Android
VI – Graphics, Touch, Sensors

Stefan BORNHOFEN
EISTI

2D graphics

- Create a class extending View
- Override the onDraw() method
- Start drawing primitives

```
public class Draw2D extends View {
    Paint paint;

    public Draw2D(Context context) {
        super(context);
        paint = new Paint();
    }

    protected void onDraw(Canvas c){
        super.onDraw(c);
        paint.setStyle(Paint.Style.FILL);
        paint.setAntiAlias(true);
        paint.setColor(Color.WHITE);
        c.drawPaint(paint);
        paint.setColor(Color.BLUE);
        c.drawCircle(c.getWidth()/2, c.getHeight()/2, 50, paint);
    }
}
```

Animated 2D graphics

```
public class Draw2D extends View {
    Paint paint;

public Draw2D(Context context) {
    super(context);
    paint = new Paint();
 }

 protected void onDraw(Canvas c){
    super.onDraw(c);
    paint.setStyle(Paint.Style.FILL);
    paint.setAntiAlias(true);
    paint.setColor(Color.WHITE);
    c.drawPaint(paint);
    // draw here
    invalidate();
  }
}
```

Good enough for board games but not for animations because the view has tons of things to do

```
public class SurfaceDraw2D extends SurfaceView implements Runnable {
   Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG);
   Thread thread = null; SurfaceHolder sh;
   boolean paused = true;

   public SurfaceDraw2D(Context context) {
         super(context);
         sh = getHolder();
   }
   public void resume() { // called in Activity.onResume
         paused = false;
         thread = new Thread(this); thread.start();
   }
   public void pause() {// called in Activity.onPause
         paused = true;
         while (true) {
                    try { thread.join(); break; }
                    catch (InterruptedException e) { e.printStackTrace(); }
         }
         thread = null;
   }
    public void run() {
         while (!paused) {
                    if (!sh.getSurface().isValid()) continue;
                    Canvas c = sh.lockCanvas();
                    // PAINT HERE
                    sh.unlockCanvasAndPost(c);
         }
   }
}
```

Better solution: separate graphics thread and view thread

OpenGL ES

- 2D and 3D graphics APIs
- OpenGL ES 1.x: fixed rendering pipeline
- OpenGL ES 2.x: shaders (GLSL)
- GLSurfaceView and GLSurfaceView.Renderer

OpenGL

```
// OpenGL SurfaceView
public GLSurfaceView mGLSurfaceView;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    if (!isOGLES20Compatible()) {
      showOGLES20ErrorDialogBox();
        return;
    }

    // We don't use Layout. But you can. Create an OpenGLView:
    mGLSurfaceView = new GLSurfaceView(this);
    mGLSurfaceView.setEGLContextClientVersion(2);
    mGLSurfaceView.setRenderer(new GLES20Renderer(this));

    setContentView(mGLSurfaceView);
}
```

GLSurfaceView.Renderer

```
public static interface GLSurfaceView.Renderer
{
   void onSurfaceCreated(GL10 gl, EGLConfig config);
   void onSurfaceChanged(GL10 gl, int width, int height);
   void onDrawFrame(GL10 gl);
}
```

```
public class GLES20Renderer implements GLSurfaceView.Renderer {
     private Activity mActivity;
     GLES20Renderer(Activity activity) {
         mActivity = activity;
     }
     @Override
     public void onSurfaceCreated(GL10 gl, EGLConfig eglConfig) {
         gl.glClearColor(1.0f, 0.0f, 0.0f, 1.0f);
     }
     @Override
     public void onSurfaceChanged(GL10 gl, int width, int height) {
         gl.glViewport(0, 0, width, height);
         float ratio = (float) w / h; // adjust screen ratio
         gl.glMatrixMode(GL10.GL_PROJECTION);
         gl.glLoadIdentity();
         gl.glFrustumf(-ratio, ratio, -1, 1, 3, 7);
     }
     @Override
     public void onDrawFrame(GL10 gl) {
         gl.glClear( GLES20.GL_DEPTH_BUFFER_BIT | GLES20.GL_COLOR_BUFFER_BIT);
        // YOUR CODE HERE
     }
 }
```

Touch

```
public class MyTouchListener implements OnTouchListener {

  @Override
  public boolean onTouch(View v, MotionEvent event) {
       switch (event.getAction()) {
               case MotionEvent.ACTION_DOWN: …  // touch
               case MotionEvent.ACTION_UP:  …   // release
               case MotionEvent.ACTION_MOVE …   // drag
       }
       return true; // the event has been consumed
  }
}
```

Sensors: classification

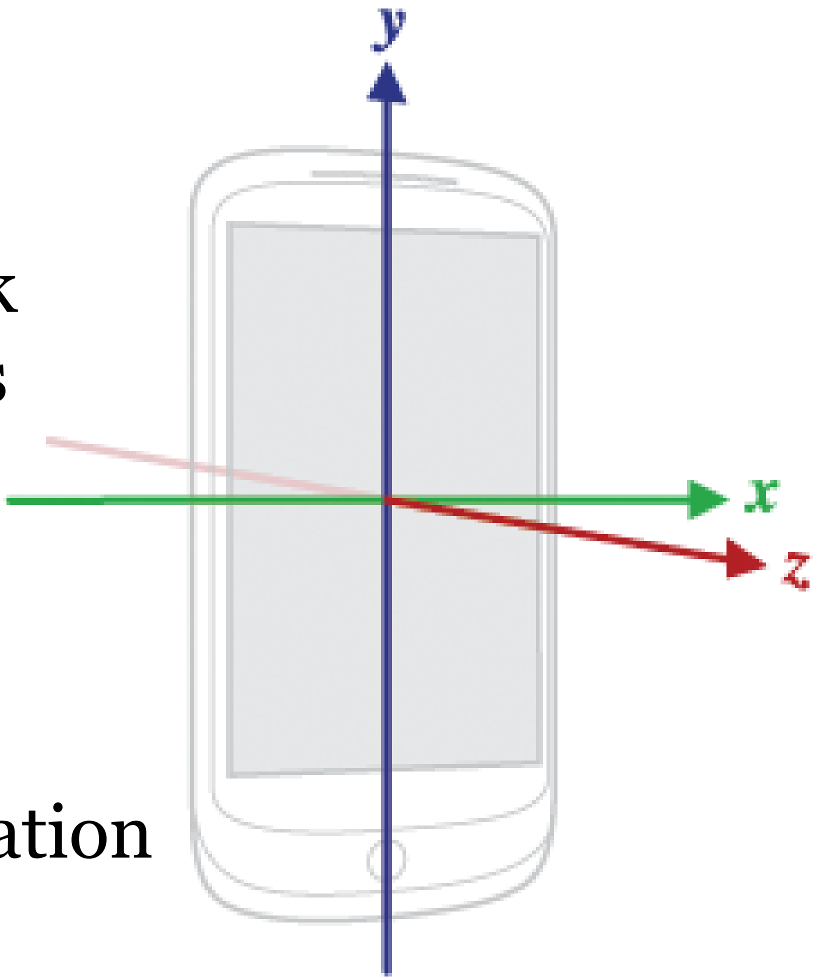
- **Motion sensors**: measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.
- **Position sensors**: measure the physical position of a device. This category includes orientation sensors and magnetometers.
- **Environmental sensors**: measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

Sensor framework

- Determine which sensors are available on a device.
- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
- Register and unregister sensor event listeners that monitor sensor changes.

Coordinates

- The sensor framework uses a standard 3-axis coordinate system to express data values
- The axes are **not swapped** when the device's screen orientation changes



Important classes

Class	Description
SensorManager	A class that gives access to the sensors available within the Android platform.
Sensor	Class representing a sensor. Use SensorManager .getSensorList(int) to get the list of available Sensors.
SensorEventListener	An interface used for receiving notifications from the SensorManager when sensor values have changed. An application implements this interface to monitor one or more sensors available in the hardware.
SensorEvent	This class represents a sensor event and holds information such as the sensor type (e.g., accelerometer, orientation, etc.), the time-stamp, accuracy and of course the sensor's data.

Initialization

```
private SensorManager sensorManager;
private Sensor sensor;


sensorManager = (SensorManager)
  getSystemService(Context.SENSOR_SERVICE);

 // use the accelerometer.
sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
  if (sensor != null) {
    // your game here
  }
  else {
    // Sorry, there are no accelerometers on your device.
    // You can't play this game.
  }
```

Registering the listener

- To avoid the unnecessary usage of battery you register your listener in the onResume() method and de-register it in the onPause() method.

```
public class SensorTest extends Activity implements SensorEventListener {
  private SensorManager sensorManager;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    ...
  }
  @Override
  public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) getAccelerometer(event);
  }
  private void getAccelerometer(SensorEvent event) {
    float x = event.values[0]; float y = event.values[1]; float z = event.values[2];
    ...
    }
  }
  @Override
  protected void onResume() {
    super.onResume();
    sensorManager.registerListener(this,
         sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
         SensorManager.SENSOR_DELAY_NORMAL);
  }
  @Override
  protected void onPause() {
    super.onPause();
    sensorManager.unregisterListener(this);
  }
}
```

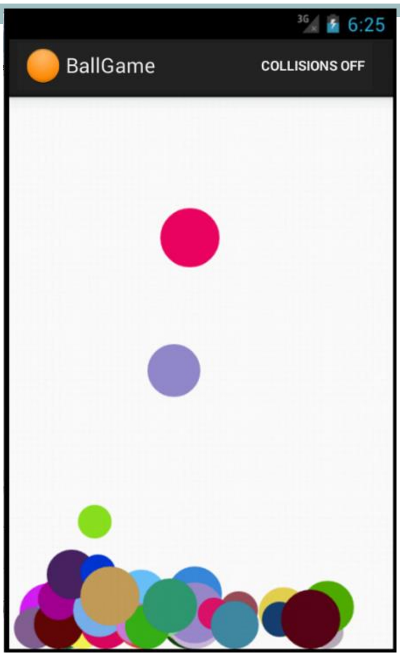
Exercise

Implement a 2D ball game (2p).

- Build an app with balls bouncing on the screen
- The balls are accelerated according to the orientation of the device
- Touching the screen creates a new ball
- Balls collide with the screen borders

Render a 3D cube (2p).

- The cube can be rotated by touch


6:25
BallGame
COLLISIONS OFF