSurfaceView

- I recommend using the Canvas Object or alternatively use a plugin for a games engine like Unity
- This approach is difficult so Google has got API's to help you out

Google Daydream API / Google Cardboard

- Google VR SDK for Android
- Primary Immersive Virtual Reality API's for Android https://developers.google.com/vr/
- Outputs to either VR Environment supported by Android phones

Daydream

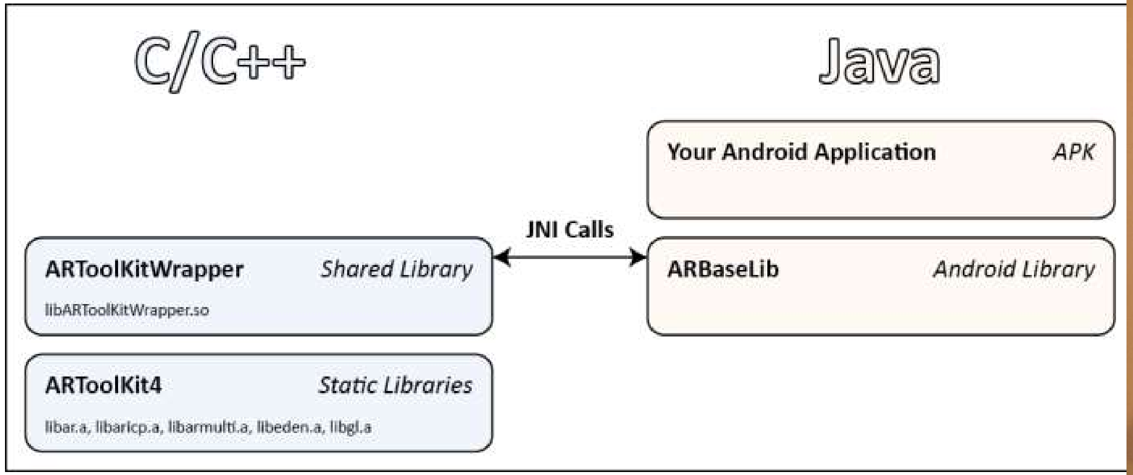
Cardboard

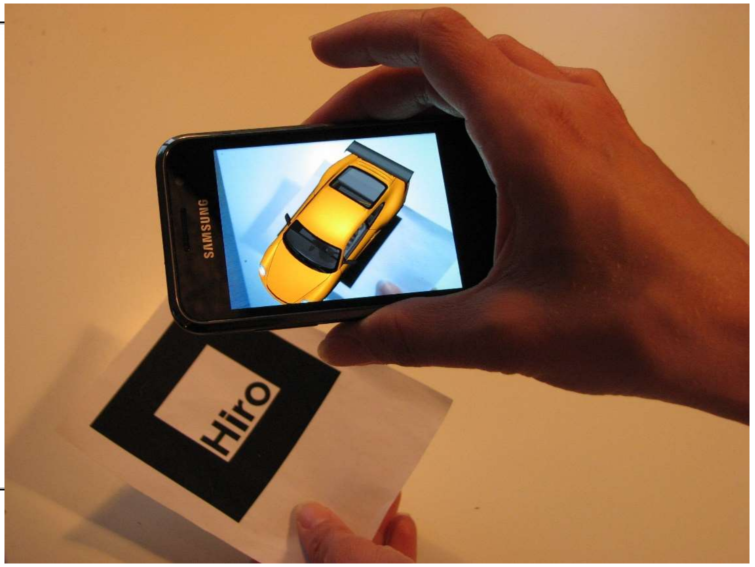
Google VR SDK supports

- Google VR SDK helps you with the following
  - Lens distortion correction.
  - Spatial audio.
  - Head tracking.
  - 3D calibration.
  - Side-by-side rendering.
  - Stereo geometry configuration.
  - User input event handling.
- OpenGL code
- Removes a lot of heavy lifting if you tried to just code a VR experience from scratch

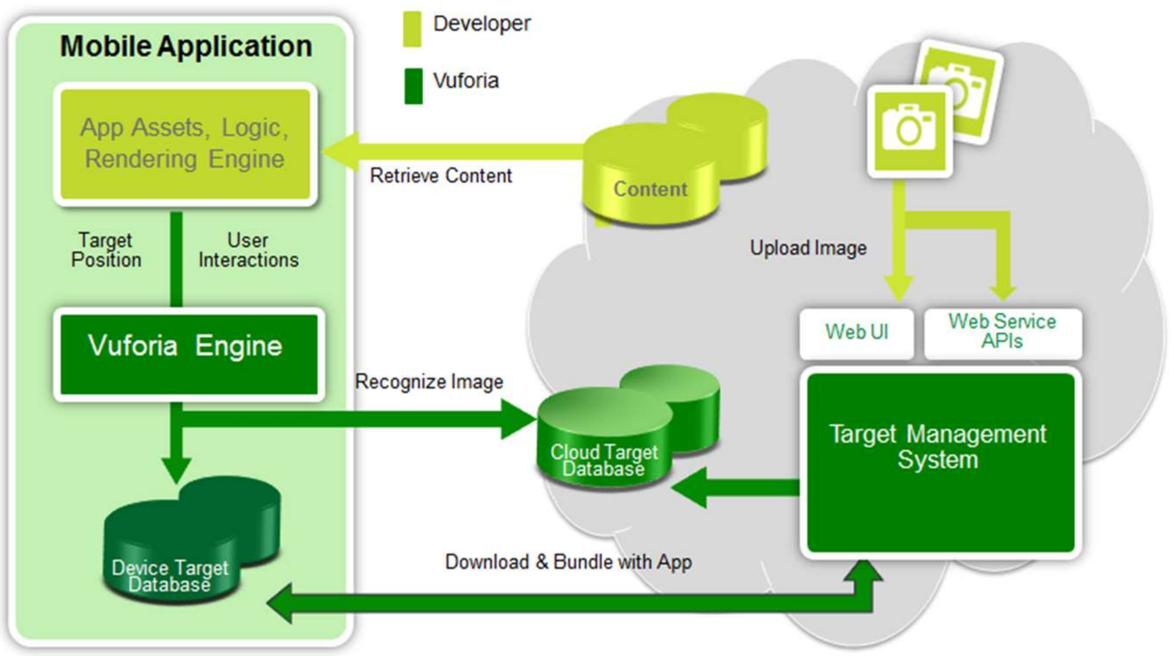
Augmented Reality in Android

- Artoolkit API for Android, compatible with Eclipse IDE only
- http://artoolkit.org/
- Alternative use is to use Artoolkit Plugin for UNITY





Augmented Reality Alternative : plugin Vurforia



- UNITY plugin so create 3D scene and then export
- More information go to https://www.vuforia.com/
- Easier to use than ARToolkit but less developer control

Last but not least : Video in Android

- To start playing a Sound or video file in android
- Use the MediaPlayer class.

```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sound_file_1);
mediaPlayer.start(); // no need to call prepare(); create() does that for you
```

- Remember if you want to add video directly in your app
- Use the VideoView and set its path

```
VideoView videoView = (VideoView)findViewById(R.id.VideoView);
VideoView.setVideoPath("/sdcard/video.mp4");
videoView.start();
```

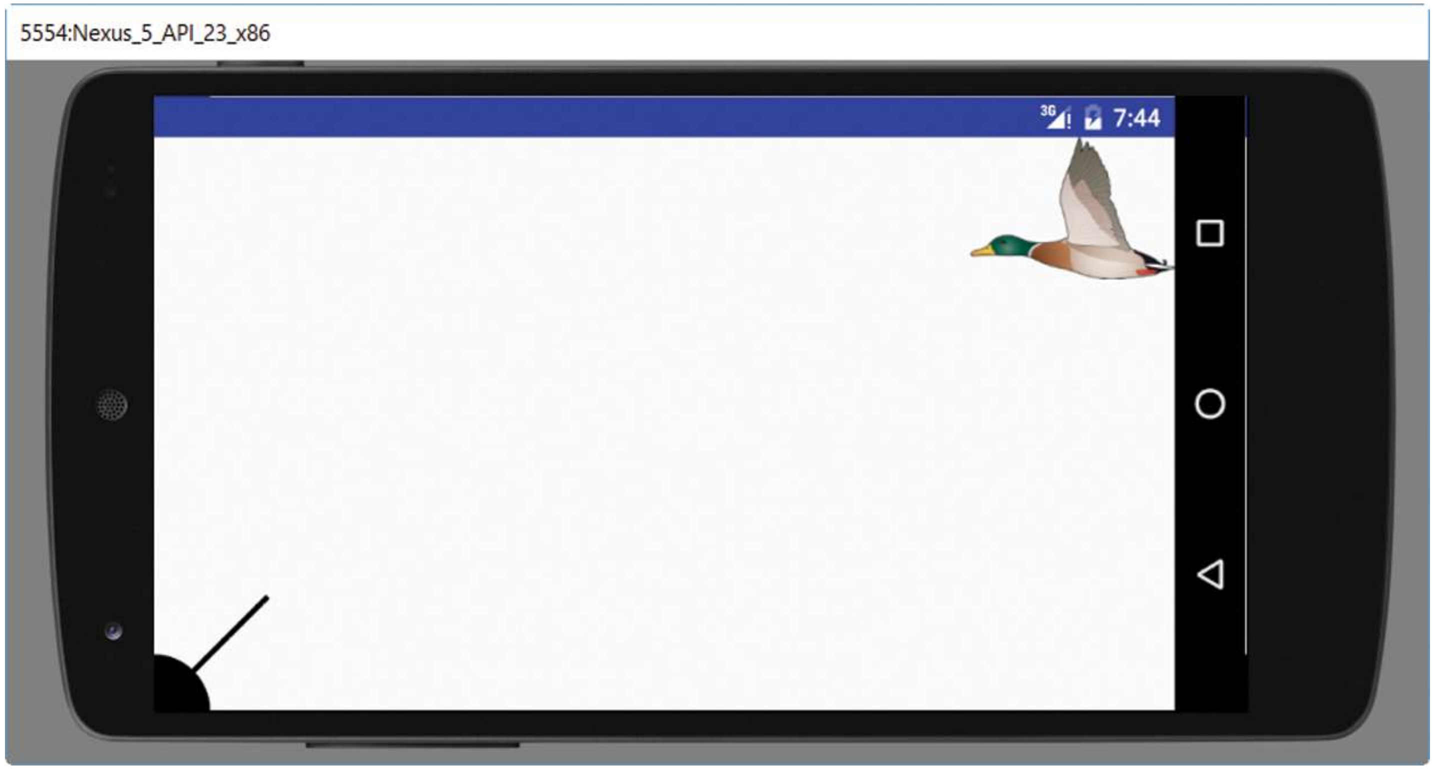
Video formats supported

Video	H.263	•	•		• 3GPP (.3gp) • MPEG-4 (.mp4)
	H.264 AVC	• (Android 3.0+)	•	Baseline Profile (BP)	• 3GPP (.3gp) • MPEG-4 (.mp4) • MPEG-TS (.ts, AAC audio only, not seekable, Android 3.0+)
	MPEG-4 SP		•		3GPP (.3gp)
	VP8		• (Android 2.3.3+)	Streamable only in Android 4.0 and above	• WebM (.webm) • Matroska (.mkv, Android 4.0+)

Duck Hunting Game (1 of 2)

- Drawing shapes
- Drawing a bitmap (from a picture)
- Playing a sound
- Capture and respond to touch events
- Animating the View

Duck Hunting Game (2 of 2)

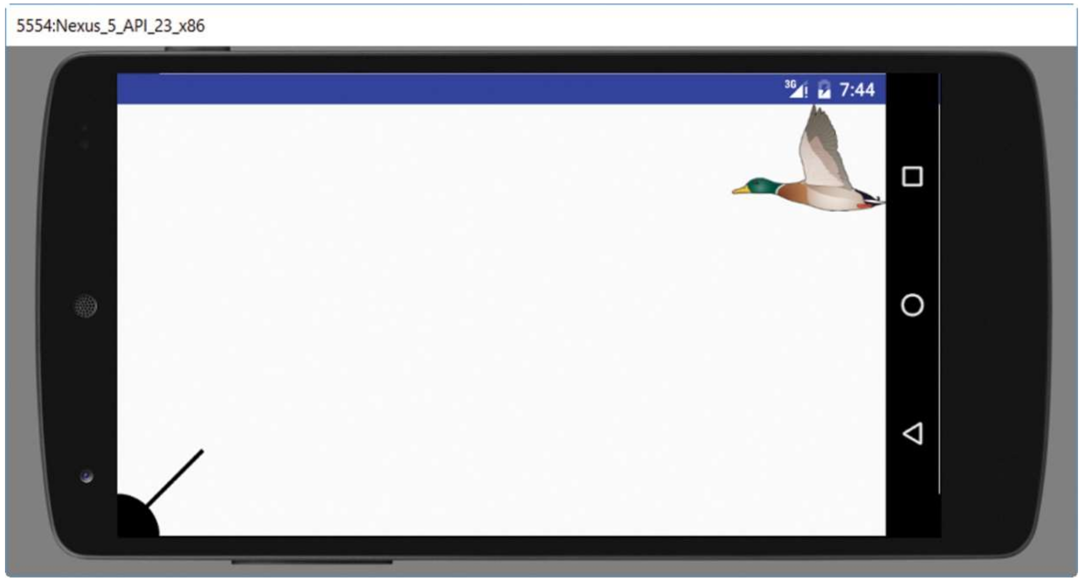


Animating the View

- We create a timer (Timer class) that schedules a task (TimerTask class) that will run at a given frequency (every 50 ms, or 100ms ...).
- Each time the task runs, we update the state of the game and redraw the View accordingly.

App Version 0—Drawing the Cannon and the Duck

- In Version 0, we draw the cannon and the duck.
- For the cannon, we draw shapes.
- The duck is a transparent png: we convert it to a Bitmap and we draw it.



Drawing the Cannon and the Duck

- The android.graphics contains the necessary classes to draw, set the current color ...
- Important classes: Bitmap, BitmapFactory, Paint, Color, Canvas.
- Paint: Defines style and color for drawing (how to draw).
- Canvas: Draws shapes and bitmaps (what to draw).

Drawing on a View (1 of 2)

- Typically, to draw on a View, we extend View and create a custom View class.
- Inside the custom View class, we override the inherited onDraw method; its API is:

```
public void onDraw( Canvas c )
```

- This is similar in Java to:

```
public void paint( Graphics g )
```

Drawing on a View (2 of 2)

```
public class GameView extends View {
  public void onDraw( Canvas canvas ) {
     super.onDraw( canvas );
     // draw the game view here
  }
}
```

Dimensioning the View

- We expect the GameView to be instantiated from a Context such as an Activity class.
- We want to allow the clients of the GameView class to dimension width and height) a GameView however they want.
- ➔ We provide three parameters to the constructor:

```
public GameView( Context context, int width,
 int height ) {
```

MainActivity

- We want a bigger screen ➔ landscape orientation only (edit manifest) and get rid of action bar.
- Instead of extending AppCompatActivity, we extend Activity.
- ➔ The action bar is not included.

Drawing on a View

- We want the app to run on any Android device, so we first retrieve the dimensions of the device (width and height) dynamically.
- We will then use these dimensions when we draw.

Setting the View

- Inside the Activity class, we need to set up the content View to be a GameView (our custom View class).
- First, we retrieve the available size (width and height).
- We retrieve the height of the status bar and subtract it from the height of the screen.

Height of the Screen

```
Point size = new Point( );
getWindowManager( ).getDefaultDisplay( )
                         .getSize( size );
```

- size.x = width of screen.
- size.y = height of screen.
- We need to subtract status bar height from size.y

Height of the Status Bar (1 of 2)

Height of the Status Bar (2 of 2)

```
Resources res = getResources( );
int statusBarHeight = 0;
int statusBarId = res.getIdentifier(
     “status_bar_height”, “dimen”, “android” );
if( statusBarId > 0 )
  statusBarHeight =
      res.getDimensionPixerlSize( statusBarId );
```

Setting the View

- Inside the Activity class, we need to set up the content View to be a GameView

```
private GameView gameView;
// inside onCreate
gameView = new GameView( this, size.x,
                    size.y - statusBarHeight );
setContentView( gameView );
```

GameView Class: onDraw Method

- We want to show a cannon base, a cannon barrel (we draw them), and a duck (we draw it from an image).
- Paint class: Defines style and color for drawing (how to draw).
- Canvas class: Draws shapes and bitmaps (what to draw).

Useful Methods of the Paint Class

- setARGB, setColor, setStrokeWidth, setTextSize, setStyle, setAntiAlias
- We create a Paint object and set its attributes before we use it to draw shapes and bitmaps.

Drawing on a View (1 of 2)

- We create and define a Paint object (paint is an instance variable).

```
paint = new Paint( );
```

- Set color to black:

```
paint.setColor( 0x FF000000 );
```

Drawing on a View (2 of 2)

- Set stroke width to 10:

```
paint.setStrokeWidth( 10.0f );
```

- Set antialias:

```
paint.setAntiAlias( true );
```

Useful Methods of the Canvas Class

- drawLine, drawLines, drawOval, drawCircle, drawPicture, drawBitmap, drawRect, drawText
- Typically, these methods accept several parameters—one of them is a Paint parameter.

Drawing on a View (1 of 2)

- To draw a circle, we can use the drawCircle method of the Canvas class.

```
public void drawCircle( float xCenter, float
        yCenter, float radius, Paint paint )
```

- Using the style and color defined in the Paint object paint:

```
    canvas.drawCircle( 50, 100, 25, paint );
```

Drawing on a View (2 of 2)

- We want to size the cannon base and barrel relative to the height of the screen.
- ➔ We add an instance variable to store the height (assign to it the height parameter of the constructor).
- ➔ height is now available in the onDraw method.

Drawing the Cannon (1 of 3)

- To draw the cannon, we draw a filled circle (cannon) and a line (cannon barrel).
- We center the circle at the left bottom corner.

```
canvas.drawCircle( 0, height,
height / 10, paint );
```

- ➔ only a quarter of the circle will show.

Drawing the Cannon (2 of 3)

- Note that since we only want a quarter of a circle, we could use the drawArc method (better, but a little more complex to use—we have to set the angles correctly).

Drawing the Cannon (3 of 3)

- For the cannon barrel, we draw a line:

  ```
  canvas.drawLine( 0, height, height / 5,
  height – height / 5, paint );
  ```

- ➔ The cannon barrel is at a 45-degree angle, starting from the left bottom corner.
- Note: The length of the cannon is relative to the height of the View.

Drawing the Duck

- To draw the duck, we use the drawBitmap method, which takes a Bitmap and a Rect objects (to position the bitmap) as two of its parameters.
- ➔ We need to create a Bitmap object from a file storing the picture of the duck.

drawBitmap (in Canvas Class)

public void drawBitmap(Bitmap bitmap, Rect

src, Rect dest, Paint paint)

- bitmap: the Bitmap to be drawn.
- src: a rectangle within the bitmap (if null, we draw the whole bitmap).
- dest: the rectangle where we draw the bitmap.
- paint: the Paint context.

Creating the Bitmap (1 of 3)

- The decodeResource static method of the BitmapFactory class returns a Bitmap object:

```
public static Bitmap decodeResource(
    Resources res, int id )
```

- res: a Resources object reference
- id: the id of the resource (a png file here)

Creating the Bitmap (2 of 3)

- If duck.png is placed in the drawable directory, we can refer to it using R.drawable.duck (note that the extension is not included).
- getResources is inherited by Activity from ContextThemeWrapper; it returns a Resources instance for the app package.

Creating the Bitmap (3 of 3)

- We add an instance variable named duck, of type Bitmap.
- Inside the GameView constructor:
  - We define the int constant TARGET, give it the value R.drawable.duck.

```
duck = BitmapFactory.decodeResource(
            getResources( ), TARGET );
```

Bitmap Class (1 of 2)

- We can access every pixel of a Bitmap object.

int getPixel(int x, int y) ➔ returns the color of the pixel as an int

void setPixel(int x, int y, int color) ➔ colors a pixel with color

Bitmap Class (2 of 2)

- We can access the width and height of a Bitmap object:

```
int getWidth( )
int getHeight( )
```

Creating the Destination Rectangle for the Duck (1 of 4)

- We want to draw the duck in a rectangle whose size is relative to the size of the screen.
- We want to keep the same width/height ratio as in the original image of the duck.

Creating the Destination Rectangle for the Duck (2 of 4)

- We set the width of the rectangle in which we draw the duck to be equal to 20% (1/5) of the screen’s width.
- We scale the height of the rectangle using a scaling factor.

Creating the Destination Rectangle for the Duck (3 of 4)

- The width of duck becomes width / 5 (where width is the width of the screen).
- The scaling factor is (width / 5) divided by the width of the duck image:

```
 float scale =
    ( ( float ) width / ( duck.getWidth( ) * 5 ) );
```

Creating the Destination Rectangle for the Duck (4 of 4)

- We add the instance variable duckRect, a Rect.
- Inside the GameView constructor:
  - Rect constructor is Rect(left, top, right, bottom), all four parameters are ints:

```
duckRect = new Rect( width – width / 5, 0,
width, ( int ) ( duck.getHeight( ) * scale ) );
```

Drawing the Duck

- Inside onDraw:

```
canvas.drawBitmap( duck, null,
duckRect, paint );
```

The Model for the App (1 of 3)

- We build a Game class to encapsulate the functionality of the game.
- We need to manage the game, that is:
  - The cannon
  - The bullet
  - The duck

The Model for the App (2 of 3)

- The cannon has a center, a radius, a barrel length, and an angle (which is subject to user interaction).
- The bullet has a radius, a position, a state (fired or not), a speed, and an angle (at which it was fired).
- The duck has a width, height, position, speed, and state (alive or shot).

The Model for the App (3 of 3)

- The Game class includes all this functionality, including methods to move the duck, detects whether it has been hit ...
- See Game.java

Version 1: Making the Duck Fly

- In Version 1, we animate the duck: it flies horizontally from right to left.
- We refresh and redraw the View at a certain frequency, say 10 frames per second, i.e., every 100 ms (usually, it is <= 50 ms, i.e., >= 20 fps).
- The postInvalidate method (from the View class) forces a call to onDraw (which draws the View).
- Note: This is similar to repaint in Java.

Refreshing the View

- The class Timer, in the java.util package, has several methods to schedule tasks that execute at given time intervals.

```
public void schedule( TimerTask task,
    long delay, long period )
```

- It schedules task to be run every period ms, starting after delay ms.

Sub-classing TimerTask (1 of 2)

- The schedule method takes a TimerTask parameter.
- TimerTask is abstract and includes the abstract method run.
- When sub-classing TimerTask, we need to implement the run method.
- run is called automatically at the specified time frequency.

Sub-classing TimerTask (2 of 2)

```
public class GameTimerTask extends TimerTask {
  // instance variables and constructor
  public void run( ) {
    // code to update game
  }
}
```

GameTimerTask—Instance Variables

- We use the Model to move the duck; thus, we need a reference to the Model.
- ➔ We add an instance variable game of type Game.
- To redraw the View, we need a reference to the View.
- ➔ We add an instance variable gameView of type GameView.

GameTimerTask—Constructor

```
public class GameTimerTask( GameView
          view ) {
  gameView = view;
  game = view.getGame( );
  game.startDuckFromRightTopHalf( );
}
```

GameTimerTask, Run Method (1 of 2)

```
public class GameTimerTask extends TimerTask {
 public void run( ) {
   // move the duck
   // if duck is off screen, restart it from the right
   // redraw the game View
 }
}
```

GameTimerTask, Run Method (2 of 2)

```
public void run( ) {
  game.moveDuck( );
  if( game.duckOffScreen( ) )
    game.startDuckFromRightTopHalf( );
  // redraw the View
  gameView.postInvalidate( );
 }
```

MainActivity Class

- Initialize timer and schedule a GameTimerTask (if not done already).

```
Timer gameTimer = new Timer( );
gameTimer.schedule( new
GameTimerTask( gameView ), 0,
GameView.DELTA_TIME );
```

- We need to add the DELTA_TIME constant in the GameView class.

GameView Class (1 of 4)

- We add new instance variables:
  - Game game: the Model.
  - Bitmap [ ] ducks: an array of duck Bitmaps (so we can animate the duck when it is flying).
  - int duckFrame: tracks the current Bitmap frame (index of array ducks) being drawn.

GameView Class (2 of 4)

- We use four frames to show the duck flying (from three pngs ➔ three resources)
- We add an array of Bitmaps as an instance variable (ducks), replacing duck; it has four elements (pngs 0, 1, 2, 1).
- We can add more frames in future versions for more realistic flying motion.

GameView Class (3 of 4)

```
private int [ ] TARGETS = {
  R.drawable.anim_duck0,
  R.drawable.anim_duck1,
  R.drawable.anim_duck2,
  R.drawable.anim_duck1 };

private Bitmap [ ] ducks;
private int duckFrame;
```

GameView Class (4 of 4)

- Inside the constructor, we initialize ducks and game, passing the available screen size.
- We call setHuntingRect and setCannon, passing dimensions relative to the available screen size.
- See GameView class.

GameView Class—onDraw (1 of 2)

- Draw the cannon based on the game’s size.
- Draw the cannon barrel based on the game’s size.
- Update duckFrame (add 1 mod 4).
- Draw the current duck frame.

GameView Class—onDraw (2 of 2)

- Update duckFrame (add 1 mod 4)

```
duckFrame = ( duckFrame + 1 ) %
            ducks.length;
```

- Draw the current duck frame

```
canvas.drawBitmap( ducks[duckFrame], null,
  game.getDuckRect( ), paint );
```

- See GameView class.

Version 2

- Move the cannon and enable shooting
- ➔ set up event handling
- Touches and touch motion ➔ move the cannon
- Double tap ➔ shoot

Handling Events (1 of 2)

- There are several classes available to handle touch events. We want to handle gestures and taps.
- The GestureDetector class, along with its static inner interfaces GestureDetector. OnGestureListener and GestureDetector. OnDoubleTapListener, provides the tools to handle gestures and taps.

Handling Events (2 of 2)

- The GestureDetector class has a static inner class, GestureDetector.SimpleOnGestureListener, that implements these two static inner interfaces (OnGestureListener and OnDoubleTapListener) with do nothing methods.

GestureDetector.SimpleOnGestureListener

- Useful Methods to handle our touch events:
  - onSingleTapConfirmed ➔ move cannon barrel
  - onScroll ➔ move cannon barrel
  - onDoubleTap ➔ shoot

Capturing Touch Events (1 of 2)

- Use GestureDetector
- Create a private inner class (we call it TouchHandler) that extends GestureDetector.SimpleOnGestureListener
- Override onSingleTapConfirmed, onScroll, and onDoubleTapEvent methods

Capturing Touch Events (2 of 2)

- onSingleTapConfirmed and onScroll ➔ move the cannon barrel (since these two methods both move the cannon barrel, they can both call the same method to do that)
- onDoubleTapEvent ➔ shoot, i.e., fire a bullet

Handling Touch Events (1 of 2)

- Add GestureDetector instance variable:

```
private GestureDetector detector;
```

- Code private inner class extending GestureDetector.SimpleOnGestureListener.

Handling Touch Events (2 of 2)

```
private class TouchHandler
        extends
GestureDetector.SimpleOnGestureListener {
```

onSingleTapConfirmed Method

```
public boolean onSingleTapConfirmed(
           MotionEvent event) {
  updateCannon( event ); // moves cannon
  return true;
}
```

onScroll Method

```
public boolean onScroll( MotionEvent
event1,
    MotionEvent event2, float d1, float d2 ) {
  updateCannon( event2 ); // moves cannon
  return true;
}
```

updateCannon Method (1 of 2)

```
public void updateCannon( MotionEvent
          event ) {
  // calculate new angle for the cannon barrel
  // call setCannonAngle with game
}
```

updateCannon Method (2 of 2)

```
float x =
   event.getX( ) - game.getCannonCenter( ).x;
float y =
   game.getCannonCenter( ).y - event.getY( );
float angle = ( float ) Math.atan2( y, x );
game.setCannonAngle( angle );
```

onDoubleTapEvent Method (1 of 2)

```
public boolean onDoubleTapEvent(
            MotionEvent event ) {
  // if the bullet has not bee fired yet,
  //   fire the bullet
  // consume the event (return true)
}
```

onDoubleTapEvent Method (2 of 2)

```
public boolean onDoubleTapEvent(
          MotionEvent event ) {
  if( !( game.isBulletFired( ) ) )
    game.fireBullet( );
  return true;
}
```

GameView Class (1 of 3)

private GestureDetector detector;

- Inside GameView constructor:

  TouchHandler th = newTouchHandler( );

- Set up touch and scroll event dispatching.
- Set up double tap event dispatching.

GameView Class (2 of 3)

- Set up touch and scroll event dispatching:

```
detector =
new GestureDetector( getContext( ), th );
```

- Set up double tap event dispatching:

```
detector.setDoubleTapListener( th );
```

GameView Class (3 of 3)

- The onTouchEvent(MotionEvent event) method from the View class (inherited by GameView) is automatically called when a MotionEvent happens.
- Inside it, we can dispatch the event for handling to methods of the class extending GestureDetector.SimpleOnGestureListener.

Inside onTouchEvent

```
public boolean onTouchEvent( MotionEvent
          event ) {
  // dispatch event handling to th
  detector.onTouchEvent( event );
  return true;
}
```

onDraw Method

- We need to add the code to draw the bullet in the onDraw method:

```
if( ! game.bulletOffScreen( ) )
  canvas.drawCircle( game.getBulletCenter( ).x,
                     game.getBulletCenter( ).y,
                     game.getBulletRadius( ),
                     paint );
```

GameTimerTask Class

- In the run method, we now need to manage the bullet:
  - If it is off the screen, reload.
  - If it is on the screen, update its position.
- For both, we call methods of the Model.

GameTimerTask Class, Run Method

```
if( game.bulletOffScreen( ) )
   game.loadBullet( );
else if( game.isBulletFired( ) )
   game.moveBullet( );
```

Version 3

- We want to play a sound when the bullet is fired and when the duck is hit.
- Collision detection: We need to detect whether the duck has been hit.
- We animate the duck when it is shot.

Sounds

- The SoundPool class can be used to manage and play sounds.
- load, play, pause, resume methods
- API level 21 ➔ update gradle file, specify min sdk of 21

Using SoundPool Class

- The constructor

  SoundPool(int maxStreams, int streamType,

  int srcQuality)

  is now deprecated.

SoundPool.Builder

- SoundPool.Builder, an inner static class of SoundPool, has the factory method build( ) that creates and returns a SoundPool object:

```
SoundPool.Builder poolBuilder = new
  SoundPool.Builder( );
SoundPool pool = poolBuilder.build( );
```

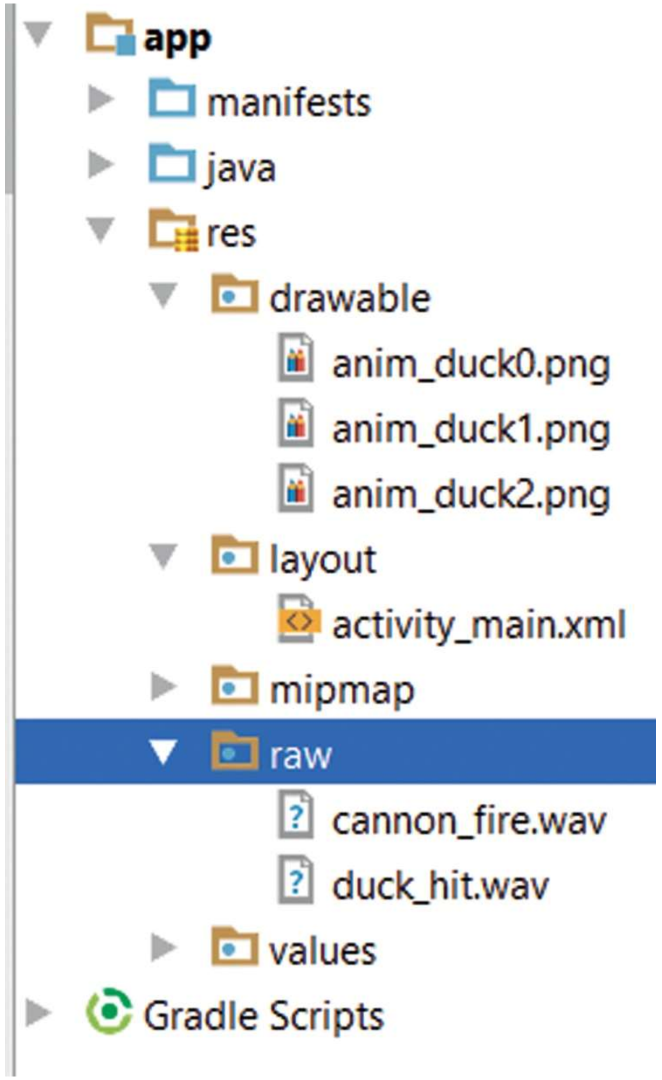
Using SoundPool Class

- We instantiate a SoundPool object.
- We call one of the load methods to load a sound either from a resource or by specifying a file path; each load method returns an id for the sound.
- To play a sound, we call the play method, passing the id of the sound to be played. We can play the sound in a loop or one time.

Loading a Sound

- It is common to create a directory named raw and place the sound files in it.
- We can then load a sound as a resource (the sound is in the raw subdirectory of the res directory).

The Raw Directory



```
▼ app
  ▶ manifests
  ▶ java
  ▼ res
    ▼ drawable
        anim_duck0.png
        anim_duck1.png
        anim_duck2.png
    ▼ layout
        activity_main.xml
    ▶ mipmap
    ▼ raw
        cannon_fire.wav
        duck_hit.wav
    ▶ values
▶ Gradle Scripts
```

Loading a Sound

```
int load( Context context, int resId, int priority )
```

- priority is not used at this time ➔ use 1 as default.
- To load cannon_fire.wav, located in the raw directory (we create the raw directory) of the res directory.

```
        int fireSoundId = pool.load( getContext( ),
          R.raw.cannon_fire, 1 );
```

Playing a Sound

- Use the play method of SoundPool:

```
play( int soundId, float leftVolume, float
        rightVolume, int priority, int loop, float rate )
```

- loop value: -1 ➔ loop, 0 ➔ 1 time only, 1 ➔ 2 times
- rate value is the playback speed: range is 0.5 to 2 ➔ 1 = original speed

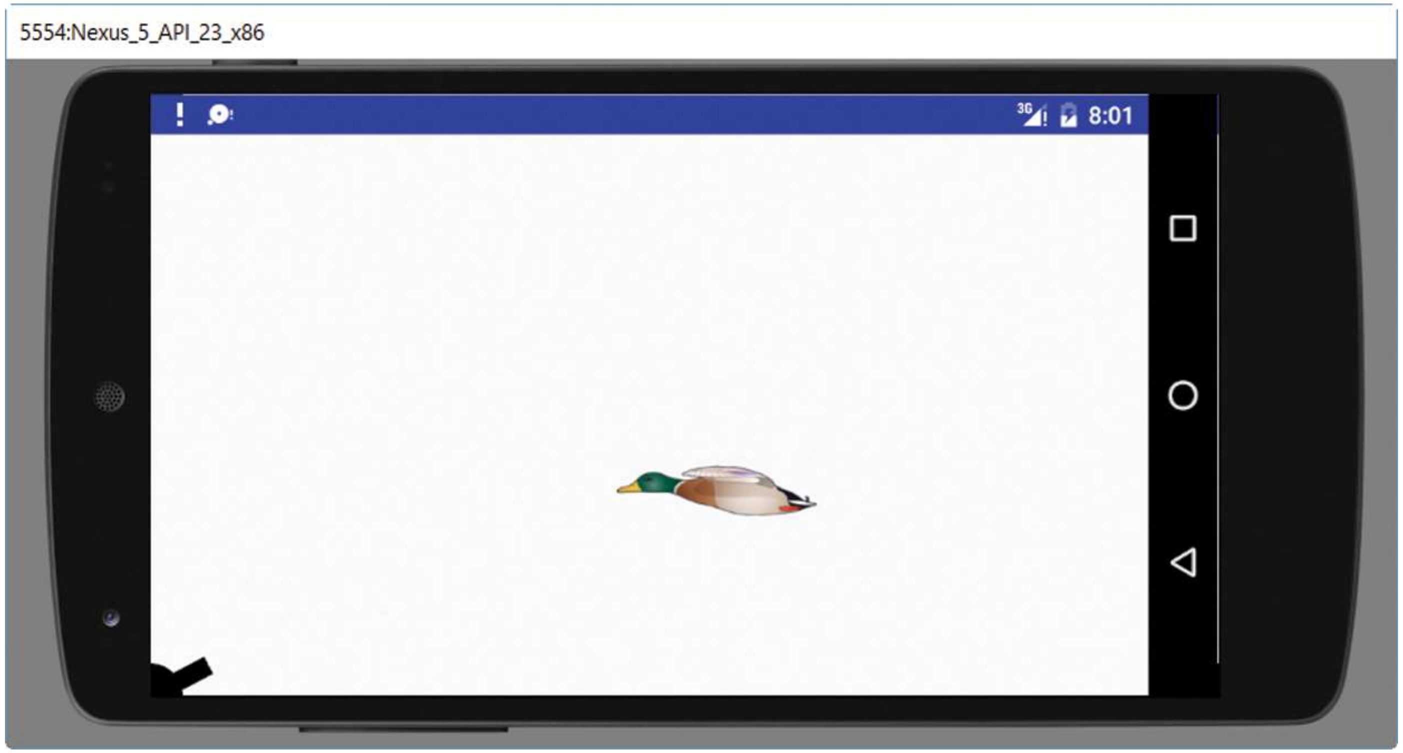
```
pool.play( fireSoundId, 1.0f, 1.0f, 0, 0, 1.0f );
```

The Duck Has Been Hit (1 of 2)

- Detect whether the duck has been hit.
- If it has been hit:
  - Play a sound
  - Update the game (duck has been hit)
  - Load the bullet
  - No longer animate the duck flying using four frames: use the first frame only

The Duck Has Been Hit (2 of 2)

- When the duck is hit, it goes straight down and is no longer animated.



Inside the Run Method

```
if( game.duckOffScreen( ) ) {
   game.setDuckShot( false );
   game.startDuckFromRightHalf( );
} else if( game.duckHit( ) ) {
   game.setDuckShot( true );
   gameView.playHitSound( );
   game.loadBullet( );
}
```

Inside onDraw Method

```
if( game.isDuckShot( ) )
   canvas.drawBitmap( ducks[0],
                     null, game.getDuckRect( ), paint );
else
   canvas.drawBitmap( ducks[duckFrame], null,
                     game.getDuckRect( ), paint );
```

Duck Hunting Game

- Paint and Canvas classes
- Coding a custom View
- Drawing shapes
- Drawing a bitmap (from a picture)
- Playing a sound
- Capture and respond to touch events
- Animating the View (Timer, TimerTask)