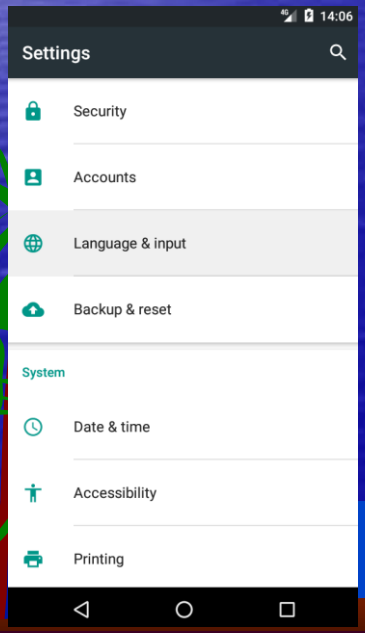
Android

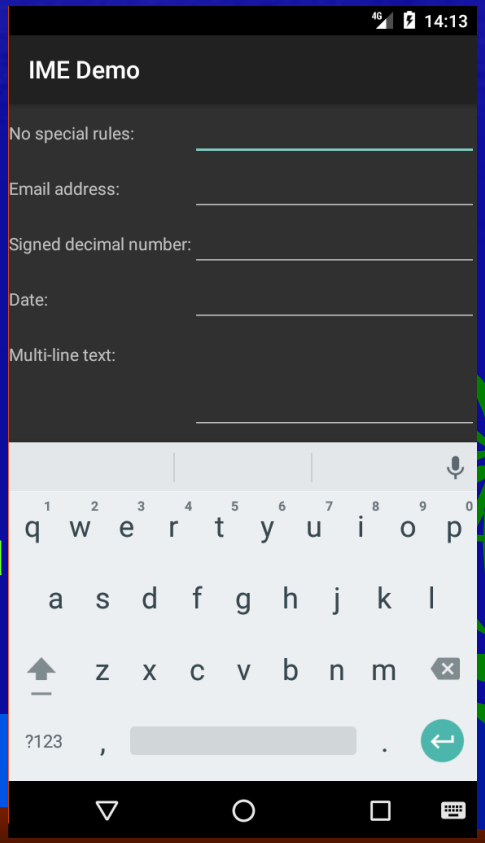
Soft Keyboard
Graphics
and Media

Soft keyboards

- Devices can have hard keyboards or only a directional pad (arrows plus select)
  - But most don't have keyboards
  - All have soft keyboards controlled by the IME (the input method editor)
  - Many of the soft keyboard properties can be set from the device 'Settings'



Device settings

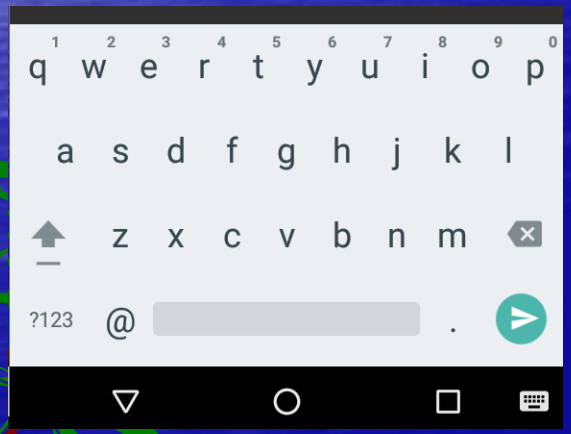


The general soft keyboard

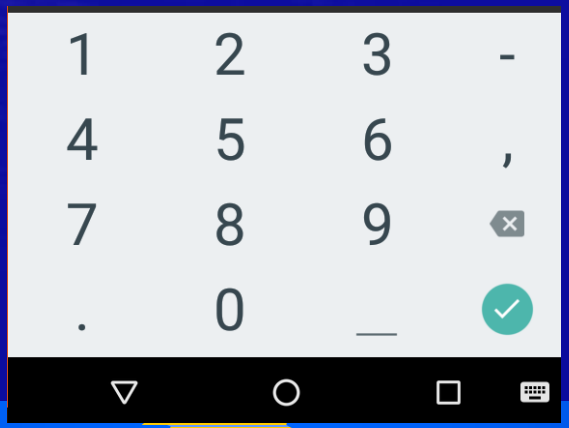
Tailoring the soft keyboard

- EditText views can modify the keyboard
  - Using the attribute android:inputType
    - allows different keys (i.e numeric, email, …)
  - Using the attribute android:imeOptions
    - allows different bottom-right keys instead of ‘return’
      - Examples: Next, Send, Done, …

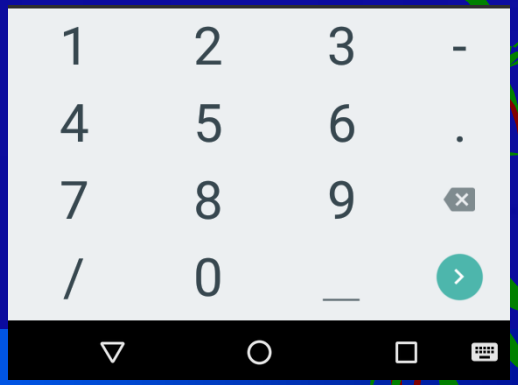
Email and send button



Numeric and done



Date and next



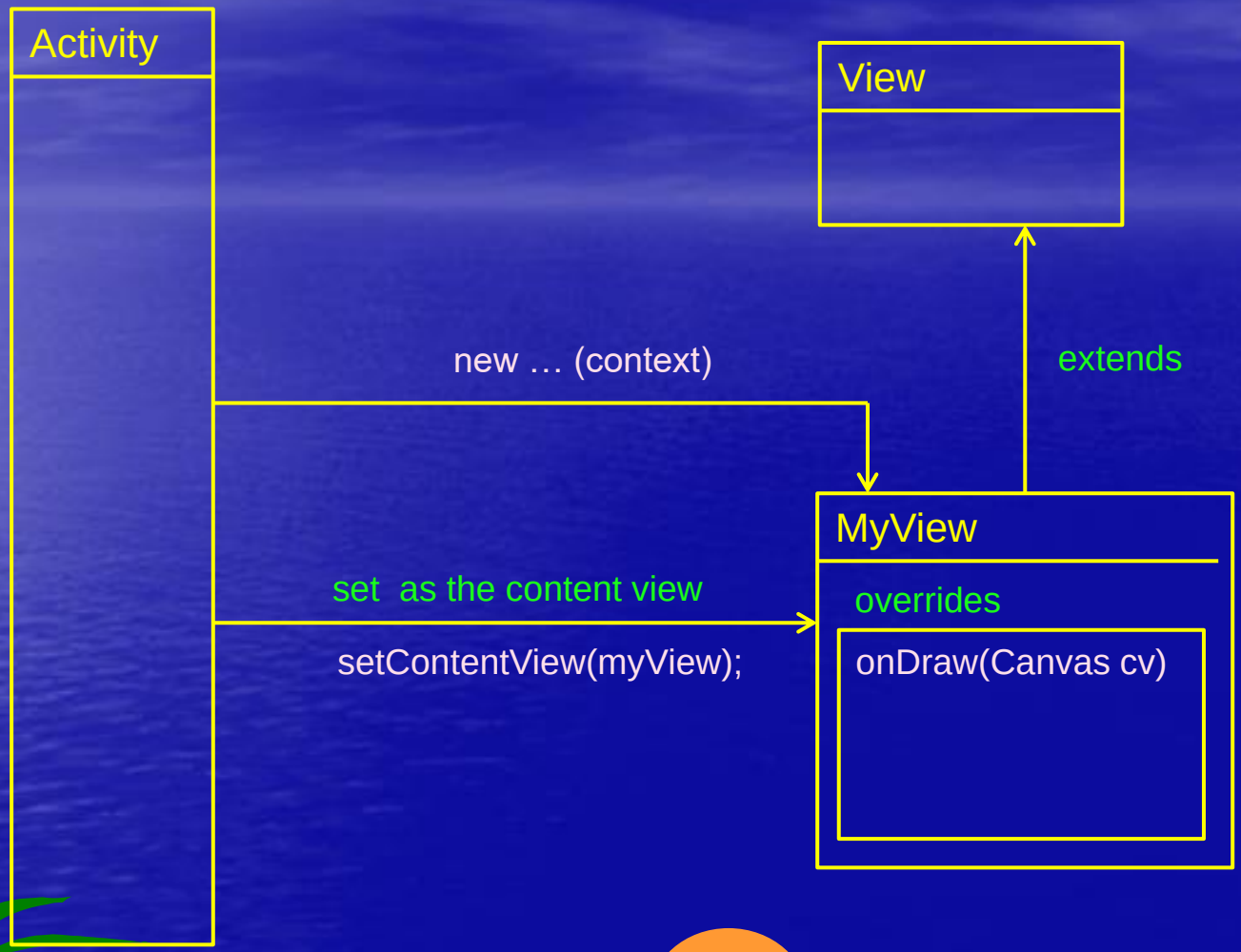
Action events

- Pressing the bottom-right key raises the EditorAction event
  - A listener can be defined in EditText views with
    - setOnEditorActionListener()
- You can dismiss the keyboard in the handler
  - By default, the Done key does that
  - Or use the code in the handler:

```
InputMethodManager mgr = (InputMethodManager) getSystemService(INPUT_METHOD_SERVICE);
mgr.hideSoftInputFromWindow(view.getWindowToken(), 0);
```

the EditText that has the focus (passed as a parameter to the handler)

2D graphics on the screen



2D graphics on the screen

- The Canvas instance defines a lot of primitives
  - draw...()
  - They need an instance of Paint
  - Paint defines the characteristics of the drawings, like color, line style and width, fonts and sizes, etc
- Many geometric shapes are defined through a Path instance
  - Paths go to the screen with canvas.drawPath()
- Other graphic elements are Drawable instances
  - Bitmaps, Shapes, NinePatches, etc
- Some graphic elements can be defined in xml resources and directly used or 'inflated'
  - Colors, Gradients, Shapes, …

Full custom Views

- Full custom Views need to override several methods from the View class
  - They can be used in XML layouts
  - Parameters from the layout are passed in the constructor
  - You can create your own event listeners and property accessors and modifiers
  - You should override the onMeasure() method for proper behavior when this View is integrated inside a layout
  - You should override onDraw() with your customized drawing, based on this View properties

A small example

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

background.xml on res/drawable

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/
                                              res/android">
  <gradient
    android:startColor="#FFFFFF"
    android:endColor="#808080"
    android:angle="270" />
</shape>
```


```
public class GraphicsView extends View {
    private static final String QUOTE = "Now is the time for all " +
                "good men to come to the aid of their country.";
    private final Path circle;
    private final Paint cPaint;
    private final Paint tPaint;

    public GraphicsView(Context context) {
      super(context);
      circle = new Path();
      circle.addCircle(150, 150, 100, Direction.CW);
      cPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
      cPaint.setStyle(Paint.Style.STROKE);
      cPaint.setColor(Color.LTGRAY);
      cPaint.setStrokeWidth(3);
      tPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
      tPaint.setStyle(Paint.Style.FILL_AND_STROKE);
      tPaint.setColor(Color.BLACK);
      tPaint.setTextSize(20f);
      setBackgroundResource(R.drawable.background);
  }

    @Override
    protected void onDraw(Canvas canvas) {
      canvas.drawPath(circle, cPaint);
      canvas.drawTextOnPath(QUOTE, circle, 0, 20, tPaint);
    }
}
```

Playing audio

- The Android framework encapsulates a complex media player
  - Can be used through the framework class MediaPlayer
  - It can work asynchronously (playing independently of the application)
  - It works as a state transition machine object
  - Supports a lot of audio formats
    - WAV, AAC, MP3, WMA, AMR (speech), OGG, MIDI
  - For a very simple operation call in order
    - release() (if the object of the MediaPalyer is not null)
    - create() (specifying a resource ID (in res/raw) or a URI)
    - start() (to start playing; returns immediately)

Example

```
public class Audio extends Activity {
   private MediaPlayer mp;

   @Override
   public void onCreate(Bundle savedInstanceState)  {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      setVolumeControlStream(
            AudioManager.STREAM_MUSIC);
   }
   . . .
```

```
<LinearLayout xmlns:android=
      "http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent" >
   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Press the F key"
   />
</LinearLayout>
```

```
. . .
@Override
public boolean onKeyDown(int keyCode, KeyEvent event)  {
   int resId;
   switch (keyCode) {
      case KeyEvent.KEYCODE_F:
         resId = R.raw.f;
         break;
      default:
         return super.onKeyDown(keyCode, event);
   }

 // Release any resources from previous MediaPlayer
   if (mp != null) {
      mp.release();
   }

// Create a new MediaPlayer to play this sound
   mp = MediaPlayer.create(this, resId);
   mp.start();

// Indicate this key was handled
   return true;
}
}
```

Playing video

- A video inside a file accessible to your application can be played within a VideoView
  - Formats supported include
    - MP4, H.263 (3GP), H.264 (AVC)
  - Inform the VideoView about the video file path with setVideoPath()
  - Start playing with the start() method

Example

```
. . .
<FrameLayout
   xmlns:android=
      "http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent" >
   <VideoView
      android:id="@+id/video"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:layout_gravity="center" />
</FrameLayout>
```

```
public class Video extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      // Fill view from resource
      setContentView(R.layout.main);
      VideoView video = (VideoView) findViewById(R.id.video);

      // Load and start the movie
      video.setVideoPath("/mnt/sdcard/samplevideo.3gp" );
      video.start();
   }
}
```

Manifest file:

```
. . .
<activity android:name=".Video"
   android:label="@string/app_name"
   android:theme=
      "@android:style/Theme.NoTitleBar.Fullscreen" >
   . . .
```

Video playing in landscape

Camera in preview mode

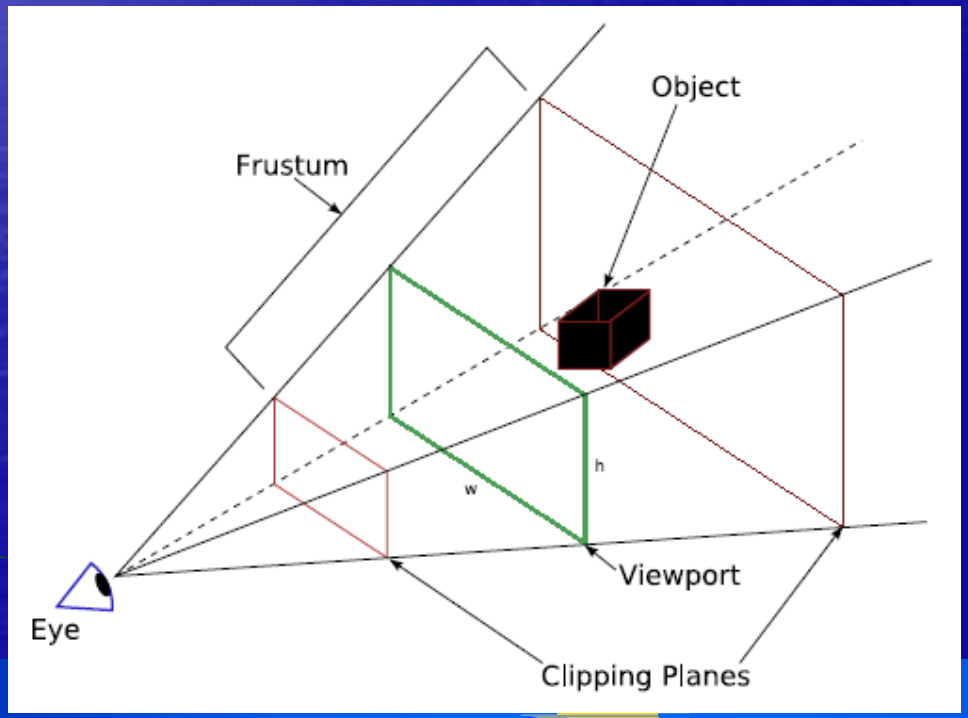
To display video directly from the camera we need a SurfaceView in an Activity layout

We need also to orchestrate the camera activation with that SurfaceView and the Activity life-cycle

0. Put a SurfaceView in the Activity layout
1. [In onCreate()]
   get the SurfaceView from the layout (findViewById())
   get a SurfaceHolder from the SurfaceView (save it on variable)
   add a SurfaceHolder.Callback object (with the callbacks) to the SurfaceHolder
2. [In onResume()]
   open the Camera (static open() method) and save it
   if the camera was already configured go to 4.
3. [In the surfaceChanged callback (inside the SurfaceHolder.Callback object)]
   setPreviewDisplay( )
   -- configure the camera
      getParameters()
      -- modify some Parameters
      setParameters()
4. startPreview() (we see the preview in the screen)
5. [In onPause()]
   stopPreview()
   release the Camera (release())

3D graphics in Android

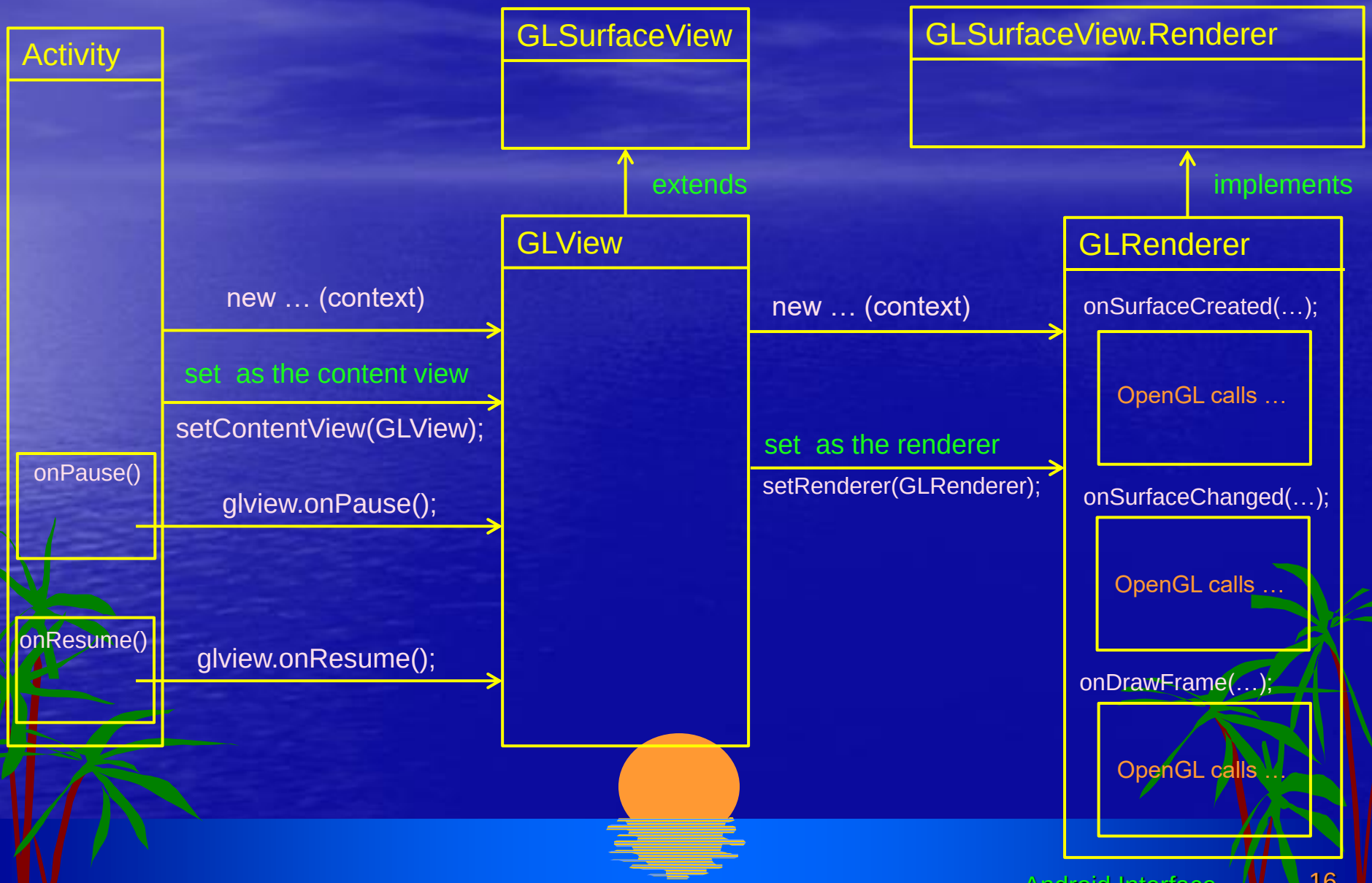
- 3D graphics are the projection of objects and light on a plane
  - The plane is the viewport and is mapped to the screen
  - The piece of space projected on the viewport is the view frustum (a piece of the pyramidal field of view)



OpenGL

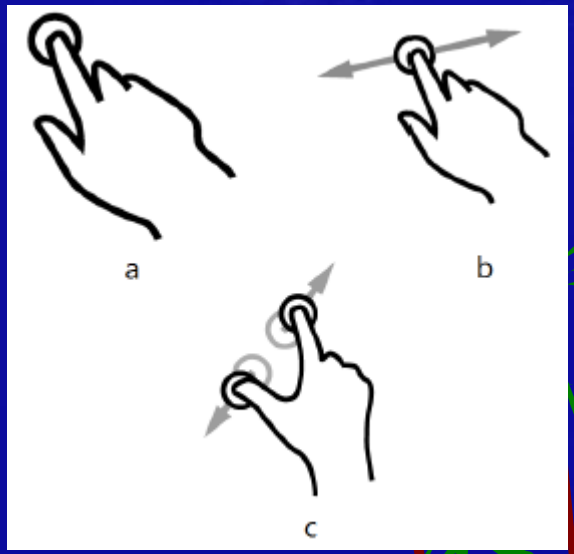
- OpenGL is a big library for 3D graphics programming
  - Independent of graphics hardware
  - Designed in 1992 for graphical workstations
  - There is a lighter version for mobile devices
    - OpenGL for Embedded Systems (or OpenGL ES)
    - A Java binding was standardized in JSR 239
    - Android started supporting OpenGL ES v 1.0 and some of v 1.1
      - After Android 2.2 OpenGL ES other versions were also supported but with some incompatible programming interfaces
- For using OpenGL ES in Android we use a special view derived from GLSurfaceView

OpenGL surface in Android



Touch events

- Many Android devices have only as input the touch screen and gestures
  - Many of the events generated by touch are transformed in high level ones like:
    - click, long click, list item select, key, …
  - But we can intercept them at a lower level using the OnTouchListener (and its onTouch() method)
    - The View and most of its subclasses generate onTouch events
    - Registered with setOnTouchListener()
    - When the listener is called it receives the View that caused it and a MotionEvent instance describing it



MotionEvent event

- MotionEvent objects provide information about the touch
  - getAction() returns in the lower 8 bits a code for the action: DOWN, UP, MOVE, OUTSIDE, …
  - In the higher 8 bits it gives a ‘finger’ number starting with 0 (in and after Android 2.2 multitouch is supported)
  - getPointerCount() returns the number of active ‘fingers’
  - getPointerId(i), getX(i), getY(i) allows us to extract the number and position of each active ‘finger’

Example

```
private void dumpEvent(MotionEvent event) {
   String names[] = { "DOWN" , "UP" , "MOVE" , "CANCEL" , "OUTSIDE" ,
                      "POINTER_DOWN" , "POINTER_UP" , "7?" , "8?" , "9?" };
   StringBuilder sb = new StringBuilder();
   int action = event.getAction();
   int actionCode = action & MotionEvent.ACTION_MASK;
   sb.append("event ACTION_" ).append(names[actionCode]);
   if (actionCode == MotionEvent.ACTION_POINTER_DOWN
                      || actionCode == MotionEvent.ACTION_POINTER_UP) {
      sb.append("(pid " ).append(action >> MotionEvent.ACTION_POINTER_ID_SHIFT);
      sb.append(")" );
   }
   sb.append("[" );
   for (int i = 0; i < event.getPointerCount(); i++) {
      sb.append("#" ).append(i);
      sb.append("(pid " ).append(
                                 event.getPointerId(i));
      sb.append(")=" ).append((int) event.getX(i));
      sb.append("," ).append((int) event.getY(i));
      if (i + 1 < event.getPointerCount())
         sb.append(";" );
   }
   sb.append("]" );
   Log.d(TAG, sb.toString());
}
```

Log touch events

Results

```
event ACTION_DOWN[#0(pid 0)=135,179]
event ACTION_MOVE[#0(pid 0)=135,184]
event ACTION_MOVE[#0(pid 0)=144,205]
event ACTION_MOVE[#0(pid 0)=152,227]
event ACTION_POINTER_DOWN(pid 1)[#0(pid 0)=153,230;#1(pid 1)=380,538]
event ACTION_MOVE[#0(pid 0)=153,231;#1(pid 1)=380,538]
event ACTION_MOVE[#0(pid 0)=155,236;#1(pid 1)=364,512]
event ACTION_MOVE[#0(pid 0)=157,240;#1(pid 1)=350,498]
event ACTION_MOVE[#0(pid 0)=158,245;#1(pid 1)=343,494]
event ACTION_POINTER_UP(pid 0)[#0(pid 0)=158,247;#1(pid 1)=336,484]
event ACTION_MOVE[#0(pid 1)=334,481]
event ACTION_MOVE[#0(pid 1)=328,472]
event ACTION_UP[#0(pid 1)=327,471]
```

Higher level gestures

- The onTouch listener can pass the MotionEvent data to gesture detectors (Android has two)
  - GestureDetector
    - Can detect and trigger events corresponding to one finger gestures
      - Down, Fling, LongPress, Scroll, ShowPress, SingleTap, DoubleTap
  - ScaleGestureDetector
    - Detects the pinch two finger gesture
      - Generates three events during the gesture: ScaleBegin, Scale, ScaleEnd