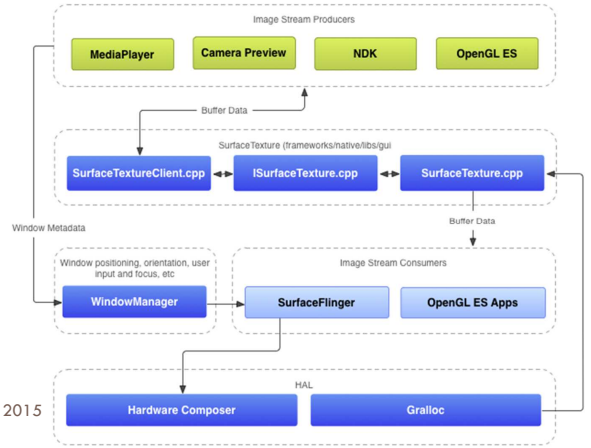
ANDROID APPS DEVELOPMENT FOR MOBILE GAME

Lecture 4: Canvas and Animation

Peter Lo

Graphics

- There are two general ways that app developers can draw things to the screen: Canvas or OpenGL.



2D Drawing in Android

- Android provides a set of 2D-drawing APIs to render graphics onto a canvas:
  - Draw graphics or animations into a View object from layout.
    - Best choice when want to draw simple graphics that do not need to change dynamically and are not part of a performance-intensive game.
  - Draw graphics directly to a Canvas.
    - Better when application needs to regularly re-draw itself. Applications such as video games should be drawing to the Canvas on its own.

Creating Custom View

- All of the view classes defined in the Android framework extend View. Your custom view can also extend View directly, or you can save time by extending one of the existing view subclasses

```
public class MyView extends View {
    public MyView(Context context) {
        super(context);
    }
}
```

Create a custom view class by extending the View class

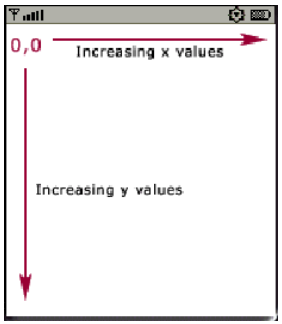
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    View MyView = new MyView(this);
    setContentView(MyView);
}
```

Create a custom view instance and set it as the ContentView for this Activity.

The Coordinate System

- The coordinate system for the Canvas class originates in the upper-left corner of the display.
  - x values increase moving to the right;
  - y values increase going down.
- When drawing, the default pen thickness is one pixel in width and height.



Screen Width and Height

- The height and width of the Canvas represent the entire drawing area available to the device display.
- To determine the width and height of the canvas, you can use the method *getDisplayMetrics()*.

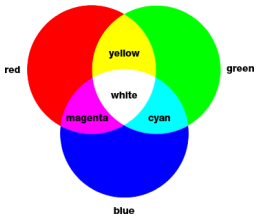
```
public class MyView extends View {
    public MyView(Context context) {
        super(context);

        // Obtain the screen dimension
        DisplayMetrics metrics = this.getResources().getDisplayMetrics();
        int ScreenWidth = metrics.widthPixels;
        int ScreenHeight = metrics.heightPixels;
    }
}
```

Obtain the screen size using the method *getDisplayMetrics()*, then get the screen width from *widthPixels*, and screen height from *heightPixel*.

Define Color

- Red, Green, and Blue components of the three primary colors.
- The color can be define by:
  - *paint.setColor(0xFF0000);*
  - *paint.setColor(Color.RED);*

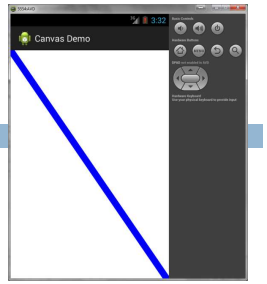


Drawing on a View

- If the application does not require a significant amount of processing or frame-rate speed, then should consider creating a custom View component and drawing with a Canvas in *View.onDraw()*.
- The most convenient aspect of doing so is that the Android framework will provide a pre-defined Canvas to place the drawing calls.
- To start, extend the View class and define the *onDraw()* callback method. This method will be called by the Android framework to request that the View draw itself.

Drawing Line

- Draw a line segment with the specified start and stop x, y coordinates, using the specified paint.




```
public class MyView extends View {
    private Paint mPaint;

    public MyView(Context context) {
        super(context);
        mPaint = new Paint();
    }

    public void onDraw(Canvas canvas) {
        mPaint.setAntiAlias(true);
        mPaint.setStrokeWidth(20);
        mPaint.setColor(Color.BLUE);
        canvas.drawLine(0, 0, 480, 700, mPaint);
    }
}
```

Constructor for the custom view

An overridden *onDraw()* method to draw onto the provided canvas.

The custom view can extend View directly for saving time

Create a new paint with default settings

Smooth out the edges

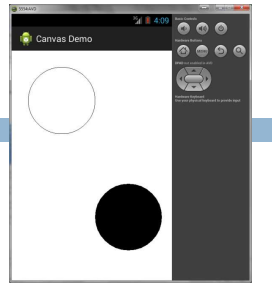
Define the width for the line

Define the Color

Draw line on screen

Drawing Sharp

- Draw a circle with the specified center x, y coordinates, and radius r using the specified style.




```
public class MyView extends View {
    private Paint mPaint;

    public MyView(Context context) {
        super(context);
        mPaint = new Paint();
    }

    public void onDraw(Canvas canvas) {
        mPaint.setStyle(Paint.Style.STROKE);
        canvas.drawCircle(150, 150, 100, mPaint);
        mPaint.setStyle(Paint.Style.FILL);
        canvas.drawCircle(350, 500, 100, mPaint);
    }
}
```

Set the drawing style to Stroke

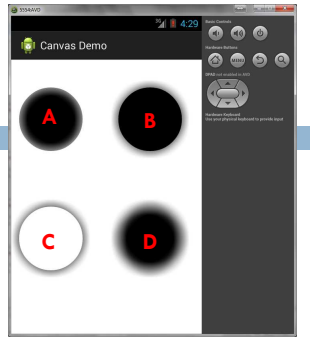
Draw a circle on Screen

Set the drawing style to Fill

Draw a filled circle on Screen

Drawing with Blur Effect

- *BlurMaskFiltertakes* a mask, and blurs its edge by the specified radius.




```
public class MyView extends View {
    private Paint mPaint;

    public MyView(Context context) {
        super(context);
        mPaint = new Paint();
    }

    public void onDraw(Canvas canvas) {
        mPaint.setStyle(Paint.Style.FILL);
        mPaint.setMaskFilter(new BlurMaskFilter(20, Blur.INNER));
        canvas.drawCircle(100, 150, 80, mPaint);
        mPaint.setMaskFilter(new BlurMaskFilter(20, Blur.OUTER));
        canvas.drawCircle(100, 450, 80, mPaint);
        mPaint.setMaskFilter(new BlurMaskFilter(20, Blur.SOLID));
        canvas.drawCircle(350, 150, 80, mPaint);
        mPaint.setMaskFilter(new BlurMaskFilter(20, Blur.NORMAL));
        canvas.drawCircle(350, 450, 80, mPaint);
    }
}
```

Draw a filled circle with inner blur mask effect (Circle A)

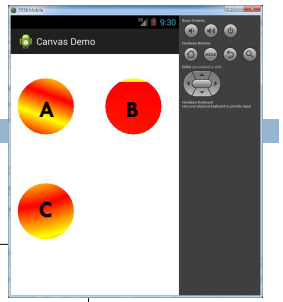
Draw a filled circle with outer blur mask effect (Circle B)

Draw a filled circle with solid blur mask effect (Circle C)

Draw a filled circle with normal blur mask effect (Circle D)

Drawing with Gradient Effect

- *setShader* is used to set or clear the shader object with gradient.




```
public class MyView extends View {
    private Paint mPaint;
    private Shader shader;

    public MyView(Context context) {
        super(context);
        mPaint = new Paint();
    }

    public void onDraw(Canvas canvas) {
        shader = new LinearGradient(8, 80, 30, 20, Color.RED,Color.YELLOW, TileMode.MIRROR);
        mPaint.setShader(shader);
        canvas.drawCircle(100, 150, 80, mPaint);
        shader = new RadialGradient(8, 80, 90, Color.RED,Color.YELLOW, TileMode.MIRROR);
        mPaint.setShader(shader);
        canvas.drawCircle(100, 450, 80, mPaint);
        shader = new SweepGradient(80, 80, Color.RED, Color.YELLOW);
        mPaint.setShader(shader);
        canvas.drawCircle(350, 150, 80, mPaint);
    }
}
```

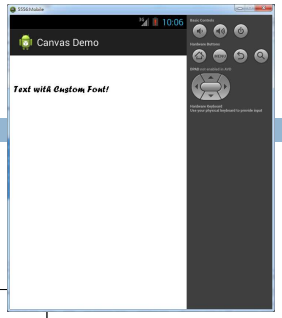
Draw a filled circle with linear gradient effect (Circle A)

Draw a filled circle with radial gradient effect (Circle B)

Draw a filled circle with sweep gradient effect (Circle C)

Drawing with Custom Font

- Define the typeface and intrinsic style of a font.




```
public class MyView extends View {
    private Paint mPaint;
    private Typeface myFont;

    public MyView(Context context) {
        super(context);
        mPaint = new Paint();
    }

    public void onDraw(Canvas canvas) {
        myFont = Typeface.createFromAsset(getContext().getAssets(), "forte.ttf");
        mPaint.setTypeface(myFont);
        mPaint.setTextSize(25);
        canvas.drawText("Text with Custom Font!", 10, 100, mPaint);
    }
}
```

Load the True Type Font from folder: res/asset/ (the filename must in small letter with space)

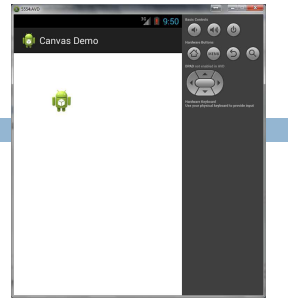
Define the font face

Define the font size

Draw the text with custom font

Drawing Bitmap

- Draw the specified bitmap, with its top/left corner at (x,y), using the specified paint, transformed by the current matrix.




```
public class MyView extends View {
    public MyView(Context context) {
        super(context);
    }

    public void onDraw(Canvas canvas) {
        Bitmap myBitmap = BitmapFactory.decodeResource(getResources(),
                R.drawable.ic_launcher);
        canvas.drawBitmap(myBitmap, 100, 100, null);
    }
}
```

Load the bitmap from the folder: res/drawable/ (the filename must in small letter with space)

Draw the bitmap “ic_launcher.png” at (100, 100)

OnTouch handler

- Unlike widgets, graphics use an alternative callback for touch events which allow for the handling of gestures.
- When the user touches/clicks on the View, Android invokes the *onTouchEvent()* method
- The *onTouch* method has two parameters :
  - *View* which was touched
  - *MotionEvent* object contains following information:
    - The type of Action (Up, Down / Move, Release)
    - Where the event occurred (X, Y coordinate)
    - The time at which the event occurred

Refresh Screen

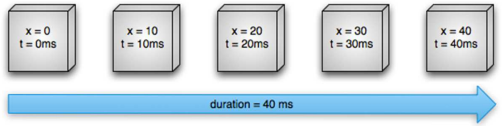
- Once your *onDraw()* is complete, the Android framework will use your Canvas to draw a Bitmap handled by the system
  - Inside your View component's *onDraw()*, use the Canvas given to you for all your drawing using various draw methods
- Android only call *onDraw()* as necessary.
  - Each time that your application is prepared to be drawn, you must request your View be invalidated by calling *invalidate()*.
  - This indicates that you'd like your View to be drawn and Android will then call your *onDraw()* method.
- Once any states have been updated, the View can be redrawn by calling the *invalidate()* method.

Animation

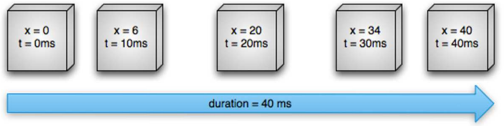
- For animation to work, the pictures making up the animation must replace one another quickly enough to trick the human eye into believing there is movement.
- A replacement rate of at least 14 frames per second or faster accomplishes this sense of movement

Animation Type

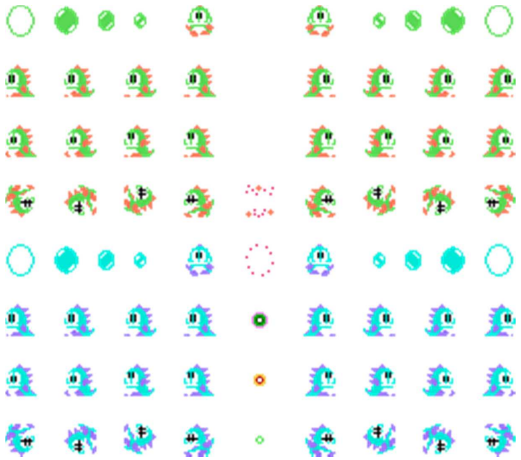
- Linear Animation



- Nonlinear Animation



Source of Animation



 More image can be found from opengameart.org

Creating the Game World