Chapter 4

Artificial Horizons

An *artificial horizon* is defined by the *Oxford English Dictionary* as "a gyroscopic instrument or a fluid surface, typically one of mercury, used to provide the pilot of an aircraft with a horizontal reference plane for navigational measurement when the natural horizon is obscured." Artificial horizons have been used for navigational purposes long before augmented reality (AR) came into existence, and navigation still remains their primary use. They came into prominence when heads up displays came into mass use in planes, especially with military aircraft.

Artificial horizons are basically a horizontal reference line for navigators to use if the natural horizon is obscured. For all of us people who are obsessed with using AR in our apps, this is an important feature to be familiar with. It can be very useful when making navigational apps and even games.

It may be difficult to grasp the concept of a horizon that actually doesn't exist, but must be used to make all sorts of calculations that could affect the user in several ways. To get around this problem, we'll make a small sample app that doesn't implement AR, but shows you what an artificial horizon is and how it is implemented. After that, we'll make an AR app to use artificial horizons.

A Non-AR Demo App

In this app, we will have a compass with an artificial horizon indicator inside it. I will only be providing an explanation for the artificial horizon code, as the rest of it is not part of the book's subject.

R. Sood, *Pro Android Augmented Reality*

The XML

Let's get the little XML files out of the way first. We will need a `/res/layout/main.xml`, a `/res/values/strings.xml` and a `/res/values/colors.xml`.

Let's start with the `main.xml` file:

Listing 4-1. *main.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <com.paar.ch4nonardemo.HorizonView
    android:id="@+id/horizonView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
  />
</LinearLayout>
```

Nothing special here. We simply set the view of our `Activity` to a custom view, which we'll get around to making in a few minutes.

Let's take a look at `strings.xml` now:

Listing 4-2. *strings.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Pro Android AR 4 Non AR Demo</string>
  <string name="cardinal_north">N</string>
  <string name="cardinal_east">E</string>
  <string name="cardinal_south">S</string>
  <string name="cardinal_west">W</string>
</resources>
```

This file declared the string resources for four cardinals: N corresponds to North, E to East, S to South, and W to West.

Let's move on to the `colours.xml`:

Listing 4-3. *colours.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="text_color">#FFFF</color>
  <color name="background_color">#F000</color>
  <color name="marker_color">#FFFF</color>
```

```
  <color name="shadow_color">#7AAA</color>

  <color name="outer_border">#FF444444</color>
  <color name="inner_border_one">#FF323232</color>
  <color name="inner_border_two">#FF414141</color>
  <color name="inner_border">#FFFFFFFF</color>

  <color name="horizon_sky_from">#FFA52A2A</color>
  <color name="horizon_sky_to">#FFFFC125</color>
  <color name="horizon_ground_from">#FF5F9EA0</color>
  <color name="horizon_ground_to">#FF00008B</color>
</resources>
```

All of the colors are specified in either ARGB or AARRGGBB. They are used to add a bit of visual appeal to our little demo app. There is a slight difference in the "to" and "from" colors so that we can have a gradient in our final demo. The sky colors are shades of blue, and the ground colors are shades of orange.

The Java

Now we will create that custom view we mentioned in the main.xml.

Creating the View

Create a Java file called HorizonView.java in your main package (com.paar.ch4nonardemo on my end). Add the following global variables to it:

Listing 4-4. *HorizonView.java Global Variables*

```
public class HorizonView extends View {
  private enum CompassDirection { N, NNE, NE, ENE,
                                  E, ESE, SE, SSE,
                                  S, SSW, SW, WSW,
                                  W, WNW, NW, NNW }

  int[] borderGradientColors;
  float[] borderGradientPositions;

  int[] glassGradientColors;
  float[] glassGradientPositions;

  int skyHorizonColorFrom;
  int skyHorizonColorTo;
  int groundHorizonColorFrom;
  int groundHorizonColorTo;
```

```
  private Paint markerPaint;
  private Paint textPaint;
  private Paint circlePaint;
  private int textHeight;

  private float bearing;
  float pitch = 0;
  float roll = 0;
```

The names of the variables are reasonably good descriptions of their task. `CompassDirections` provides the strings we will use to create our 16-point compass. The ones with *Gradient*, *Color*, and *Paint* in their names are used in drawing the `View`, as is `textHeight`.

Getting and Setting the Bearing, Pitch, and Roll

Now add the following methods to the class:

Listing 4-5. *Bearing, Pitch, and Roll Methods*

```
public void setBearing(float _bearing) {
  bearing = _bearing;
}
public float getBearing() {
  return bearing;
}

public float getPitch() {
  return pitch;
}
public void setPitch(float pitch) {
  this.pitch = pitch;
}

public float getRoll() {
  return roll;
}
public void setRoll(float roll) {
  this.roll = roll;
}
```

These methods let us get and set the bearing, pitch and roll, which is later normalized and used to draw our view.

Calling and Initializing the Compass

Next, add the following three constructors to the class:

Listing 4-6. *HorizonView Constructors*

```
public HorizonView(Context context) {
  super(context);
  initCompassView();
}

public HorizonView(Context context, AttributeSet attrs) {
  super(context, attrs);
  initCompassView();
}

public HorizonView(Context context,
                   AttributeSet ats,
                   int defaultStyle) {
  super(context, ats, defaultStyle);
  initCompassView();
}
```

All three of the constructors end up calling `initCompassView()`, which does the main work in this class.

Speaking of `initCompassView()`, here's its code:

Listing 4-7. *initCompassView()*

```
protected void initCompassView() {
  setFocusable(true);
  Resources r = this.getResources();

  circlePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  circlePaint.setColor(R.color.background_color);
  circlePaint.setStrokeWidth(1);
  circlePaint.setStyle(Paint.Style.STROKE);

  textPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  textPaint.setColor(r.getColor(R.color.text_color));
  textPaint.setFakeBoldText(true);
  textPaint.setSubpixelText(true);
  textPaint.setTextAlign(Align.LEFT);

  textHeight = (int)textPaint.measureText("yY");

  markerPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  markerPaint.setColor(r.getColor(R.color.marker_color));
  markerPaint.setAlpha(200);
  markerPaint.setStrokeWidth(1);
  markerPaint.setStyle(Paint.Style.STROKE);
  markerPaint.setShadowLayer(2, 1, 1, r.getColor(R.color.shadow_color));
```

```
    borderGradientColors = new int[4];
    borderGradientPositions = new float[4];

    borderGradientColors[3] = r.getColor(R.color.outer_border);
    borderGradientColors[2] = r.getColor(R.color.inner_border_one);
    borderGradientColors[1] = r.getColor(R.color.inner_border_two);
    borderGradientColors[0] = r.getColor(R.color.inner_border);
    borderGradientPositions[3] = 0.0f;
    borderGradientPositions[2] = 1-0.03f;
    borderGradientPositions[1] = 1-0.06f;
    borderGradientPositions[0] = 1.0f;

    glassGradientColors = new int[5];
    glassGradientPositions = new float[5];

    int glassColor = 245;
    glassGradientColors[4] = Color.argb(65, glassColor,
                                        glassColor, glassColor);
    glassGradientColors[3] = Color.argb(100, glassColor,
                                        glassColor, glassColor);
    glassGradientColors[2] = Color.argb(50, glassColor,
                                        glassColor, glassColor);
    glassGradientColors[1] = Color.argb(0, glassColor,
                                        glassColor, glassColor);
    glassGradientColors[0] = Color.argb(0, glassColor,
                                        glassColor, glassColor);
    glassGradientPositions[4] = 1-0.0f;
    glassGradientPositions[3] = 1-0.06f;
    glassGradientPositions[2] = 1-0.10f;
    glassGradientPositions[1] = 1-0.20f;
    glassGradientPositions[0] = 1-1.0f;

    skyHorizonColorFrom = r.getColor(R.color.horizon_sky_from);
    skyHorizonColorTo = r.getColor(R.color.horizon_sky_to);

    groundHorizonColorFrom = r.getColor(R.color.horizon_ground_from);
    groundHorizonColorTo = r.getColor(R.color.horizon_ground_to);
}
```

In this method, we work with our colors to form proper gradients. We also assign values to some of the variables we declared in the beginning.

Calculating the Size of the Compass

Now add the following two methods to the class:

Listing 4-8. *onMeasure() and Measure()*

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

  int measuredWidth = measure(widthMeasureSpec);
  int measuredHeight = measure(heightMeasureSpec);

  int d = Math.min(measuredWidth, measuredHeight);

  setMeasuredDimension(d, d);
}

private int measure(int measureSpec) {
  int result = 0;

  int specMode = MeasureSpec.getMode(measureSpec);
  int specSize = MeasureSpec.getSize(measureSpec);

  if (specMode == MeasureSpec.UNSPECIFIED) {
    result = 200;
  } else {
    result = specSize;
  }
  return result;
}
```

These two methods allow us to measure the screen and let us decide how big we want our compass to be.

Drawing the Compass

Now finally add the `onDraw()` method to the class:

Listing 4-9. *onDraw()*

```
  @Override
  protected void onDraw(Canvas canvas) {
    float ringWidth = textHeight + 4;
    int height = getMeasuredHeight();
    int width =getMeasuredWidth();

    int px = width/2;
    int py = height/2;
    Point center = new Point(px, py);

    int radius = Math.min(px, py)-2;
```

```
    RectF boundingBox = new RectF(center.x - radius,
                                  center.y - radius,
                                  center.x + radius,
                                  center.y + radius);

    RectF innerBoundingBox = new RectF(center.x - radius + ringWidth,
                                       center.y - radius + ringWidth,
                                       center.x + radius - ringWidth,
                                       center.y + radius - ringWidth);

    float innerRadius = innerBoundingBox.height()/2;
    RadialGradient borderGradient = new RadialGradient(px, py, radius,
 borderGradientColors, borderGradientPositions, TileMode.CLAMP);

    Paint pgb = new Paint();
    pgb.setShader(borderGradient);

    Path outerRingPath = new Path();
    outerRingPath.addOval(boundingBox, Direction.CW);

    canvas.drawPath(outerRingPath, pgb);
    LinearGradient skyShader = new LinearGradient(center.x,
innerBoundingBox.top, center.x, innerBoundingBox.bottom,
 skyHorizonColorFrom, skyHorizonColorTo, TileMode.CLAMP);

    Paint skyPaint = new Paint();
    skyPaint.setShader(skyShader);

    LinearGradient groundShader = new LinearGradient(center.x,
innerBoundingBox.top, center.x, innerBoundingBox.bottom,
 groundHorizonColorFrom, groundHorizonColorTo, TileMode.CLAMP);

    Paint groundPaint = new Paint();
    groundPaint.setShader(groundShader);
    float tiltDegree = pitch;
    while (tiltDegree > 90 || tiltDegree < -90) {
      if (tiltDegree > 90) tiltDegree = -90 + (tiltDegree - 90);
        if (tiltDegree < -90) tiltDegree = 90 - (tiltDegree + 90);
    }

    float rollDegree = roll;
    while (rollDegree > 180 || rollDegree < -180) {
      if (rollDegree > 180) rollDegree = -180 + (rollDegree - 180);
       if (rollDegree < -180) rollDegree = 180 - (rollDegree + 180);
    }
    Path skyPath = new Path();
    skyPath.addArc(innerBoundingBox, -tiltDegree, (180 + (2 * tiltDegree)));
    canvas.rotate(-rollDegree, px, py);
    canvas.drawOval(innerBoundingBox, groundPaint);
    canvas.drawPath(skyPath, skyPaint);
```

```
canvas.drawPath(skyPath, markerPaint);
int markWidth = radius / 3;
int startX = center.x - markWidth;
int endX = center.x + markWidth;

double h = innerRadius*Math.cos(Math.toRadians(90-tiltDegree));
double justTiltY = center.y - h;

float pxPerDegree = (innerBoundingBox.height()/2)/45f;
for (int i = 90; i >= -90; i -= 10) {
  double ypos = justTiltY + i*pxPerDegree;

  if ((ypos < (innerBoundingBox.top + textHeight)) ||
      (ypos > innerBoundingBox.bottom - textHeight))
    continue;

  canvas.drawLine(startX, (float)ypos,
                  endX, (float)ypos,
                  markerPaint);
  int displayPos = (int)(tiltDegree - i);
  String displayString = String.valueOf(displayPos);
  float stringSizeWidth = textPaint.measureText(displayString);
  canvas.drawText(displayString,
                  (int)(center.x-stringSizeWidth/2),
                  (int)(ypos)+1,
                  textPaint);
}
markerPaint.setStrokeWidth(2);
canvas.drawLine(center.x - radius / 2,
                (float)justTiltY,
                center.x + radius / 2,
                (float)justTiltY,
                markerPaint);
markerPaint.setStrokeWidth(1);

Path rollArrow = new Path();
rollArrow.moveTo(center.x - 3, (int)innerBoundingBox.top + 14);
rollArrow.lineTo(center.x, (int)innerBoundingBox.top + 10);
rollArrow.moveTo(center.x + 3, innerBoundingBox.top + 14);
rollArrow.lineTo(center.x, innerBoundingBox.top + 10);
canvas.drawPath(rollArrow, markerPaint);
String rollText = String.valueOf(rollDegree);
double rollTextWidth = textPaint.measureText(rollText);
canvas.drawText(rollText,
                (float)(center.x - rollTextWidth / 2),
                innerBoundingBox.top + textHeight + 2,
                textPaint);
canvas.restore();
```

```
canvas.save();
canvas.rotate(180, center.x, center.y);
for (int i = -180; i < 180; i += 10) {
  if (i % 30 == 0) {
    String rollString = String.valueOf(i*-1);
    float rollStringWidth = textPaint.measureText(rollString);
    PointF rollStringCenter = new PointF(center.x-rollStringWidth /2,
                                         innerBoundingBox.top+1+textHeight);
    canvas.drawText(rollString,
                    rollStringCenter.x, rollStringCenter.y,
                    textPaint);
  }
  else {
    canvas.drawLine(center.x, (int)innerBoundingBox.top,
                    center.x, (int)innerBoundingBox.top + 5,
                    markerPaint);
  }

  canvas.rotate(10, center.x, center.y);
}
canvas.restore();
canvas.save();
canvas.rotate(-1*(bearing), px, py);

double increment = 22.5;

for (double i = 0; i < 360; i += increment) {
  CompassDirection cd = CompassDirection.values()
                        [(int)(i /  22.5)];
  String headString = cd.toString();

  float headStringWidth = textPaint.measureText(headString);
  PointF headStringCenter = new PointF(center.x - headStringWidth / 2,
                                       boundingBox.top + 1 + textHeight);

  if (i % increment == 0)
    canvas.drawText(headString,
                    headStringCenter.x, headStringCenter.y,
                    textPaint);
  else
    canvas.drawLine(center.x, (int)boundingBox.top,
                    center.x, (int)boundingBox.top + 3,
                    markerPaint);

    canvas.rotate((int)increment, center.x, center.y);
}
canvas.restore();
RadialGradient glassShader = new RadialGradient(px, py, (int)innerRadius,
glassGradientColors, glassGradientPositions, TileMode.CLAMP);
Paint glassPaint = new Paint();
```

```
    glassPaint.setShader(glassShader);

    canvas.drawOval(innerBoundingBox, glassPaint);
    canvas.drawOval(boundingBox, circlePaint);

    circlePaint.setStrokeWidth(2);
    canvas.drawOval(innerBoundingBox, circlePaint);

    canvas.restore();
  }
}
```

The `onDraw()` method draws the outer circles, clamps the pitch and roll values, colors the circles, takes care of adding the compass directions to the circle, rotates it when required, and draws the actual artificial horizon lines and moves them around.

In a nutshell, we create a circle with N, NE, and so on markers at 30-degree intervals. Inside the compass, we have an altimeter-like view that gives the position of the horizon relative to the way the phone is held.

Updating the Activity

We need to update this entire display from our main activity. For that to happen, we need to update `AHActivity.java`:

Listing 4-10. *AHActivity.java*

```
public class AHActivity extends Activity {
        float[] aValues = new float[3];
        float[] mValues = new float[3];
        HorizonView horizonView;
        SensorManager sensorManager;

        @Override
        public void onCreate(Bundle icicle) {
          super.onCreate(icicle);
          setContentView(R.layout.main);

          horizonView = (HorizonView)this.findViewById(R.id.horizonView);
          sensorManager =
(SensorManager)getSystemService(Context.SENSOR_SERVICE);
          updateOrientation(new float[] {0, 0, 0});
        }

    private void updateOrientation(float[] values) {
      if (horizonView!= null) {
        horizonView.setBearing(values[0]);
```

```
        horizonView.setPitch(values[1]);
            horizonView.setRoll(-values[2]);
            horizonView.invalidate();
          }
        }

    private float[] calculateOrientation() {
      float[] values = new float[3];
      float[] R = new float[9];
      float[] outR = new float[9];

      SensorManager.getRotationMatrix(R, null, aValues, mValues);
      SensorManager.remapCoordinateSystem(R,
                                          SensorManager.AXIS_X,
                                          SensorManager.AXIS_Z,
                                          outR);

      SensorManager.getOrientation(outR, values);

      values[0] = (float) Math.toDegrees(values[0]);
      values[1] = (float) Math.toDegrees(values[1]);
      values[2] = (float) Math.toDegrees(values[2]);

      return values;
    }

    private final SensorEventListener sensorEventListener = new
SensorEventListener() {
      public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
          aValues = event.values;
        if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD)
          mValues = event.values;

        updateOrientation(calculateOrientation());
      }

      public void onAccuracyChanged(Sensor sensor, int accuracy) {}
        };

        @Override
        protected void onResume() {
          super.onResume();

          Sensor accelerometer =
sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
          Sensor magField =
sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
```

```
        sensorManager.registerListener(sensorEventListener,
                                       accelerometer,
                                       SensorManager.SENSOR_DELAY_FASTEST);
        sensorManager.registerListener(sensorEventListener,
                                       magField,
                                       SensorManager.SENSOR_DELAY_FASTEST);
    }

    @Override
    protected void onStop() {
      sensorManager.unregisterListener(sensorEventListener);
      super.onStop();
    }
}
```

This is where that actual work happens. In the `onCreate()` method, we set the view to `main.xml`, get a reference to the `horizonView`, register a `SensorEventListener` and update the orientation to an ideal situation. The `updateOrientation()` method is responsible for passing new values to our view so that it can change appropriately. `calculateOrientation()` uses some of the provided methods from the SDK to accurately calculate the orientation from the raw values provided by the sensors. These methods provided by Android take care of a lot of complex math for us. You should be able to understand the `SensorEventListener`, `onResume()`, and `onStop()` easily enough. They do the same job they did in the previous chapters.

The Android Manifest

Finally, you should update your AndroidManifest to the following:

Listing 4-11. *AndroidManifest.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.paar.ch4nonardemo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="7" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".AHActivity"
            android:screenOrientation="portrait"
```

```
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Testing the Completed App

If you run the app now, you will get a very good idea of what an artificial horizon truly is. Figures 4-1 and 4-2 give you an idea about what the finished app looks like.

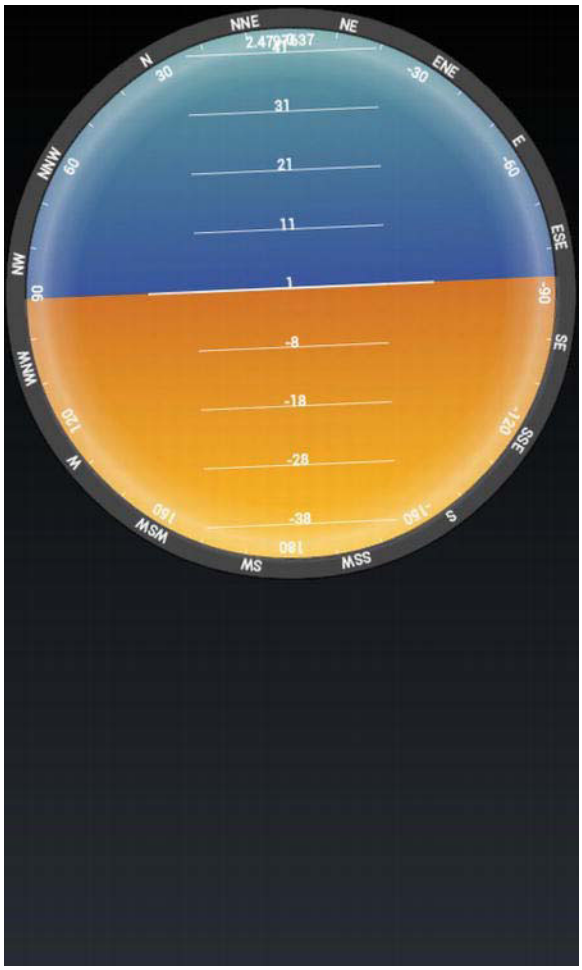


Figure 4-1. *The app when device is upright*

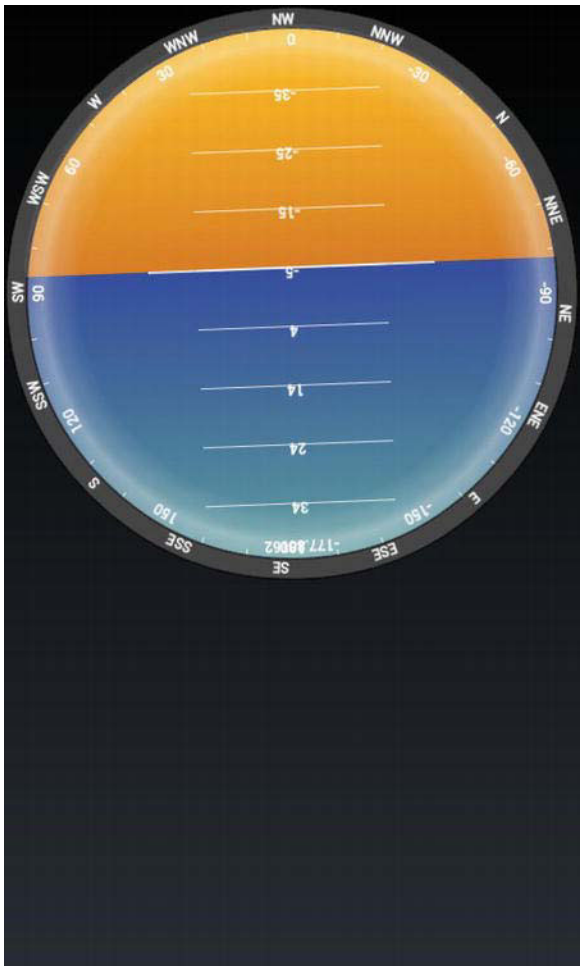


Figure 4-2. *The app when the device is upside down*

An AR Demo App

After going through and running the previous example, you should understand the concept of artificial horizons pretty well now. We will now be designing an app that does the following:

- Displays the live camera preview

- Displays a semitransparent version of `HorizonView` over the camera preview, colored like the movie versions in the aircrafts
- Tells you what your altitude will be in 5 minutes, assuming you continue at the current inclination

There are a few things to keep in mind before we start on the coding. Seeing as it is almost impossible for the user to hold the device perfectly still, the inclination will keep changing, which will cause the altitude in 5 minutes to change as well. To get around this, we will add a button that allows the user to update the altitude whenever they want.

> **NOTE:** On some devices, this app moves the artificial horizon in a clockwise and anticlockwise direction, instead of moving it up and down as in the non-AR demo. All the values are correct, except the display has a problem that can be fixed by running the app in portrait mode.

Setting Up the Project

To begin, create a new project. The one used as the example has the package name com.paar.ch4ardemo, and targets Android 2.1. As usual, you can change the name to whatever you like; just make sure to update any references in the example code as well. The screenshot in Figure 4-3 shows the project details.

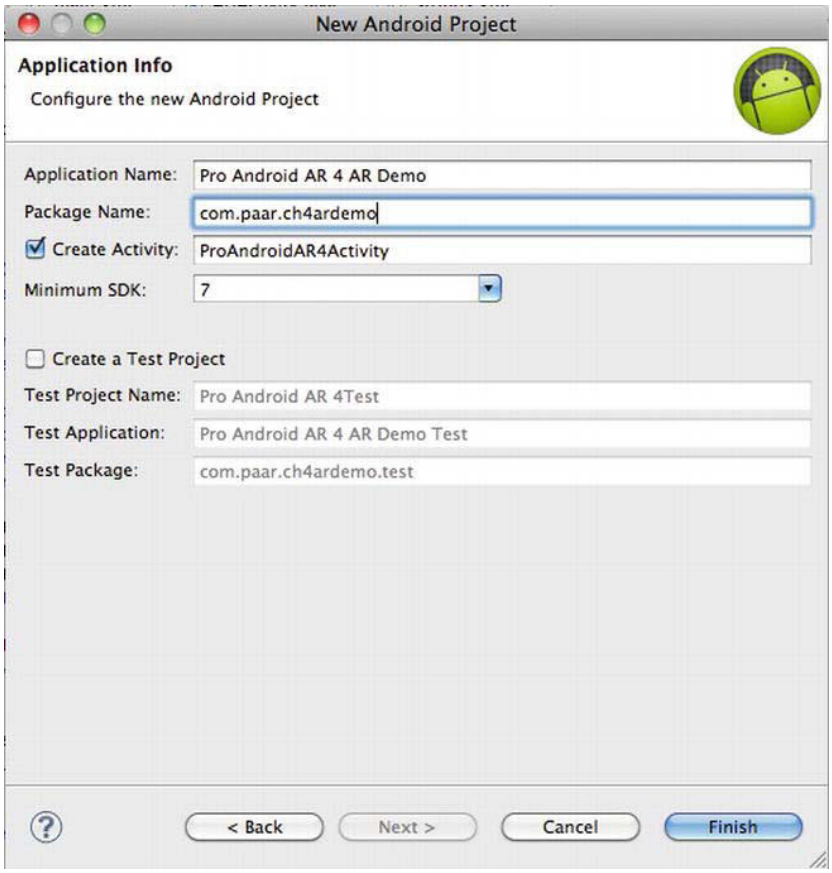


Figure 4-3. *The application details*

After you have created the new project, copy everything from the nonaugmented reality demo into this one. We will be building upon the previous project. Make sure to update the package name in the files where needed.

Updating the XML

To begin, we need to update the XML for our app. Our app currently has only four XML files: `AndroidManifest.xml`, `main.xml`, `colours.xml`, and `strings.xml`. We will just edit the ones copied over from the previous example instead of building new ones from scratch. The updated and new lines are in bold.

Updating AndroidManifest.xml to Access the GPS

Let's begin with `AndroidManifest.xml`. Because our updated app will require the altitude of the user, we will need to use the GPS to get it. The GPS requires the `ACCESS_FINE_LOCATION` permission to be declared in the manifest. In addition to the new permission, we must update that package name and change the orientation of the `Activity` to landscape.

Listing 4-12. *Updated AndroidManifest.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.paar.ch4ardemo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="7" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".AHActivity"
            android:screenOrientation="landscape"
android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
                android:configChanges = "keyboardHidden|orientation">
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-feature android:name="android.hardware.camera" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
</manifest>
```

Updating strings.xml to Display the Altitude

Next, let's take a look at `strings.xml`. We will be adding two new strings that will serve as the labels for the `Button` and `TextView`. We do not add a string for

the other TextView because it will be updated at runtime when the user clicks the button. Add the following two strings anywhere in your strings.xml file.

Listing 4-13. *Updated strings.xml*

```
<string name="altitudeButtonLabel">Update Altitude</string>
  <string name="altitudeLabel">Altitude in \n 5 minutes</string>
```

That little \n in the second string tells Android to print out the remainder of the string on a new line. We do this because on the smaller screen devices, the string might overlap with the button.

Updating colours.xml to Provide a Two-Color Display

Now let's update colours.xml. This time, we need only two colors, out of which only one is a visible color. In the previous example, we set a different color for the ground, sky, and so on. Doing so here will result in the dial of the meter covering our camera preview. However, using the ARGB color codes, we can make everything except the text transparent. Completely replace the contents of your colours.xml file with the following code.

Listing 4-14. *Updated colours.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="text_color">#F0F0</color>
  <color name="transparent_color">#0000</color>
</resources>
```

Updating main.xml to a RelativeLayout

Now we come to our final XML file—and the one with the maximum changes: main.xml. Previously, main.xml had only a LinearLayout with a HorizonView inside it. However, to allow for our AR overlaps, we will be replacing the LinearLayout with a RelativeLayout, and adding two TextViews and a Button, in addition to the HorizonView. Update the main.xml to the following code.

Listing 4-15. *Updated main.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
```

```
  <SurfaceView
          android:id="@+id/cameraPreview"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent" />
  <com.paar.ch4ardemo.HorizonView
      android:id="@+id/horizonView"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
  />
  <TextView
      android:id="@+id/altitudeLabel"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerVertical="true"
      android:layout_toRightOf="@id/horizonView"
      android:text="@string/altitudeLabel"
      android:textColor="#00FF00">
  </TextView>
  <TextView
      android:id="@+id/altitudeValue"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerVertical="true"
      android:layout_below="@id/altitudeLabel"
      android:layout_toRightOf="@id/horizonView"
      android:textColor="#00FF00">
  </TextView>
  <Button
      android:id="@+id/altitudeUpdateButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/altitudeButtonLabel"
      android:layout_centerVertical="true"
      android:layout_alignParentRight="true">
  </Button>
</RelativeLayout>
```

In this case, there are only five lines that were not modified. As usual, being a `RelativeLayout`, any mistake in the ids or position is fatal.

This takes care of the XML part of our app. Now we must move onto the Java files.

Updating the Java Files

The Java files have considerably more changes than the XML files, and some of the changes might not make sense at first. We'll take each change, one block of code at a time.

Updating HorizonView.java to make the compass transparent

Let's begin with `HorizonView.java`. We are modifying our code to make the dial transparent and work in landscape mode. Let's start by modifying `initCompassView()`. The only change we are making is replacing the old colors with the updated ones. The lines that have been modified are in bold.

Listing 4-16. *Updated initCompassView()*

```
protected void initCompassView() {
    setFocusable(true);
    Resources r = this.getResources();

    circlePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    circlePaint.setColor(R.color.transparent_color);
    circlePaint.setStrokeWidth(1);
    circlePaint.setStyle(Paint.Style.STROKE);

    textPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    textPaint.setColor(r.getColor(R.color.text_color));
    textPaint.setFakeBoldText(true);
    textPaint.setSubpixelText(true);
    textPaint.setTextAlign(Align.LEFT);

    textHeight = (int)textPaint.measureText("yY");

    markerPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    markerPaint.setColor(r.getColor(R.color.transparent_color));
    markerPaint.setAlpha(200);
    markerPaint.setStrokeWidth(1);
    markerPaint.setStyle(Paint.Style.STROKE);
    markerPaint.setShadowLayer(2, 1, 1, r.getColor(R.color.transparent_color));

    borderGradientColors = new int[4];
    borderGradientPositions = new float[4];

    borderGradientColors[3] = r.getColor(R.color.transparent_color);
    borderGradientColors[2] = r.getColor(R.color.transparent_color);
    borderGradientColors[1] = r.getColor(R.color.transparent_color);
```

```
    borderGradientColors[0] = r.getColor(R.color.transparent_color);
    borderGradientPositions[3] = 0.0f;
    borderGradientPositions[2] = 1-0.03f;
    borderGradientPositions[1] = 1-0.06f;
    borderGradientPositions[0] = 1.0f;

    glassGradientColors = new int[5];
    glassGradientPositions = new float[5];

    int glassColor = 245;
    glassGradientColors[4] = Color.argb(65, glassColor,
                                    glassColor, glassColor);
    glassGradientColors[3] = Color.argb(100, glassColor,
                                    glassColor, glassColor);
    glassGradientColors[2] = Color.argb(50, glassColor,
                                    glassColor, glassColor);
    glassGradientColors[1] = Color.argb(0, glassColor,
                                     glassColor, glassColor);
    glassGradientColors[0] = Color.argb(0, glassColor,
                                    glassColor, glassColor);
    glassGradientPositions[4] = 1-0.0f;
    glassGradientPositions[3] = 1-0.06f;
    glassGradientPositions[2] = 1-0.10f;
    glassGradientPositions[1] = 1-0.20f;
    glassGradientPositions[0] = 1-1.0f;

    skyHorizonColorFrom = r.getColor(R.color.transparent_color);
    skyHorizonColorTo = r.getColor(R.color.transparent_color);

    groundHorizonColorFrom = r.getColor(R.color.transparent_color);
    groundHorizonColorTo = r.getColor(R.color.transparent_color);
  }
```

Next, we need to update the onDraw() method to work with the landscape orientation. Because a good amount of the first part is unchanged, the entire method isn't given here. We are updating code right after the clamping of the pitch and roll takes place.

Listing 4-17. *Updated onDraw()*

```
//Cut Here
Path skyPath = new Path();
    skyPath.addArc(innerBoundingBox, -rollDegree, (180 + (2 * rollDegree)));
    canvas.rotate(-tiltDegree, px, py);
    canvas.drawOval(innerBoundingBox, groundPaint);
    canvas.drawPath(skyPath, skyPaint);
    canvas.drawPath(skyPath, markerPaint);
    int markWidth = radius / 3;
    int startX = center.x - markWidth;
    int endX = center.x + markWidth;
```

```
    Log.d("PAARV ", "Roll " + String.valueOf(rollDegree));
    Log.d("PAARV ", "Pitch " + String.valueOf(tiltDegree));


    double h = innerRadius*Math.cos(Math.toRadians(90-tiltDegree));
    double justTiltX = center.x - h;

    float pxPerDegree = (innerBoundingBox.height()/2)/45f;
    for (int i = 90; i >= -90; i -= 10) {
      double ypos = justTiltX + i*pxPerDegree;

      if ((ypos < (innerBoundingBox.top + textHeight)) ||
          (ypos > innerBoundingBox.bottom - textHeight))
        continue;

      canvas.drawLine(startX, (float)ypos,
                      endX, (float)ypos,
                      markerPaint);
      int displayPos = (int)(tiltDegree - i);
      String displayString = String.valueOf(displayPos);
      float stringSizeWidth = textPaint.measureText(displayString);
      canvas.drawText(displayString,
                      (int)(center.x-stringSizeWidth/2),
                      (int)(ypos)+1,
                      textPaint);
    }
    markerPaint.setStrokeWidth(2);
    canvas.drawLine(center.x - radius / 2,
                    (float)justTiltX,
                    center.x + radius / 2,
                    (float)justTiltX,
                    markerPaint);
    markerPaint.setStrokeWidth(1);
//Cut Here
```

These changes make our app look nice and transparent.

Updating the Activity to Access GPS, and Find and Display the Altitude

Now, we must move onto our final file, `AHActivity.java`. In this file, we will be adding GPS code, `TextView` and `Button` references, slightly modifying our Sensor code, and finally putting in a small algorithm to calculate our altitude after 5 minutes. We will be using trigonometry to find the change in altitude, so if yours is a little rusty, you might want to brush up on it quickly.

To begin, add the following variables to the top of your class.

Listing 4-18. *New Variable Declarations*

```
LocationManager locationManager;

Button updateAltitudeButton;
TextView altitudeValue;

double currentAltitude;
double pitch;
double newAltitude;
double changeInAltitude;
double thetaSin;
```

`locationManager` will, well, be our location manager. `updateAltitudeButton` and `altitudeValue` will hold references to their XML counterparts so that we can listen for clicks and update them. `currentAltitude`, `newAltitude`, and `changeInAltitude` will all be used to store values during the operation of our algorithm. The `pitch` variable will be storing the pitch, and `thetaSin` will be storing the sine of the pitch angle.

We will now update our `onCreate()` method to get the location service from Android, set the location listener, and set the `OnClickListener` for the button. Update it to the following code.

Listing 4-19. *Updated onCreate()*

```
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);

inPreview = false;

cameraPreview = (SurfaceView)findViewById(R.id.cameraPreview);
previewHolder = cameraPreview.getHolder();
previewHolder.addCallback(surfaceCallback);
previewHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

altitudeValue = (TextView) findViewById(R.id.altitudeValue);

updateAltitudeButton = (Button) findViewById(R.id.altitudeUpdateButton);
updateAltitudeButton.setOnClickListener(new OnClickListener() {

          public void onClick(View arg0) {
                  updateAltitude();
          }
```

```
});

locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 2000, 2,↩
 locationListener);

    horizonView = (HorizonView)this.findViewById(R.id.horizonView);
    sensorManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
    updateOrientation(new float[] {0, 0, 0});
  }
```

Right about now, Eclipse should be telling you that the `updateAltitude()` method and the `locationListener` don't exist. We'll fix that by creating them. Add the following `LocationListener` to any part of your class, outside of a method. If you're wondering why we have three unused methods, it's because a `LocationListener` must implement all four of the methods, even if they aren't used. Removing them will throw an error when compiling.

Listing 4-20. *LocationListener*

```
LocationListener locationListener = new LocationListener() {
        public void onLocationChanged(Location location) {
                currentAltitude = location.getAltitude();
        }

        public void onProviderDisabled(String arg0) {
                //Not Used
        }

        public void onProviderEnabled(String arg0) {
                //Not Used
        }

        public void onStatusChanged(String arg0, int arg1, Bundle arg2) {
                //Not Used
        }
    };
```

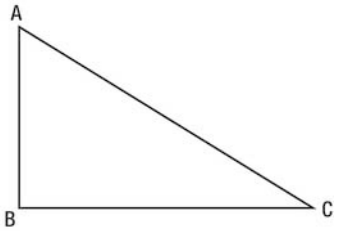
Before we move on to the `updateAltitude()` method, we will quickly add a line to the `calculateOrientation()` method so that the pitch variable isn't empty. Add the following right before the return statement.

Listing 4-21. *Ensuring the Pitch Variable isn't Empty in calculateOrientation()*

```
pitch = values[1];
```

Calculating the Altitude

Now that our pitch has a value, let's move to the `updateAltitude()` method. This method implements an algorithm to find the altitude of a person after 5 minutes, taking the current pitch as the angle at which they're moving up. We take the walking speed as 4.5 feet/second, which is the average speed of an adult. Using the speed and time, we can find out the distance traveled in 5 minutes. Then using trigonometry, we can find out the change in altitude from the distance traveled and the angle of inclination. We then add the change in altitude to the old altitude to get our updated altitude and show it in a `TextView`. If either the pitch or the current altitude is zero, the app asks the user to try again. See Figure 4-4 for a graphical explanation of the concept.



AC = distance moved (Speed * time)
AB = change in altitude
Angle C = Angle of inclination (pitch)
△ABC = right angled triangle
sin(c) = AB/AC
Hence, AB = AC * sin(c)

Figure 4-4. *A graphical representation of the algorithm*

Here's the code for `updateAltitude()`:

Listing 4-22. *Calculating and Displaying the Altitude*

```
public void updateAltitude() {
        int time = 300;
        float speed = 4.5f;

        double distanceMoved = (speed*time)*0.3048;
        if(pitch != 0 && currentAltitude != 0)
        {
                thetaSin = Math.sin(pitch);
                changeInAltitude = thetaSin * distanceMoved;
                newAltitude = currentAltitude + changeInAltitude;
                altitudeValue.setText(String.valueOf(newAltitude));
        }
        else
```

```
        {
                altitudeValue.setText("Try Again");
        }
}
```

And with that, we have finished the AR version of our example app.

Testing the Completed AR app

Take a look at the screenshots in Figures 4-5 and 4-6 to see how the app functions.

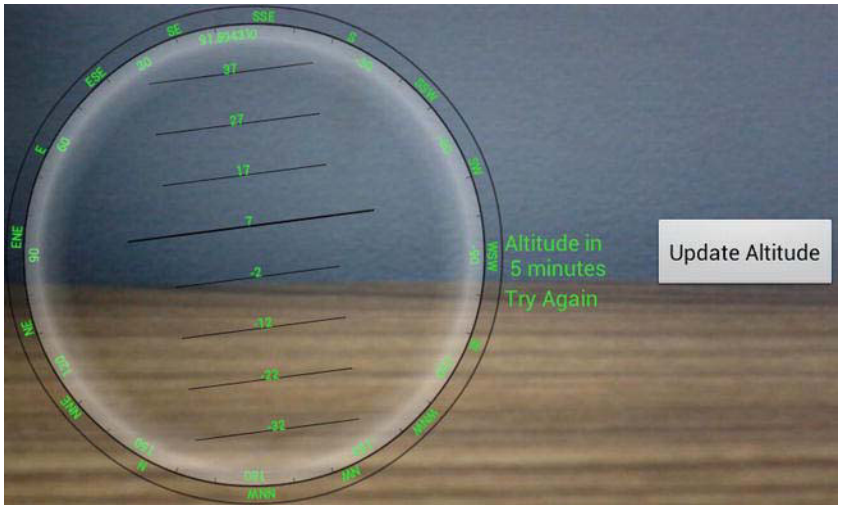


Figure 4-5. *The app running, with a try again message being displayed to the user*

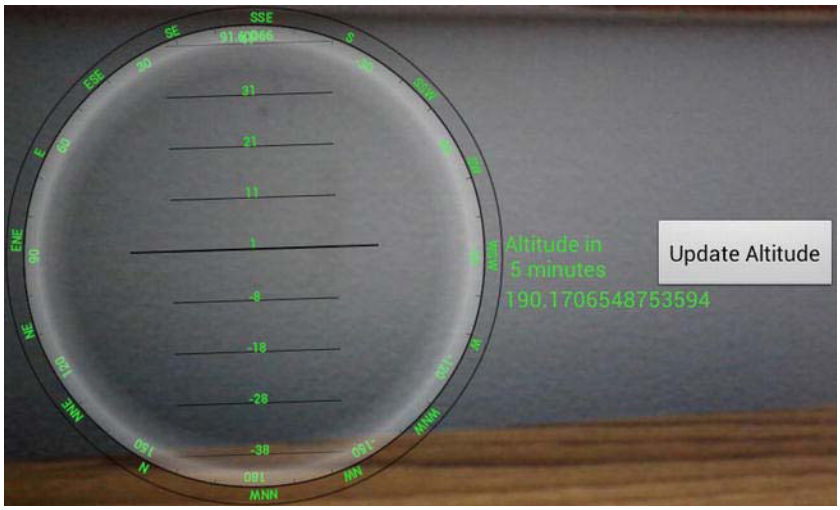


Figure 4-6. *The user is shown the altitude as well this time*

Summary

This chapter explored the concept of artificial horizons and how to create apps utilizing them. We devised an algorithm to find the change in altitude and implemented it in an example app. The apps given here are only an example of what you can do with artificial horizons. They are widely used in the military, especially the Air Force; and in cities, where the natural horizon is distorted due to height or is blocked by buildings.