MOBILE APPLICATION TO DISPLAY 3D MODELS

By Allen Hurtig
Academic Advisor: Professor Hongli Luo
Instructor: Professor Paul Lin

1

Outline of Presentation

- Executive Summary
- Introduction
- System Requirements
- Feasibility
- Design
- Issues and Resolutions
- Testing and Validation
- Conclusion
- Q & A
- Demo

2

Executive Summary

- Smart phones have become commonplace
- Capable of performing complex processes
- Excellent point for marketing goods and services
- Many manufacturers use apps as part catalogs

3

Introduction

- Oren Elliott Products is a manufacturer of couplings
- A competitor has a application capable of displaying 3D versions of their products
- The application was originally large in size but has now been reduced
- Oren Elliott Products suggested a mobile app to rival the competitor back in fall 2013

4

Introduction

- When resuming college, I proposed the application as my senior design project
- Motivation was to learn Android programming and OpenGL library
- Lead me into game development and 3D applications

5

Introduction

6

System Requirements

Project: Mobile Application to Display 3D Models
Revision: 2
Date: 18-Dec-15

Requirement Data			Verification Planning		
ID	Requirement Type	Requirement (Shall or Should statements)	Verification Method	Date Verific	Verification Report
1	Functional	The application shall use a GUI to take in user input.	Inspection		
2	Operational	The application shall display 3D models of mechanical couplings specifically Oldham, Block, and Jaw couplings.	Inspection		
3	Functional	The application shall use user derived measurements of various dimensions of couplings to display 3D models.	Demonstration		
4	Functional	The application shall draw 3D models to a screen using the OpenGL library.	Demonstration		
5	Physical	The application should use less than 100MB of RAM.	Test		
6	Physical	The application shall be a mobile application no larger than 100MB in size.	Inspection		
7	Physical	The application shall need a touch screen device with a resolution of 1920x1080 pixels.	Demonstration		
8	Environmental	The application shall run on Android OS only from version 4.4.4 on up.	Demonstration		
9	Environmental	The application shall run on a Samsung Galaxy S5 with a screen resolution of 1080x1920 pixels.	Demonstration		
10	Functional	The application shall use the metric system as the basis for measurements of the couplings.	Inspection		
11	Functional	The application should have the ability to switch between the metric system and the U.S. customary measurement system (Imperial units).	Demonstration		
12	Functional	The application shall allow the user to rotate the 3D model using the touch screen.	Inspection		
13	Functional	The application should allow the user to zoom in and out on the model.	Inspection		
14	Functional	The application should color code different parts of the coupling being displayed.	Inspection		
15	Performance	The application shall be able to resize its font depending on the screen size of the device the application is running on.	Inspection		
16	Performance	The application shall be able to save the user derived input of a selected coupling to a text file containing all the dimensions of said coupling.	Inspection		

7

Feasibility

- Research revealed the application to be quite possible
- Android Studio already includes OpenGL libraries
- Google and various websites have tutorials on Android and OpenGL programming

8

Design

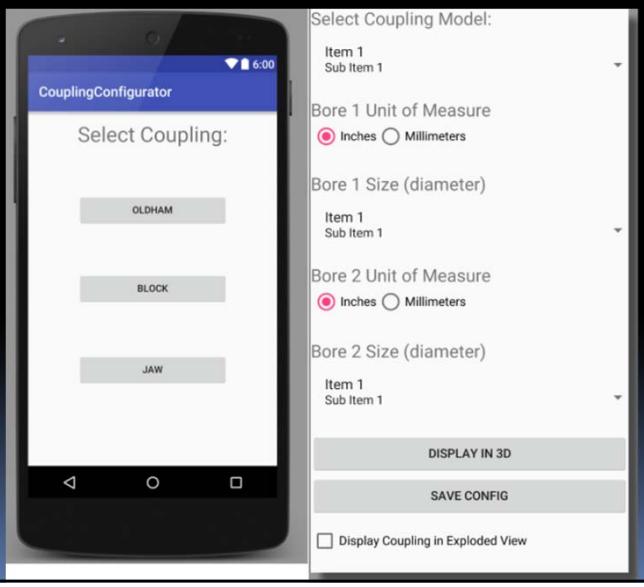
- Application uses three different "activities" as displays and GUIs
- Early OV-1 diagrams were main inspiration for GUI design
- Functionality was altered to simplify application

9

Design

10

Design



11

Design (MainActivity)

- Initializes singleton class called cType of CouplingType
- Sets the positions to use and type of coupling
- Sets the coupling midsection color and rotation for top cap
- Starts the ConfigActivity class

12

Design (MainActivity)

```
// Oldham button is pushed.

public void oldhamSelected(View view){

    cType.setCap1Positions(new ArrayList<float[]>());
    cType.addToCap1Positions(OldhamCap.oldhamCapPositions);
    cType.setCap1Vertices(OldhamCap.oldhamCapVertices);
    cType.addToCap2Positions(OldhamCap.oldhamCapPositions);
    cType.setCap2Vertices(OldhamCap.oldhamCapVertices);
    cType.setMidsectionPositions(OldhamMidsection.oldhamMidsectionPositions);
    cType.setMidsectionVertices(OldhamMidsection.oldhamMidsectionVertices);
    cType.setRedColor(1.0f);
    cType.setGreenColor(1.0f);
    cType.setBlueColor(0.0f);
    cType.setCapRotation(90.0f);
    cType.setCouplingType("Oldham");
    Intent i = new Intent(MainActivity.this, ConfigActivity.class);
    startActivity(i);
}
```

13

Design (CouplingType)

- Singleton class used to store data
- Data is shared between classes and activities
- Stores coupling type, measurements, colors
- References positions of coupling model to draw
- Uses getter and setter functions to store and retrieve data
- ArrayList variables have addition functions
- Has a function to return current instance

14

Design (CouplingType)

```
import java.util.ArrayList;
import java.util.List;

public class CouplingType{

    private static CouplingType instance;

    // Global variables.
    private String couplingType;
    private double couplingModel;
    private String boreOneUnitofMeasure;
    private double boreOneMeasurements;
    private String boreTwoUnitofMeasure;
    private double boreTwoMeasurements;
    private boolean isExplodedView;
    private List<float[]> cap1Positions = new ArrayList<float[]>();
    private List<float[]> cap2Positions = new ArrayList<float[]>();
    private float[] midsectionPositions;
    private int cap1Vertices;
    private int cap2Vertices;
    private int midsectonVertices;
    private float capRotation;
    private float redColor;
    private float greenColor;
    private float blueColor;
```

15

Design (CouplingType)

```
public List<float[]> getCap2Positions() { return cap2Positions; }

public float[] getMidsectionPositions() { return midsectionPositions; }

public int getCap1Vertices() { return cap1Vertices; }

public int getCap2Vertices() { return cap2Vertices; }

public int getMidsectonVertices() { return midsectonVertices; }

public float getCapRotation() { return capRotation; }

public float getRedColor() { return redColor; }

public float getGreenColor() { return greenColor; }

public float getBlueColor() { return blueColor; }

// Set value functions.

public void setCouplingType(String couplingType) { this.couplingType = couplingType; }

public void setCouplingModel(double couplingModel) { this.couplingModel = couplingModel; }

public void setBoreOneUnitofMeasure(String boreOneUnitofMeasure) {
    this.boreOneUnitofMeasure = boreOneUnitofMeasure;
```

16

Design (ConfigActivity)

- GUI to retrieve user input and set coupling parameters
- Uses spinners, radio buttons, buttons, and a check box
- Holds many local variables for configuration of display

17

Design (ConfigActivity)

```
public class ConfigActivity extends AppCompatActivity {

    // Local variables
    String[] oldhamModels={"12OC - 0.750\" OD", "14OC - 0.875\" OD", "16OC - 1.000\" OD",
            "21OC - 1.312\" OD", "26OC - 1.625\" OD"};
    String[] blockModels={"12BC - 0.750\" OD", "14BC - 0.875\" OD", "16BC - 1.000\" OD",
            "21BC - 1.312\" OD", "26BC - 1.625\" OD"};
    String[] jawModels={"12SC - 0.750\" OD", "14SC - 0.875\" OD", "16SC - 1.000\" OD",
            "21SC - 1.312\" OD", "26SC - 1.625\" OD"};
    double[] couplingSize={0.750, 0.875, 1.000, 1.312, 1.625};
    double[] boreInches={0.1875, 0.250, 0.3125, 0.375, 0.4375, 0.500, 0.5625, 0.6250, 0.6875, 0.75};
    double[] boreMM={4.0, 5.0, 6.0, 7.0, 8.0, 10.0, 12.0, 14.0, 15.0, 16.0, 18.0, 20.0};
    double[] percentsForBores={0.111, 0.138, 0.166, 0.194, 0.222, 0.25, 0.277, 0.305, 0.333, 0.388, 0.5};
    int[] inchesPosition={3, 4, 5, 7, 9};
    int[] mmPosition={4, 5, 6, 9, 11};
    String[]couplingInfo;
    CouplingType cType = CouplingType.getInstance();

    // Strings for saving units to file.
    String bore1units = "Inches";
    String bore2units = "Inches";

    // Declare the file attributes.
    String fileName = "Coupling_Configuration.txt";
    //String path = Environment.getExternalStorageDirectory().getAbsolutePath()+"/Coupling Configurator";
    String path = "/storage/emulated/0/Coupling Configurator";
    File myDirectory = new File(path);
```

18

Design (ConfigActivity)

```
// Function to handle when a new item is selected from the model spinner.
private void handleCouplingModelChanges(Boolean isEnglish, List<String> measurements, int position,
                                        ArrayAdapter<String> adapter, Spinner boreSpinner){
    // Clear data in measurements.
    measurements.clear();

    // Fill the spinner with the appropriate measurements starting at 0.1875".
    if (isEnglish){

        // minimum measurement is 0.1875".
        if (position < 2) {

            // Add the items to the spinner.
            for (int i = 0; i <= inchesPosition[position]; i++){
                measurements.add(String.valueOf(boreInches[i]));
            }
        }

        // Minimum measurement is 0.25".
        else{

            // Add the items to the spinner.
            for (int i = 1; i <= inchesPosition[position]; i++){
                measurements.add(String.valueOf(boreInches[i]));
            }
        }
    }
```

19

Design (ConfigActivity)

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[], int[] grantResults) {
    switch (requestCode) {
        case 1: {

            // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED) {

                saveConfig(couplingInfo);
            }
            return;
        }

    }
```

20

Design (ConfigActivity)

```
// Function to find the bore diameter.
private List<float[]> findBoreDiameterPositions(String units, List<float[]> knownBores,
                                                List<float[]> couplingCap, double measurement, double size)
{

    if (units.equals("Millimeters"))
    {
        // Convert measurement to english.
        measurement = measurement * 0.04;
    }

    for (int i = 0; i < knownBores.size(); i++)
    {
        // Get as close to the selected bore size as possible.
        if ((measurement / size) <= percentsForBores[i])
        {
            // Add necessary geometry.
            couplingCap.add(knownBores.get(i));
        }
    }

    // Find the size.
    if (size == 0.750)
    {
        // Do nothing since this is the minimum.
    }
    else if (size == 0.875)
    {
```

21

Design (ConfigActivity)

```
// Find the size.
if (size == 0.750)
{
    // Do nothing since this is the minimum.
}
else if (size == 0.875)
{
    couplingCap.add(CouplingSizes.M_14DPositions);
}
else if (size == 1.000)
{
    couplingCap.add(CouplingSizes.M_14DPositions);
    couplingCap.add(CouplingSizes.M_16DPositions);
}
else if (size == 1.312)
{
    couplingCap.add(CouplingSizes.M_14DPositions);
    couplingCap.add(CouplingSizes.M_16DPositions);
    couplingCap.add(CouplingSizes.M_21DPositions);
}
else
{
    // Largest cap selected.
    couplingCap.add(CouplingSizes.M_14DPositions);
    couplingCap.add(CouplingSizes.M_16DPositions);
    couplingCap.add(CouplingSizes.M_21DPositions);
    couplingCap.add(CouplingSizes.M_26DPositions);
}
```

22

Design (CouplingDisplay)

- Third and final display activity
- Initializes a GLSurfaceView and Renderer class
- GLSurfaceView acts as a "listener" for screen contact and display
- Render class performs heavy duty work

23

Design (CouplingDisplay)

```
public class CouplingDisplay extends Activity
{
    // Reference the surface view.
    private CouplingGLSurfaceView mGLSurfaceView;
    private CouplingRenderer mRenderer;

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

        // Initialize the surface view.
        mGLSurfaceView = new CouplingGLSurfaceView(this);
        setContentView(mGLSurfaceView);

        // Check if the system supports OpenGL ES 2.0.
        final ActivityManager activityManager = (ActivityManager) getSystemService(Context.ACTIVITY_SERVICE);
        final ConfigurationInfo configurationInfo = activityManager.getDeviceConfigurationInfo();
        final boolean supportsEs2 = configurationInfo.reqGlEsVersion >= 0x20000;

        if (supportsEs2)
        {
            // Request an OpenGL ES 2.0 compatible context.
            mGLSurfaceView.setEGLContextClientVersion(2);

            final DisplayMetrics displayMetrics = new DisplayMetrics();
            getWindowManager().getDefaultDisplay().getMetrics(displayMetrics);
```

24

Design (CouplingDisplay)

```
    if (supportsEs2)
    {
        // Request an OpenGL ES 2.0 compatible context.
        mGLSurfaceView.setEGLContextClientVersion(2);

        final DisplayMetrics displayMetrics = new DisplayMetrics();
        getWindowManager().getDefaultDisplay().getMetrics(displayMetrics);

        // Set the renderer to our demo renderer, defined below.
        mRenderer = new CouplingRenderer(this);
        mGLSurfaceView.setRenderer(mRenderer, displayMetrics.density);
    }
    else
    {
        // Won't display any models.
        return;
    }
}

@Override
protected void onResume()
{
    // The activity must call the GL surface view's onResume() on activity
    // onResume().
    super.onResume();
    mGLSurfaceView.onResume();
}
```

25

Design (CouplingGLSurfaceView)

- Class responsible for receiving touch commands
- Determines rotation factors and zoom factors
- Passes these values to the renderer class

26

Design (CouplingGLSurfaceView)

```
package com.example.allen.couplingconfigurator;

import android.content.Context;
import android.opengl.GLSurfaceView;
import android.util.AttributeSet;
import android.view.MotionEvent;

// GLSurfaceView class.
public class CouplingGLSurfaceView extends GLSurfaceView
{
    private CouplingRenderer mRenderer;

    // Offsets for touch events
    private float mPreviousX;
    private float mPreviousY;
    private float distanceOfFingers = 0;

    private float mDensity;

    public CouplingGLSurfaceView(Context context)
    {
        super(context);
    }
```

27

Desing (CouplingGLSurfaceView)

```
@Override
public boolean onTouchEvent(MotionEvent event)
{
    if (event != null)
    {
        float x = event.getX();
        float y = event.getY();
        float fingerDistance = 0;

        if (event.getAction() == MotionEvent.ACTION_MOVE)
        {
            // One finger for rotating.
            if (event.getPointerCount() == 1)
            {
                float deltaX = (x - mPreviousX) / mDensity / 2f;
                float deltaY = (y - mPreviousY) / mDensity / 2f;

                mRenderer.mDeltaX += deltaX;
                mRenderer.mDeltaY += deltaY;
            }

            // Two fingers for pinch zoom.
            if(event.getPointerCount() == 2)
            {
                if(distanceOfFingers == 0)
                {
```

28

Design (CouplingGLSurfaceView)

```
            // Two fingers for pinch zoom.
            if(event.getPointerCount() == 2)
            {
                if(distanceOfFingers == 0)
                {
                    distanceOfFingers = CalculateZoomFactor(event);
                }
                fingerDistance = CalculateZoomFactor(event);
                float newDistance = distanceOfFingers / fingerDistance;
                mRenderer.pinchZoom(newDistance);
                distanceOfFingers = fingerDistance;

            }
        }

        mPreviousX = x;
        mPreviousY = y;

        return true;
    }
    else
    {
        return super.onTouchEvent(event);
    }
}
```

29

Design (CouplingGLSurfaceView)

```
public void setRenderer(CouplingRenderer renderer, float density)
{
    mRenderer = renderer;
    mDensity = density;
    super.setRenderer(renderer);
}

// Calculate the new zoom distance.
private float CalculateZoomFactor(MotionEvent event)
{
    float x = event.getX(0) - event.getX(1);
    float y = event.getY(0) - event.getY(1);

    // Return the calculated zoom factor.
    return (float) Math.sqrt(x * x + y * y);
}
```

30

Design (CouplingRenderer)

- Class that uses OpenGL ES 2.0 to display 3D models
- Does all drawing of positions to screen

31

Design (CouplingRenderer)

```
+import ...

// Model rendering class.
public class CouplingRenderer implements GLSurfaceView.Renderer
{
    // Debug log variable.
    private static final String TAG = "ModelRenderer";

    // Matrices.
    private float[] mModelMatrix = new float[16];
    private float[] mViewMatrix = new float[16];
    private float[] mProjectionMatrix = new float[16];
    private float[] mMVPMatrix = new float[16];
    private final float[] mAccumulatedRotation = new float[16];
    private final float[] mCurrentRotation = new float[16];
    private float[] mTemporaryMatrix = new float[16];

    // Buffers for model data.
    private final FloatBuffer mMidsectionPositions;
    private final FloatBuffer mCap1Positions;
    private final FloatBuffer mCap2Positions;
    private final FloatBuffer mMidsectionColors;
    private final FloatBuffer mCap1Colors;
    private final FloatBuffer mCap2Colors;
```

32

Design (CouplingRenderer)

```
// Program and matrix handles.
private int mMVPMatrixHandle;
private int mMVMatrixHandle;
private int mColorHandle;
private int mPositionHandle;
private int mProgramHandle;

/** How many bytes per float. */
private final int mBytesPerFloat = 4;
private final int mPositionDataSize = 3;
private final int mColorDataSize = 4;

// Variables for rotation.
public volatile float mDeltaX;
public volatile float mDeltaY;

// Zoom factor.
public volatile float mZoomFactor = 1;

// Matrix view variables.
private volatile float ratio;
private volatile float left;
private volatile float right;
private volatile float bottom;
private volatile float top;
private volatile float near;
private volatile float far;
```

33

Design (CouplingRenderer)

```
@Override
public void onDrawFrame(GL10 glUnused)
{
    // Clear the screen and configure the setup of the view.
    GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT | GLES20.GL_DEPTH_BUFFER_BIT);

    // Set the per-vertex program.
    GLES20.glUseProgram(mProgramHandle);

    // Set program handles for model drawing.
    mMVPMatrixHandle = GLES20.glGetUniformLocation(mProgramHandle, "u_MVPMatrix");
    mMVMatrixHandle = GLES20.glGetUniformLocation(mProgramHandle, "u_MVMatrix");
    mColorHandle = GLES20.glGetAttribLocation(mProgramHandle, "a_Color");
    mPositionHandle = GLES20.glGetAttribLocation(mProgramHandle, "a_Position");

    // Draw a Model.

    // ----Midsection ----

    // Translate the model into the screen.
    Matrix.setIdentityM(mModelMatrix, 0);
    Matrix.translateM(mModelMatrix, 0, 0.0f, 0.0f, -7.0f);

    // Set a matrix that contains the current rotation.
    Matrix.setIdentityM(mCurrentRotation, 0);
    Matrix.rotateM(mCurrentRotation, 0, mDeltaX, 0.0f, 1.0f, 0.0f);
    Matrix.rotateM(mCurrentRotation, 0, mDeltaY, 1.0f, 0.0f, 0.0f);
    mDeltaX = 0.0f;
```

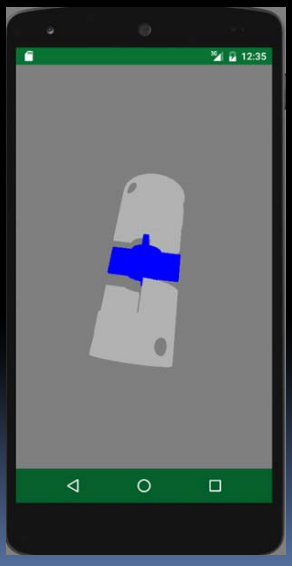
34

Design (CouplingRenderer)

```
// Set a matrix that contains the current rotation.
Matrix.setIdentityM(mCurrentRotation, 0);
Matrix.rotateM(mCurrentRotation, 0, mDeltaX, 0.0f, 1.0f, 0.0f);
Matrix.rotateM(mCurrentRotation, 0, mDeltaY, 1.0f, 0.0f, 0.0f);
mDeltaX = 0.0f;
mDeltaY = 0.0f;

// Multiply the current rotation by the accumulated rotation,
// and then set the accumulated rotation to the result.
Matrix.multiplyMM(mTemporaryMatrix, 0, mCurrentRotation, 0, mAccumulatedRotation, 0);
System.arraycopy(mTemporaryMatrix, 0, mAccumulatedRotation, 0, 16);

// Rotate the model taking the overall rotation into account.
Matrix.multiplyMM(mTemporaryMatrix, 0, mModelMatrix, 0, mAccumulatedRotation, 0);
System.arraycopy(mTemporaryMatrix, 0, mModelMatrix, 0, 16);

// Draw the midsection.
drawModel(mMidsectionPositions, mMidsectionColors, cType.getMidsectonVertices());
```

35

Design (CouplingRenderer)



36

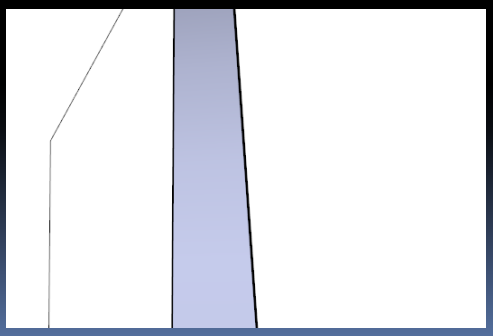
Issues and Resolutions

- Issue saving text files on Android 6.0 (Marshmallow) devices
- Fixed by asking for user permission
- Issue where models were not being displayed on screen
- Fixed by assigning color to the models
- Needed a way to convert models into Java code
- Wrote a separate program in VB.NET to perform conversion (2 to 3 months extra efforts)

37

Issues and Resoultions

- Issue designing 3D models in Sketchup
- Sketchup not the best modeling program
- Models had "tearing" (30 hours)



38

Testing and Validation

- Testing performed on virtual android device running 6.0 and Samsung Galaxy S5 running 4.4.4
- All project requirements were met when testing

39

Conclusion

- Project was very tough and labor intensive
- Learned to better time management
- Learned how to program in Android
- Learned 3D Modeling and 3D display
- Would do this project over again with a partner to even the workload

40

Questions?

Demo