Chapter 8

A 3D Augmented Reality Model Viewer

After completing the informational chapters and going through the previous two example apps, you should now be quite familiar with augmented reality (AR) on Android. This is the second-to-last example app, and the last that is in the realm of a normal, nongame app because the final example app is a game built using AR.

This app uses markers to function and is pretty straightforward. When launched, it will display a list of built-in objects to users to display on the marker or give them the option to select a custom object from the device's memory. The app accepts the objects in wavefront .obj files, along with their .mtl counterparts. If you are unfamiliar with these formats and wavefront in general, I recommend that you read up on it before continuing.

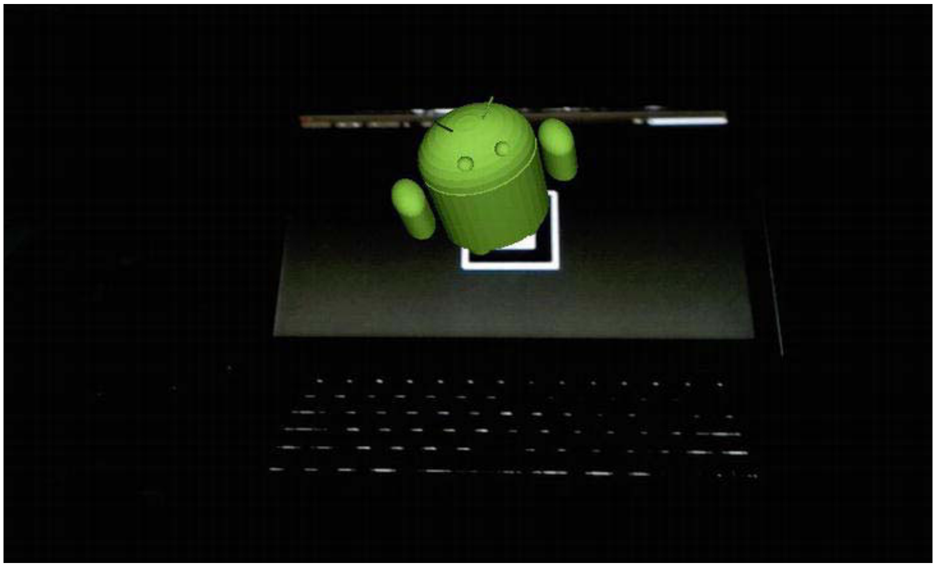


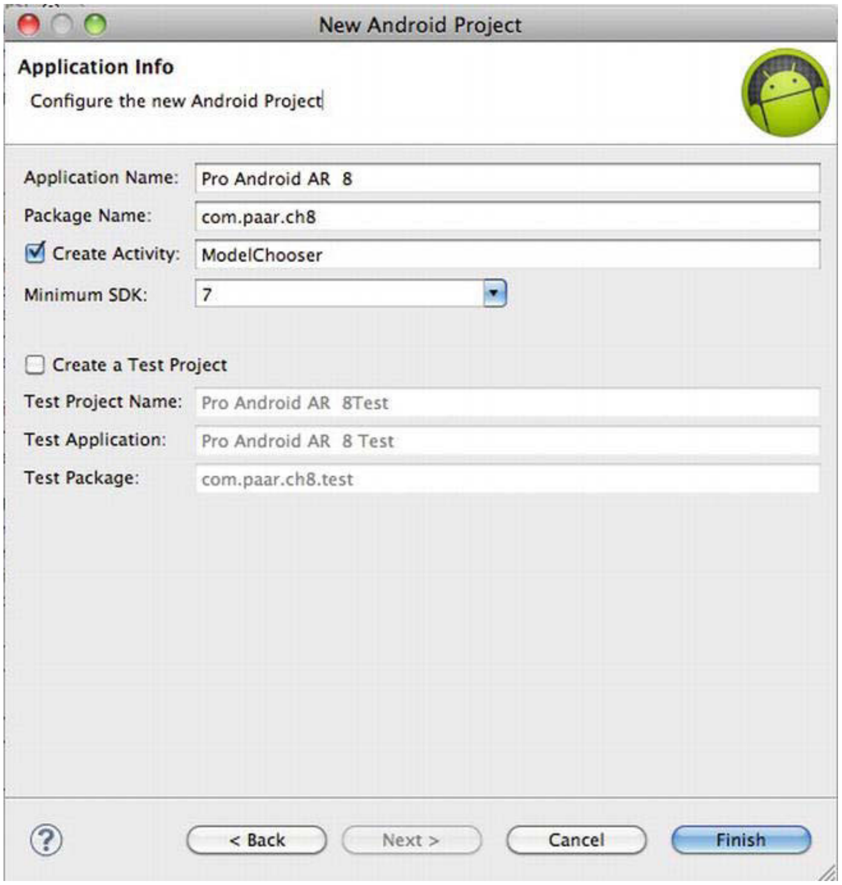
Figure 8-1. *The Android model being displayed.*

Key Features of this App

The following are the key features of this app:

- Allows users to view any of the preloaded models on a marker
- Allows users to view an external model that is located on the SD card, by using OI Filemanager to locate and select it
- Displays all models in 3D on the marker

Once more, start by creating a new project. This project does not extend any of the previous ones, so we'll be starting from scratch. We will have 22 Java files, 8 drawables, 4 layouts, 1 strings.xml, and 31 asset files. Figure 8-2 shows the details of the project.



Application Info

Configure the new Android Project

Application Name: Pro Android AR 8

Package Name: com.paar.ch8

[✓] Create Activity: ModelChooser

Minimum SDK: 7

[ ] Create a Test Project

Test Project Name: Pro Android AR 8Test

Test Application: Pro Android AR 8 Test

Test Package: com.paar.ch8.test

Figure 8-2. *The details of the project in this chapter.*

This project will be using AndAR as an external library to make our AR tasks easier. Marker recognition algorithms are difficult to implement, and this library is a working implementation that we can use safely within the scope of this book. You can obtain a copy of the AndAR library from http://code.google.com/p/andar/downloads/list, but it would be better if you downloaded it from this project's source on GitHub.com or Apress.com because future or older versions of AndAR may not be implemented the same way as the one used in this project.

The Manifest

To begin, here's the `AndroidManifest.xml` that declares all the permissions and activities in the app.

Listing 8-1. *AndroidManifest.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.paar.ch8"
    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=".ModelChooser" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

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

<activity android:exported="false"
    android:clearTaskOnLaunch="true"
    android:screenOrientation="landscape"
    android:icon="@drawable/ic_launcher"
    android:name=".ModelViewer">
</activity>
<activity android:exported="false"
    android:icon="@drawable/ic_launcher"
    android:name=".Instructions">
</activity>
            <activity android:label="@string/app_name"
            android:icon="@drawable/ic_launcher"
            android:name=".CheckFileManagerActivity">
        </activity>
    </application>

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.CAMERA"/>

<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

```
<supports-screens android:smallScreens="true"
                  android:normalScreens="true"
                  android:largeScreens="true"
                  android:anyDensity="true" />
</manifest>
```

In this file, we declare the four Activities in our app: ModelChooser, ModelViewer, Instructions and CheckFileManagerActivity. We then tell Android that we will be using the external storage and the camera, and ask for permission. We tell Android that we will be using the camera feature and its autofocus. Finally, we declare the screen sizes that our app will support.

Java Files

Let's begin the Java code by creating the file that will be our main activity.

Main Activity

In my case, this file is called `ModelChooser.java`, which displays when the user first starts the app. It has a list of the preloaded models that can be displayed, an option to load an external, user-provided model from the device memory, and a link to the help file.

onCreate() method

Let's start off its coding by making some changes to the `onCreate()` method of this file.

Listing 8-2. *onCreate() method in ModelChooser.java*

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    AssetManager am = getAssets();
    Vector<Item> models = new Vector<Item>();
    Item item = new Item();
    item.text = getResources().getString(R.string.choose_a_model);
    item.type = Item.TYPE_HEADER;
    models.add(item);

    try {
        String[] modelFiles = am.list("models");
        List<String> modelFilesList = Arrays.asList(modelFiles);
        for (int i = 0; i < modelFiles.length; i++) {
        String currFileName = modelFiles[i];
        if(currFileName.endsWith(".obj")) {
            item = new Item();
            String trimmedFileName =
currFileName.substring(0,currFileName.lastIndexOf(".obj"));
            item.text = trimmedFileName;
            models.add(item);
            if(modelFilesList.contains(trimmedFileName+".jpg")) {
            InputStream is = am.open("models/"+trimmedFileName+".jpg");
            item.icon=(BitmapFactory.decodeStream(is));
        } else if(modelFilesList.contains(trimmedFileName+".png")) {
            InputStream is = am.open("models/"+trimmedFileName+".png");
            item.icon=(BitmapFactory.decodeStream(is));
        }
    }
}
} catch (IOException e) {
    e.printStackTrace();
}
    item = new Item();
    item.text = getResources().getString(R.string.custom_model);
    item.type = Item.TYPE_HEADER;
    models.add(item);
    item = new Item();
    item.text = getResources().getString(R.string.choose_custom_model);
    item.icon = new Integer(R.drawable.open);
    models.add(item);
    item = new Item();
    item.text = getResources().getString(R.string.help);
    item.type = Item.TYPE_HEADER;
    models.add(item);
    item = new Item();
    item.text = getResources().getString(R.string.instructions);
    item.icon = new Integer(R.drawable.help);
    models.add(item);

    setListAdapter(new ModelChooserListAdapter(models));
}
```

This code may look a little complex, but its task is quite simple. It retrieves a list of all the models we have in our assets folder and creates a nice list out of them. If there is a corresponding image file to go with the model, it displays that image in icon style next to the name of the object; otherwise it simply displays a cross-like image. Apart from adding the models that are shipped with the app, this code also adds the option to select your own model and to access the help files.

Listening for Clicks

Next, we need a method to listen for clicks and do the appropriate work for each click.

Listing 8-3. *onListItemClick() method*

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    Item item = (Item) this.getListAdapter().getItem(position);
    String str = item.text;
    if(str.equals(getResources().getString(R.string.choose_custom_model))) {
    //start oi file manager activity
    Intent intent = new Intent(ModelChooser.this,
CheckFileManagerActivity.class);
    startActivity(intent);
    } else if(str.equals(getResources().getString(R.string.instructions))) {
        //show the instructions activity
        startActivity(new Intent(ModelChooser.this, Instructions.class));
    } else {
        //load the selected internal file
        Intent intent = new Intent(ModelChooser.this, ModelViewer.class);
    intent.putExtra("name", str+".obj");
    intent.putExtra("type", ModelViewer.TYPE_INTERNAL);
    intent.setAction(Intent.ACTION_VIEW);
    startActivity(intent);
    }
}
```

This code listens for clicks on any of our list items. When a click is detected, it checks to see which item was clicked. If the user wants to select an external model, we use an intent to check for and launch the OI Filemanager. If the user wants to see the instructions, we launch the instructions activity. If an internal model is selected, we launch the model viewer while setting its action to `ACTION_VIEW` and sending the name of the model as an extra.

List Adapter

If you've been looking closely at the code in the `onCreate`, you will see an error where we are setting the adapter for our list. We'll fix this now by creating an inner class to function as our list's adapter.

Listing 8-4. *The Adapter for our list*

```
    class ModelChooserListAdapter extends BaseAdapter{
        private Vector<Item> items;

        public ModelChooserListAdapter(Vector<Item> items) {
            this.items = items;
        }

        public int getCount() {
            return items.size();
        }

        public Object getItem(int position) {
            return items.get(position);
        }

        public long getItemId(int position) {
            return position;
        }

        @Override
        public int getViewTypeCount() {
            //normal items, and the header
            return 2;
        }

        @Override
        public boolean areAllItemsEnabled() {
            return false;
        }

        @Override
        public boolean isEnabled(int position) {
            return !(items.get(position).type==Item.TYPE_HEADER);
    }

        @Override
        public int getItemViewType(int position) {
            return items.get(position).type;
        }


        public View getView(int position, View convertView, ViewGroup parent) {
            View v = convertView;
            Item item = items.get(position);
            if (v == null) {
                LayoutInflater vi =
(LayoutInflater)getSystemService(Context.LAYOUT_INFLATER_SERVICE);
                switch(item.type) {
                case Item.TYPE_HEADER:
                    v = vi.inflate(R.layout.list_header, null);
                         break;
                case Item.TYPE_ITEM:
                     v = vi.inflate(R.layout.choose_model_row, null);
                         break;
                }
            }
            if(item != null) {
                switch(item.type) {
                    case Item.TYPE_HEADER:
                        TextView headerText = (TextView)
v.findViewById(R.id.list_header_title);
                        if(headerText != null) {
                            headerText.setText(item.text);
                        }
                        break;
                    case Item.TYPE_ITEM:
                        Object iconImage = item.icon;
                        ImageView icon = (ImageView)
v.findViewById(R.id.choose_model_row_icon);
                        if(icon!=null) {
                            if(iconImage instanceof Integer) {

icon.setImageResource(((Integer)iconImage).intValue());
                            } else if(iconImage instanceof Bitmap) {
                                icon.setImageBitmap((Bitmap)iconImage);
                            }
                        }
            TextView text = (TextView)
v.findViewById(R.id.choose_model_row_text);
            if(text!=null)
            text.setText(item.text);
        break;
        }
            }
            return v;
        }

    }
```

In a nutshell, this code is responsible for actually pulling the icon images, names, and so on; and then creating a list out of them. There is nothing remarkable about it. It's more or less standard Android code when working with lists.

The following is another extremely small inner class that deals with our items.

Listing 8-5. *The inner class Item*

```
    class Item {
        private static final int TYPE_ITEM=0;
        private static final int TYPE_HEADER=1;
        private int type = TYPE_ITEM;
        private Object icon = new Integer(R.drawable.missingimage);
        private String text;
    }
```

These five variables are used to set up each row in our list. `TYPE_ITEM` is a constant that we can use to denote a row with a model in it instead of using integers. `TYPE_HEADER` is the same as `TYPE_ITEM`, except that it is for headers. The `type` variable is used to store the type of the item that is currently being worked upon. It is set to TYPE_ITEM by default. The icon variable is used to denote the icon used whenever a corresponding image is not available for a model. The text variable is used to store the text of the current item being worked upon.

This brings us to the end of the main ModelChooser class. Don't forget to insert a final "`}`" to close the entire outer class.

Now that we've created our main Activity, let's tackle the remaining 21 Java files in alphabetical order to keep track of them easily and to make it all a tad bit easier.

AssetsFileUtility.java

We now need to create a file called `AssetsFileUtility`, which will be responsible for reading in the data we have stored in our `/assets` folder. The `/assets` folder is a place in which you can store any file you want and then later retrieve it as a raw byte stream. In the capability to store raw files, it is similar to `/res/raw`. However, a file stored in `/res/raw` can be localized and accessed through a resource id such as `R.raw.filename`. The `/assets` folder offers no localization or resource id access.

Listing 8-6. *AssetsFileUtility.java*

```
public class AssetsFileUtility extends BaseFileUtil {
    private AssetManager am;

    public AssetsFileUtility(AssetManager am) {
        this.am = am;
    }

    @Override
    public Bitmap getBitmapFromName(String name) {
        InputStream is = getInputStreamFromName(name);
        return (is==null)?null:BitmapFactory.decodeStream(is);
    }
    @Override
    public BufferedReader getReaderFromName(String name) {
        InputStream is = getInputStreamFromName(name);
        return (is==null)?null:new BufferedReader(new InputStreamReader(is));
    }

    private InputStream getInputStreamFromName(String name) {
        InputStream is;
        if(baseFolder != null) {
            try {
                is = am.open(baseFolder+name);
            } catch (IOException e) {
                e.printStackTrace();
                return null;
            }
        } else {
            try {
                is = am.open(name);
            } catch (IOException e) {
                e.printStackTrace();
                return null;
            }
        }
        return is;
    }

}
```

This code helps us retrieve a file from the `/assets` folder. It handles most of the work, such as creating `InputStreamReaders`, and so on. You will get an `IOException` if the file you are trying to read doesn't exist or is in some other way invalid (for example, by having an invalid file extension).

BaseFileUtil.java

Next up is a miniature class called `BaseFileUtil.java`. This file is the base of others such as AssetsFileUtility. It allows us to conveniently update the folder in which the model being viewed is located.

Listing 8-7. *BaseFileUtil.java*

```
public abstract class BaseFileUtil {
    protected String baseFolder = null;

    public String getBaseFolder() {
        return baseFolder;
    }
    public void setBaseFolder(String baseFolder) {
        this.baseFolder = baseFolder;
    }

    public abstract BufferedReader getReaderFromName(String name);
    public abstract Bitmap getBitmapFromName(String name);

}
```

CheckFileManagerActivity.java

Next up on our alphabetical list is `CheckFileManagerActivity`., which is called when the user wants to supply his own object to be augmented by the app. By allowing the user to view his own models, we effectively make this app into a full-fledged 3D AR viewer. The user can create designs for a chair, for example, and see how it will look in his house before having it built. This extends the usability for our app immensely. Currently, the app only supports OI Filemanager for selecting new files, but you could modify the code to allow the app to work with other file managers as well. I chose OI as the default one because it comes preinstalled on a lot of devices, and is often installed if not.

Code Listing

Let's take a look at `CheckFileManagerActivity.java` section by section.

Declarations

First are the declarations needed in this class.

Listing 8-8. *CheckFileManagerActivity.java declarations*

```
public class CheckFileManagerActivity extends Activity {

    private final int PICK_FILE = 1;
    private final int VIEW_MODEL = 2;
    public static final int RESULT_ERROR = 3;

    private final int INSTALL_INTENT_DIALOG=1;

    private PackageManager packageManager;
    private Resources res;
    private TextView infoText;

    private final int TOAST_TIMEOUT = 3;
```

onCreate() and onResume()

Immediately following the declarations are the `onCreate()` and `onResume()` methods.

The first thing we do is check whether the OI File Manager is installed. If it isn't, we ask the user to install it. If the File Manager is available, we allow the user to select a file. See Listing 8-9.

Listing 8-9. *onCreate() and onResume()*

```
    @Override
    final public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Context context = this;
        packageManager= context.getPackageManager();
        res = this.getResources();
        infoText = (TextView) findViewById(R.id.InfoText);
        if (isPickFileIntentAvailable()) {
            selectFile();
        } else {
            installPickFileIntent();
        }
    }


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

onActivityResult()

If the file selected is not a valid model file, we display a toast telling the user that and ask him to select again. If the selected file is a valid model file, we pass control to the Model Viewer, which will then have the file parsed and display it. If the user cancels the operation, we return the app to the Model Chooser screen.

Listing 8-10. *onActivityResult()*

```
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent ➥
data) {
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode) {
    default:
    case PICK_FILE:
```

```
    switch(resultCode) {
    case Activity.RESULT_OK:
    //does file exist??
    File file =  new File(URI.create(data.getDataString()));
    if (!file.exists()) {
        //notify user that this file doesn't exist
        Toast.makeText(this, res.getText(R.string.file_doesnt_exist),
TOAST_TIMEOUT).show();
         selectFile();
} else {
    String fileName = data.getDataString();
if(!fileName.endsWith(".obj")) {
    Toast.makeText(this, res.getText(R.string.wrong_file),
TOAST_TIMEOUT).show();
    selectFile();
} else {
    //hand over control to the model viewer
    Intent intent = new Intent(CheckFileManagerActivity.this,
ModelViewer.class);
    intent.putExtra("name", data.getDataString());
    intent.putExtra("type", ModelViewer.TYPE_EXTERNAL);
    intent.setAction(Intent.ACTION_VIEW);
    startActivityForResult(intent, VIEW_MODEL);
}
}
break;
default:
case Activity.RESULT_CANCELED:
    //back to the main activity
    Intent intent = new Intent(CheckFileManagerActivity.this,
ModelChooser.class);
    startActivity(intent);
break;
}
break;
case VIEW_MODEL:
switch(resultCode) {
case Activity.RESULT_OK:
    //model viewer returned...let the user view a new file
    selectFile();
break;
case Activity.RESULT_CANCELED:
    selectFile();
break;
case RESULT_ERROR:
    //something went wrong ... notify the user
    if(data != null) {
        Bundle extras = data.getExtras();
        String errorMessage = extras.getString("error_message");
        if(errorMessage != null)
        Toast.makeText(this, extras.getString("error_message"),
TOAST_TIMEOUT).show();
    }
selectFile();
break;
}
}
 }
```

selectFile()

The selectFile() method allows the user to select a model file.

Listing 8-11. *selectFile()*

```
    /** Let the user select a File. The selected file will be handled in
     *  {@link
edu.dhbw.andobjviewer.CheckFileManagerActivity#onActivityResult(int, int,
Intent)} */
    private void selectFile() {
            //let the user select a model file
        Intent intent = new Intent("org.openintents.action.PICK_FILE");
        intent.setData(Uri.parse("file:///sdcard/"));
        intent.putExtra("org.openintents.extra.TITLE", res.getText(
            R.string.select_model_file));
        startActivityForResult(intent, PICK_FILE);
    }
```

isPickFileIntentAvailable() and installPickFileIntent()

The `isPickFileIntentAvailable()` and `installPickFileIntent()` methods are called in the onCreate method().

Listing 8-12. *isPickFileIntentAvailable() and installPickFileIntent()*

```
    private boolean isPickFileIntentAvailable() {
        return packageManager.queryIntentActivities(
            new Intent("org.openintents.action.PICK_FILE"), 0).size() > 0;
    }

    private boolean installPickFileIntent() {
           Uri marketUri =
Uri.parse("market://search?q=pname:org.openintents.filemanager");
        Intent marketIntent = new Intent(Intent.ACTION_VIEW).setData(marketUri);
        if (!(packageManager
            .queryIntentActivities(marketIntent, 0).size() > 0)) {
            //no Market available
            //show info to user and exit
            infoText.setText(res.getText(R.string.android_markt_not_avail));
            return false;
        } else {
            //notify user and start Android market

            showDialog(INSTALL_INTENT_DIALOG);
                return true;
        }
    }
```

onCreateDialog()

The final method in `CheckFileManagerActivity.java` is `onCreateDialog()`.

Listing 8-13. *onCreateDialog()*

```
    @Override
    protected Dialog onCreateDialog(int id) {
        Dialog dialog = null;
        switch(id){
            case INSTALL_INTENT_DIALOG:
                AlertDialog alertDialog = new
AlertDialog.Builder(this).create();

alertDialog.setMessage(res.getText(R.string.pickfile_intent_required));
                alertDialog.setButton("OK", new
DialogInterface.OnClickListener() {
                  public void onClick(DialogInterface dialog, int which) {
                     //launch android market
                   Uri marketUri =
Uri.parse("market://search?q=pname:org.openintents.filemanager");
                        Intent marketIntent = new
Intent(Intent.ACTION_VIEW).setData(marketUri);
                        startActivity(marketIntent);
                        return;
              } });
           dialog = alertDialog;
            break;
        }
       return dialog;
    }

}
```

Configuration File

Next in our list is the Config.java file. This is literally the smallest Java file you will ever see. Excluding the package name, it is only three lines in size.

Listing 8-14. *Config.java*

```
public class Config {
    public final static boolean DEBUG = false;
}
```

This file is technically the configuration file, even though it has only one option. Setting DEBUG to true will put the app in DEBUG mode. If you ever decide to extend the app, you can add other configuration options here, such as flags for which market you're publishing the apk to.

Working with Numbers

Next is the FixedPointUtilities class, which handles some of our mathematical functions, mostly converting arrays etc. It is very important for keeping our model looking the way it should.

Listing 8-15. *FixedPointUtilities.java*

```
public class FixedPointUtilities {
    public static final int ONE = 0x10000;

    public static int toFixed(float val) {
        return (int)(val * 65536F);
    }

    public static int[] toFixed(float[] arr) {
        int[] res = new int[arr.length];
        toFixed(arr, res);
        return res;
    }

    public static void toFixed(float[] arr, int[] storage)
    {
        for (int i=0;i<storage.length;i++) {
            storage[i] = toFixed(arr[i]);
       }
    }


    public static float toFloat(int val) {
        return ((float)val)/65536.0f;
    }

    public static float[] toFloat(int[] arr) {
        float[] res = new float[arr.length];
        toFloat(arr, res);
        return res;
    }

    public static void toFloat(int[] arr, float[] storage)
    {
        for (int i=0;i<storage.length;i++) {
            storage[i] = toFloat(arr[i]);
        }
    }

    public static int multiply (int x, int y) {
        long z = (long) x * (long) y;
        return ((int) (z >> 16));
    }

    public static int divide (int x, int y) {
        long z = (((long) x) << 32);
        return (int) ((z / y) >> 16);
    }

    public static int sqrt (int n) {
        int s = (n + 65536) >> 1;
        for (int i = 0; i < 8; i++) {
            s = (s + divide(n, s)) >> 1;
        }
        return s;
     }
}
```

Now let’s look at the methods in this class. The first method converts a single float value to a 16.16 fixed point value.

```
   public static int toFixed(float val) {
        return (int)(val * 65536F);
    }
```

The second method does the same thing, only it does it to an array of floats.

```
    public static int[] toFixed(float[] arr) {
        int[] res = new int[arr.length];
        toFixed(arr, res);
        return res;
    }
```

The third method is called by the second method to help in its work.

```
    public static void toFixed(float[] arr, int[] storage)
    {
        for (int i=0;i<storage.length;i++) {
            storage[i] = toFixed(arr[i]);
       }
    }
```

The fourth method converts a single fixed-point value into a float.

```
    public static float toFloat(int val) {
        return ((float)val)/65536.0f;
    }
```

The fifth method does the same to an array of fixed-point values, and it calls the sixth method to help it.

```
    public static float[] toFloat(int[] arr) {
        float[] res = new float[arr.length];
        toFloat(arr, res);
        return res;
    }

    public static void toFloat(int[] arr, float[] storage)
    {
        for (int i=0;i<storage.length;i++) {
            storage[i] = toFloat(arr[i]);
        }
    }
```

The seventh method multiplies two fixed point values, while the eighth method divides two fixed point values.

```
    public static int multiply (int x, int y) {
        long z = (long) x * (long) y;
        return ((int) (z >> 16));
    }

    public static int divide (int x, int y) {
        long z = (((long) x) << 32);
        return (int) ((z / y) >> 16);
    }
```

The ninth and last method finds the square root of a fixed point value.

```
    public static int sqrt (int n) {
        int s = (n + 65536) >> 1;
        for (int i = 0; i < 8; i++) {
            s = (s + divide(n, s)) >> 1;
        }
        return s;
    }
```

These methods are called from MatrixUtils.java. Our models are essentially a huge number of vertices. When we parse them, we need to deal with those vertices, and this helps us do so.

Group.java

Next, we have a class called `Group.java`. This class is mainly required for and used in parsing the .obj files and their .mtl counterparts; and making proper, user-friendly graphics out of them. This is a relatively small part of our object parsing, but is still important.

In OpenGL, every graphic is a set of coordinates called vertices. When three or more of these vertices are joined by lines, they are called faces. Several faces are often grouped together. Faces may or may not have a texture. A texture alters the way light is reflected off a particular face. This class deals with the creation of groups, associating each group to a material, and setting its texture.

Listing 8-16. *Group.java*

```
public class Group implements Serializable {
    private String materialName = "default";
    private transient Material material;

    private boolean textured = false;
    public transient FloatBuffer vertices = null;
    public transient FloatBuffer texcoords = null;
    public transient FloatBuffer normals = null;
    public int vertexCount = 0;

    public ArrayList<Float> groupVertices = new ArrayList<Float>(500);
    public ArrayList<Float> groupNormals = new ArrayList<Float>(500);
    public ArrayList<Float> groupTexcoords = new ArrayList<Float>();

    public Group() {
    }

    public void setMaterialName(String currMat) {
        this.materialName = currMat;
    }

    public String getMaterialName() {
        return materialName;
    }
    public Material getMaterial() {
        return material;
    }

    public void setMaterial(Material material) {
        if(texcoords != null && material != null && material.hasTexture()) {
            textured = true;
        }
        if(material != null)
            this.material = material;
    }

    public boolean containsVertices() {
        if(groupVertices != null)
            return groupVertices.size()>0;
        else if(vertices != null)
            return vertices.capacity()>0;
        else
            return false;
    }

    public void setTextured(boolean b) {
        textured = b;
    }

    public boolean isTextured() {
        return textured;
    }

    public void finalize() {
        if (groupTexcoords.size() > 0) {
            textured = true;
            texcoords = MemUtil.makeFloatBuffer(groupTexcoords.size());
            for (Iterator<Float> iterator = groupTexcoords.iterator();
iterator.hasNext();) {
                Float curVal = iterator.next();
                texcoords.put(curVal.floatValue());
            }
            texcoords.position(0);
            if(material != null && material.hasTexture()) {
            textured = true;
            } else {
                textured = false;
            }
        }
        groupTexcoords = null;
        vertices = MemUtil.makeFloatBuffer(groupVertices.size());
        vertexCount = groupVertices.size()/3;//three floats pers vertex
        for (Iterator<Float> iterator = groupVertices.iterator();
iterator.hasNext();) {
```

```
            Float curVal = iterator.next();
            vertices.put(curVal.floatValue());
        }
        groupVertices = null;
        normals = MemUtil.makeFloatBuffer(groupNormals.size());
        for (Iterator<Float> iterator = groupNormals.iterator();
iterator.hasNext();) {
            Float curVal =  iterator.next();
            normals.put(curVal.floatValue());
        }
        groupNormals = null;
        vertices.position(0);
        normals.position(0);
    }
}
```

The code mostly deals with adding texture and “material” to the graphic we are parsing. It sets the texture to be used for the graphic and the material. Of course, this material is only virtual and is not technically speaking actual material.

Instructions.java

Next is another very simple file. This file is called `Instructions.java` and contains the `Activity` that displays our app's instructions by displaying an HTML file located in our `/assets/help` in a `WebView`.

Listing 8-17. *Instructions.java*

```
public class Instructions extends Activity {

   private WebView mWebView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.instructions_layout);
      mWebView = (WebView) findViewById(R.id.instructions_webview);

      WebSettings webSettings = mWebView.getSettings();
        webSettings.setSupportZoom(true);
        webSettings.setBuiltInZoomControls(true);

        WebChromeClient client = new WebChromeClient();
        mWebView.setWebChromeClient(client);
mWebView.loadUrl("file:///android_asset/help/"+getResources().getString(R.string
.help_file));
   }
}
```

The Activity starts, and a single WebView is set as its view. The WebView is then passed an HTML file that contains our help and is stored in the assets.

Working with Light

Now we get to some of the more complex stuff. Our models are rendered using OpenGL. To make them look even better, we also implement various lighting techniques. For this lighting, we have a class called `LightingRenderer`.

Listing 8-18. *LightingRenderer.java*

```
public class LightingRenderer implements OpenGLRenderer {

   private float[] ambientlight0 = {.3f, .3f, .3f, 1f};
   private float[] diffuselight0 = {.7f, .7f, .7f, 1f};
   private float[] specularlight0 = {0.6f, 0.6f, 0.6f, 1f};
   private float[] lightposition0 = {100.0f,-200.0f,200.0f,0.0f};

   private FloatBuffer lightPositionBuffer0 =
GraphicsUtil.makeFloatBuffer(lightposition0);
   private FloatBuffer specularLightBuffer0 =
GraphicsUtil.makeFloatBuffer(specularlight0);
   private FloatBuffer diffuseLightBuffer0 =
GraphicsUtil.makeFloatBuffer(diffuselight0);
   private FloatBuffer ambientLightBuffer0 =
GraphicsUtil.makeFloatBuffer(ambientlight0);


   private float[] ambientlight1 = {.3f, .3f, .3f, 1f};
   private float[] diffuselight1 = {.7f, .7f, .7f, 1f};
   private float[] specularlight1 = {0.6f, 0.6f, 0.6f, 1f};
   private float[] lightposition1 = {20.0f,-40.0f,100.0f,1f};

   private FloatBuffer lightPositionBuffer1 =
GraphicsUtil.makeFloatBuffer(lightposition1);
   private FloatBuffer specularLightBuffer1 =
GraphicsUtil.makeFloatBuffer(specularlight1);
   private FloatBuffer diffuseLightBuffer1 =
GraphicsUtil.makeFloatBuffer(diffuselight1);
   private FloatBuffer ambientLightBuffer1 =
GraphicsUtil.makeFloatBuffer(ambientlight1);
```

```
   private float[] ambientlight2 = {.4f, .4f, .4f, 1f};
   private float[] diffuselight2 = {.7f, .7f, .7f, 1f};
   private float[] specularlight2 = {0.6f, 0.6f, 0.6f, 1f};
   private float[] lightposition2 = {5f,-3f,-20f,1.0f};

   private FloatBuffer lightPositionBuffer2 =
GraphicsUtil.makeFloatBuffer(lightposition2);
   private FloatBuffer specularLightBuffer2 =
GraphicsUtil.makeFloatBuffer(specularlight2);
   private FloatBuffer diffuseLightBuffer2 =
GraphicsUtil.makeFloatBuffer(diffuselight2);
   private FloatBuffer ambientLightBuffer2 =
GraphicsUtil.makeFloatBuffer(ambientlight2);

   private float[] ambientlight3 = {.4f, .4f, .4f, 1f};
   private float[] diffuselight3 = {.4f, .4f, .4f, 1f};
   private float[] specularlight3 = {0.6f, 0.6f, 0.6f, 1f};
   private float[] lightposition3 = {0,0f,-1f,0.0f};

   private FloatBuffer lightPositionBuffer3 =
GraphicsUtil.makeFloatBuffer(lightposition3);
   private FloatBuffer specularLightBuffer3 =
GraphicsUtil.makeFloatBuffer(specularlight3);
   private FloatBuffer diffuseLightBuffer3 =
GraphicsUtil.makeFloatBuffer(diffuselight3);
   private FloatBuffer ambientLightBuffer3 =
GraphicsUtil.makeFloatBuffer(ambientlight3);

   public final void draw(GL10 gl) {

   }

   public final void setupEnv(GL10 gl) {
      gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_AMBIENT, ambientLightBuffer0);
      gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_DIFFUSE, diffuseLightBuffer0);
      gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_SPECULAR, specularLightBuffer0);
      gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION, lightPositionBuffer0);
      gl.glEnable(GL10.GL_LIGHT0);
      gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_AMBIENT, ambientLightBuffer1);
      gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_DIFFUSE, diffuseLightBuffer1);
      gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_SPECULAR, specularLightBuffer1);
      gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_POSITION, lightPositionBuffer1);
      gl.glEnable(GL10.GL_LIGHT1);
      gl.glLightfv(GL10.GL_LIGHT2, GL10.GL_AMBIENT, ambientLightBuffer2);
      gl.glLightfv(GL10.GL_LIGHT2, GL10.GL_DIFFUSE, diffuseLightBuffer2);
      gl.glLightfv(GL10.GL_LIGHT2, GL10.GL_SPECULAR, specularLightBuffer2);
      gl.glLightfv(GL10.GL_LIGHT2, GL10.GL_POSITION, lightPositionBuffer2);
      gl.glEnable(GL10.GL_LIGHT2);
      gl.glLightfv(GL10.GL_LIGHT3, GL10.GL_AMBIENT, ambientLightBuffer3);
      gl.glLightfv(GL10.GL_LIGHT3, GL10.GL_DIFFUSE, diffuseLightBuffer3);
      gl.glLightfv(GL10.GL_LIGHT3, GL10.GL_SPECULAR, specularLightBuffer3);
      gl.glLightfv(GL10.GL_LIGHT3, GL10.GL_POSITION, lightPositionBuffer3);
      gl.glEnable(GL10.GL_LIGHT3);
      initGL(gl);
   }

   public final void initGL(GL10 gl) {
      gl.glDisable(GL10.GL_COLOR_MATERIAL);
      gl.glShadeModel(GL10.GL_SMOOTH);
      gl.glEnable(GL10.GL_LIGHTING);
      //gl.glEnable(GL10.GL_CULL_FACE);
      gl.glEnable(GL10.GL_DEPTH_TEST);
      gl.glEnable(GL10.GL_NORMALIZE);
      gl.glEnable(GL10.GL_RESCALE_NORMAL);
   }
}
```

We create floats that store the values for the lighting in different parts and circumstances and then create `FloatBuffers` out of them. All this is then applied to our app in the `setupEnv()` method and finally put out by the `initGL` method. This code is much closer to AR than the rest of the code seen so far in this chapter and is very important to make sure that our graphic's lighting is good and it looks realistic. OpenGL supports a total of eight different lighting configurations, and we create four of them (`GL_LIGHT0-8`). We have different ambient, specular, and diffusion light settings for all of them, which allows us to give the model four different kind of looks. All the lights are set to `GL_SMOOTH`, which takes more computational power, but results in a more realistic model.

Creating a Material

Now we come to our `Material` class. This class is the material mentioned earlier that was used in the `Group` class. In the real world, light is reflected off the material of an object. Some kinds of material reflect green shades, some red, some blue, and so on. Similarly, in OpenGL we create so-called material objects that then in turn make our final model up. Each material object is set to reflect a particular shade of light. When this is combined with our lighting effects, we get a combination of the two lights. For example, a red material ball will appear black with a blue lighting source because the red material will not reflect a shade of blue; it will reflect only a shade of red. This class handles all the OpenGL code related to materials.

Listing 8-19. *Material.java*

```
public class Material implements Serializable {

   private float[] ambientlightArr = {0.2f, 0.2f, 0.2f, 1.0f};
private float[] diffuselightArr = {0.8f, 0.8f, 0.8f, 1.0f};
private float[] specularlightArr = {0.0f, 0.0f, 0.0f, 1.0f};

public transient FloatBuffer ambientlight = MemUtil.makeFloatBuffer(4);
public transient FloatBuffer diffuselight = MemUtil.makeFloatBuffer(4);
public transient FloatBuffer specularlight = MemUtil.makeFloatBuffer(4);
public float shininess = 0;
public int STATE = STATE_DYNAMIC;
public static final int STATE_DYNAMIC = 0;
public static final int STATE_FINALIZED = 1;

private transient Bitmap texture = null;
private String bitmapFileName = null;
private transient BaseFileUtil fileUtil = null;

private String name = "defaultMaterial";

public Material() {

}

public Material(String name) {
   this.name = name;
   //fill with default values
   ambientlight.put(new float[]{0.2f, 0.2f, 0.2f, 1.0f});
   ambientlight.position(0);
   diffuselight.put(new float[]{0.8f, 0.8f, 0.8f, 1.0f});
   diffuselight.position(0);
   specularlight.put(new float[]{0.0f, 0.0f, 0.0f, 1.0f});
   specularlight.position(0);
}

public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
}

public void setFileUtil(BaseFileUtil fileUtil) {
   this.fileUtil = fileUtil;
}

public String getBitmapFileName() {
   return bitmapFileName;
}

public void setBitmapFileName(String bitmapFileName) {
   this.bitmapFileName = bitmapFileName;
```

```
}

public void setAmbient(float[] arr) {
   ambientlightArr = arr;
}

public void setDiffuse(float[] arr) {
   diffuselightArr = arr;
}

public void setSpecular(float[] arr) {
   specularlightArr = arr;
}

public void setShininess(float ns) {
   shininess = ns;
}

public void setAlpha(float alpha) {
   ambientlight.put(3, alpha);
   diffuselight.put(3, alpha);
   specularlight.put(3, alpha);
}

public Bitmap getTexture() {
   return texture;
}

public void setTexture(Bitmap texture) {
   this.texture = texture;
}

public boolean hasTexture() {
   if(STATE == STATE_DYNAMIC)
      return this.bitmapFileName != null;
   else if(STATE == STATE_FINALIZED)
      return this.texture != null;
   else
      return false;
}

public void finalize() {
   ambientlight = MemUtil.makeFloatBuffer(ambientlightArr);
   diffuselight = MemUtil.makeFloatBuffer(diffuselightArr);
   specularlight = MemUtil.makeFloatBuffer(specularlightArr);
   ambientlightArr = null;
   diffuselightArr = null;
   specularlightArr = null;
   if(fileUtil != null && bitmapFileName != null) {
      texture = fileUtil.getBitmapFromName(bitmapFileName);
```

```
        }
    }
}
```

This class helps us create new materials as and when required. There is a default material specified, but the setter methods such as `setAmbient()`, `setDiffuse()`, `setSpecular()`, and `setShininess()` allow us to specify the reflection values of the new array, along with its ambient light, and so on. The finalize method converts the lighting to FloatBuffers and assigns a value to texture.

MemUtil.java

Next up we have a rather small class that is used to create FloatBuffers. This is our `MemUtil` class.

Listing 8-20. *MemUtil.java*

```
public class MemUtil {

    public static FloatBuffer makeFloatBufferFromArray(float[] arr) {
        ByteBuffer bb = ByteBuffer.allocateDirect(arr.length*4);
        bb.order(ByteOrder.nativeOrder());
        FloatBuffer fb = bb.asFloatBuffer();
        fb.put(arr);
        fb.position(0);
        return fb;
    }

    public static FloatBuffer makeFloatBuffer(int size) {
        ByteBuffer bb = ByteBuffer.allocateDirect(size*4);
        bb.order(ByteOrder.nativeOrder());
        FloatBuffer fb = bb.asFloatBuffer();
        fb.position(0);
        return fb;
    }

    public static FloatBuffer makeFloatBuffer(float[] arr) {
        ByteBuffer bb = ByteBuffer.allocateDirect(arr.length*4);
        bb.order(ByteOrder.nativeOrder());
        FloatBuffer fb = bb.asFloatBuffer();
        fb.put(arr);
        fb.position(0);
        return fb;
    }

}
```

This class is pretty straightforward and doesn't need much explanation, as it is pretty much standard Java. We need the floatbuffers as OpenGL only accepts floatbuffer arguments in its lighting and material implementations.

Model.java

Now we have a class that is very important to our app, `Model.java`.

Listing 8-21. *Model.java*

```
public class Model implements Serializable{

    public float xrot = 90;
    public float yrot = 0;
    public float zrot = 0;
    public float xpos = 0;
    public float ypos = 0;
    public float zpos = 0;
    public float scale = 4f;
    public int STATE = STATE_DYNAMIC;
    public static final int STATE_DYNAMIC = 0;
    public static final int STATE_FINALIZED = 1;

    private Vector<Group> groups = new Vector<Group>();
    protected HashMap<String, Material> materials = new HashMap<String,
Material>();
    public Model() {
        materials.put("default",new Material("default"));
    }
    public void addMaterial(Material mat) {
        materials.put(mat.getName(), mat);
    }
    public Material getMaterial(String name) {
        return materials.get(name);
    }
    public void addGroup(Group grp) {
        if(STATE == STATE_FINALIZED)
            grp.finalize();
        groups.add(grp);
    }
    public Vector<Group> getGroups() {
         return groups;
    }
    public void setFileUtil(BaseFileUtil fileUtil) {
        for (Iterator iterator = materials.values().iterator();
iterator.hasNext();) {
            Material mat = (Material) iterator.next();
            mat.setFileUtil(fileUtil);
        }
    }
    public HashMap<String, Material> getMaterials() {
        return materials;
    }
    public void setScale(float f) {
        this.scale += f;
        if(this.scale < 0.0001f)
            this.scale = 0.0001f;
    }
    public void setXrot(float dY) {
        this.xrot += dY;
    }
    public void setYrot(float dX) {
        this.yrot += dX;
    }
    public void setXpos(float f) {
        this.xpos += f;
    }
    public void setYpos(float f) {
        this.ypos += f;
    }
    public void finalize() {
        if(STATE != STATE_FINALIZED) {
            STATE = STATE_FINALIZED;
            for (Iterator iterator = groups.iterator(); iterator.hasNext();) {
                Group grp = (Group) iterator.next();
                grp.finalize();
                grp.setMaterial(materials.get(grp.getMaterialName()));
            }
            for (Iterator<Material> iterator = materials.values().iterator();
iterator.hasNext();) {
                Material mtl = iterator.next();
                mtl.finalize();
            }
        }
    }
}
```

This class does a lot of the groundwork for creating our models. Let's look at this one method by method. The `Model()` method is the constructor and sets the default material for our models. The `addMaterial()` method adds a material to our app. The addGroup() method adds another group to our groups, also finalizing it if needed. The `setFileUtil()` method takes a `BaseFileUtil` as an argument and then uses it to set the fileUtil for all our materials. The `setScale()` method allows us to pass a float to be set as the scale. It also makes sure that the scale is a nonzero and positive value. This scale value is used to scale the model. The `setXrot()` and `setYrot()` methods allow us to set the rotation on our model for the X-axis and Y-axis. The `setXpos()` and `setYpos()` methods are used to set the position of the model on the X-axis and Y-axis. The `finalize()` method finalizes everything and makes it nonalterable.

Model3D.java

Next on our list is `Model3D.java`, which is responsible for a good amount of the drawing of our models. The explanation comes after the code.

Listing 8-22. *Model3D.java*

```
public class Model3D extends ARObject implements Serializable{

   private Model model;
   private Group[] texturedGroups;
   private Group[] nonTexturedGroups;
   private HashMap<Material, Integer> textureIDs = new HashMap<Material,
Integer>();

   public Model3D(Model model, String patternName) {
      super("model", patternName, 80.0, new double[]{0,0});
      this.model = model;
      model.finalize();

      Vector<Group> groups = model.getGroups();
      Vector<Group> texturedGroups = new Vector<Group>();
      Vector<Group> nonTexturedGroups = new Vector<Group>();
      for (Iterator<Group> iterator = groups.iterator(); iterator.hasNext();) {
         Group currGroup = iterator.next();
         if(currGroup.isTextured()) {
            texturedGroups.add(currGroup);
         } else {
            nonTexturedGroups.add(currGroup);
         }
      }
      this.texturedGroups = texturedGroups.toArray(new
Group[texturedGroups.size()]);
      this.nonTexturedGroups = nonTexturedGroups.toArray(new
Group[nonTexturedGroups.size()]);
   }

   @Override
   public void init(GL10 gl){
      int[]  tmpTextureID = new int[1];

      Iterator<Material> materialI = model.getMaterials().values().iterator();
      while (materialI.hasNext()) {
         Material material = (Material) materialI.next();
         if(material.hasTexture()) {

            gl.glGenTextures(1, tmpTextureID, 0);
            gl.glBindTexture(GL10.GL_TEXTURE_2D, tmpTextureID[0]);
            textureIDs.put(material, tmpTextureID[0]);
            GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, material.getTexture(),0);
            material.getTexture().recycle();
            gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER,
GL10.GL_LINEAR);
            gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER,
GL10.GL_LINEAR);
         }
      }
   }

   @Override
   public void draw(GL10 gl) {
      super.draw(gl);

      gl.glScalef(model.scale, model.scale, model.scale);
      gl.glTranslatef(model.xpos, model.ypos, model.zpos);
      gl.glRotatef(model.xrot, 1, 0, 0);
      gl.glRotatef(model.yrot, 0, 1, 0);
      gl.glRotatef(model.zrot, 0, 0, 1);

      gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
      gl.glEnableClientState(GL10.GL_NORMAL_ARRAY);

      gl.glDisable(GL10.GL_TEXTURE_2D);
      int cnt = nonTexturedGroups.length;
      for (int i = 0; i < cnt; i++) {
         Group group = nonTexturedGroups[i];
         Material mat = group.getMaterial();
         if(mat != null) {
            gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SPECULAR,
mat.specularlight);
            gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT,
mat.ambientlight);
            gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE,
mat.diffuselight);
            gl.glMaterialf(GL10.GL_FRONT_AND_BACK, GL10.GL_SHININESS,
mat.shininess);
         }
         gl.glVertexPointer(3,GL10.GL_FLOAT, 0, group.vertices);
         gl.glNormalPointer(GL10.GL_FLOAT,0, group.normals);
         gl.glDrawArrays(GL10.GL_TRIANGLES, 0, group.vertexCount);
      }

      gl.glEnable(GL10.GL_TEXTURE_2D);
      gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
      cnt = texturedGroups.length;
      for (int i = 0; i < cnt; i++) {
         Group group = texturedGroups[i];
         Material mat = group.getMaterial();
         if(mat != null) {
            gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SPECULAR,
mat.specularlight);
            gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT,
mat.ambientlight);
            gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE,
mat.diffuselight);
            gl.glMaterialf(GL10.GL_FRONT_AND_BACK, GL10.GL_SHININESS,
mat.shininess);
            if(mat.hasTexture()) {
               gl.glTexCoordPointer(2,GL10.GL_FLOAT, 0, group.texcoords);
               gl.glBindTexture(GL10.GL_TEXTURE_2D,
textureIDs.get(mat).intValue());
            }
         }
         gl.glVertexPointer(3,GL10.GL_FLOAT, 0, group.vertices);
         gl.glNormalPointer(GL10.GL_FLOAT,0, group.normals);
         gl.glDrawArrays(GL10.GL_TRIANGLES, 0, group.vertexCount);
      }

      gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
      gl.glDisableClientState(GL10.GL_NORMAL_ARRAY);
      gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
   }
}
```

In our constructor we get our model and then separate the textured groups from the nontextured groups to attain better performance. The `init()` method loads the textures for all of our materials. The main thing in this class is the `draw()` method. The first seven gl statements after the super statement do the positioning of our models. The next two statements and the `for` loop draw all the nontextured parts. The remainder of the method draws the textured parts of our model.

Viewing the Model

Next on our list is `ModelViewer.java`. This class is responsible for loading and displaying the model the user selected, whether from our provided models or from the SD Card. It is a big class and reasonably complex.

Variable Declarations

Global variables are used to store the location type of the file (inbuilt or external), the menu options, instances of Model and Model3D for each of the models, a progress dialog box and an instance of ARToolkit.

Listing 8-23. *ModelViewer.java Variables*

```
public class ModelViewer extends AndARActivity implements SurfaceHolder.Callback
{
   public static final int TYPE_INTERNAL = 0;
   public static final int TYPE_EXTERNAL = 1;
   public static final boolean DEBUG = false;
   private final int MENU_SCALE = 0;
   private final int MENU_ROTATE = 1;
   private final int MENU_TRANSLATE = 2;
   private final int MENU_SCREENSHOT = 3;

   private int mode = MENU_SCALE;
   private Model model;
   private Model model2;
   private Model model3;
   private Model model4;
   private Model model5;
   private Model3D model3d;
   private Model3D model3d2;
   private Model3D model3d3;
   private Model3D model3d4;
   private Model3D model3d5;
   private ProgressDialog waitDialog;
   private Resources res;
   ARToolkit artoolkit;
```

Constructor

The constructor for this class looks as follows.

Listing 8-24. *The constructor of ModelViewer.java*

```
   public ModelViewer() {
      super(false);
   }
```

onCreate() Method

The `onCreate()` method for our file sets the lighting via LightingRenderer.java, gets the resources for the app, assigns an instance of ARToolkit to `artoolkit`, sets the touch event listener for the surface view, and adds the call back for the surface view.

Listing 8-25. *The onCreate() method*

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   super.setNonARRenderer(new LightingRenderer());
   res=getResources();
   artoolkit = getArtoolkit();
   getSurfaceView().setOnTouchListener(new TouchEventHandler());
   getSurfaceView().getHolder().addCallback(this);
}
```

Catching Exceptions and Handling Menu Options

uncaughtException() catches any exception that we do not explicitly catch elsewhere. The other two methods are very common standard Android code dealing with creating the menu and listening to the user's activity on it.

Listing 8-26. *Catching Exceptions and using the menu*

```
public void uncaughtException(Thread thread, Throwable ex) {
    System.out.println("");
}
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, MENU_TRANSLATE, 0, res.getText(R.string.translate))
        .setIcon(R.drawable.translate);
    menu.add(0, MENU_ROTATE, 0, res.getText(R.string.rotate))
        .setIcon(R.drawable.rotate);
    menu.add(0, MENU_SCALE, 0, res.getText(R.string.scale))
        .setIcon(R.drawable.scale);
    menu.add(0, MENU_SCREENSHOT, 0, res.getText(R.string.take_screenshot))
        .setIcon(R.drawable.screenshoticon);
    return true;
}
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case MENU_SCALE:
            mode = MENU_SCALE;
            return true;
        case MENU_ROTATE:
            mode = MENU_ROTATE;
            return true;
        case MENU_TRANSLATE:
            mode = MENU_TRANSLATE;
```

```
                return true;
            case MENU_SCREENSHOT:
                new TakeAsyncScreenshot().execute();
                return true;
        }
        return false;
    }
```

surfaceCreated()

surfaceCreated() is called when the SurfaceView is created, it shows a progress dialog box while the model is loading.

Listing 8-27. *surfaceCreated()*

```
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        super.surfaceCreated(holder);

        if(model == null) {
            waitDialog = ProgressDialog.show(this, "",
getResources().getText(R.string.loading), true);
            waitDialog.show();
            new ModelLoader().execute();
        }
    }
```

TouchEventHandler Inner Class

This inner class intercepts every single touch event that takes place in our Activity. It takes the kind of event and then appropriately scales, translates, or rotates the model.

Listing 8-28. *The inner class TouchEventHandler*

```
    class TouchEventHandler implements OnTouchListener {
        private float lastX=0;
        private float lastY=0;
        public boolean onTouch(View v, MotionEvent event) {
        if(model!=null) {
            switch(event.getAction()) {
                default:
            case MotionEvent.ACTION_DOWN:
                lastX = event.getX();
                lastY = event.getY();
                break;
            case MotionEvent.ACTION_MOVE:
                float dX = lastX - event.getX();
                float dY = lastY - event.getY();
                lastX = event.getX();
                lastY = event.getY();
                if(model != null) {
                    switch(mode) {
                        case MENU_SCALE:
                            model.setScale(dY/100.0f);
                            break;
                        case MENU_ROTATE:
                             model.setXrot(-1*dX);
                             model.setYrot(-1*dY);
                             break;
                        case MENU_TRANSLATE:
                             model.setXpos(dY/10f);
                             model.setYpos(dX/10f);
                             break;
                    }
                }
                break;
            case MotionEvent.ACTION_CANCEL:
            case MotionEvent.ACTION_UP:
                lastX = event.getX();
                lastY = event.getY();
                break;
            }
            }
            return true;
        }
    }
```

ModelLoader Inner Class

In this ModelLoader inner class, we use a series of if else statements to determine the model that we need to load. We also set the different markers required for some of the inbuilt models. The default marker for some of the inbuilt models, and all external models are called android. If the model is from an external file, we first trim it before loading it. If it is an inbuilt model, we load it directly. In onPostExecute(), we register all the models, and dismiss the progress dialog box.

Listing 8-29. *ModelLoader*

```
private class ModelLoader extends AsyncTask<Void, Void, Void> {
    private String modelName2patternName (String modelName) {
        String patternName = "android";
        if (modelName.equals("plant.obj")) {
            patternName = "marker_rupee16";
        } else if (modelName.equals("chair.obj")) {
            patternName = "marker_fisch16";
        } else if (modelName.equals("tower.obj")) {
            patternName = "marker_peace16";
        } else if (modelName.equals("bench.obj")) {
            patternName = "marker_at16";
        } else if (modelName.equals("towergreen.obj")) {
            patternName = "marker_hand16";
        }
    return patternName;
}
    @Override
    protected Void doInBackground(Void... params) {
        Intent intent = getIntent();
        Bundle data = intent.getExtras();
        int type = data.getInt("type");
        String modelFileName = data.getString("name");
        BaseFileUtil fileUtil= null;
        File modelFile=null;
        switch(type) {
            case TYPE_EXTERNAL:
                fileUtil = new SDCardFileUtil();
                modelFile =  new File(URI.create(modelFileName));
                modelFileName = modelFile.getName();

fileUtil.setBaseFolder(modelFile.getParentFile().getAbsolutePath());
                break;
            case TYPE_INTERNAL:
                fileUtil = new AssetsFileUtility(getResources().getAssets());
                fileUtil.setBaseFolder("models/");
                break;
        }
        if(modelFileName.endsWith(".obj")) {
            ObjParser parser = new ObjParser(fileUtil);
            try {
                if(Config.DEBUG)
                Debug.startMethodTracing("AndObjViewer");
                if(type == TYPE_EXTERNAL) {
                BufferedReader modelFileReader = new BufferedReader(new
FileReader(modelFile));
                String shebang = modelFileReader.readLine();

                if(!shebang.equals("#trimmed")) {
                File trimmedFile = new File(modelFile.getAbsolutePath()+".tmp");
                BufferedWriter trimmedFileWriter = new BufferedWriter(new
FileWriter(trimmedFile));
                Util.trim(modelFileReader, trimmedFileWriter);
                if(modelFile.delete()) {
                trimmedFile.renameTo(modelFile);
                }
                }
                }
                if(fileUtil != null) {
                BufferedReader fileReader =
fileUtil.getReaderFromName(modelFileName);
                if(fileReader != null) {
                model = parser.parse("Model", fileReader);
                Log.w("ModelLoader", "model3d = new Model3D(model, " +
modelName2patternName(modelFileName) + ".patt");
               model3d = new Model3D(model, modelName2patternName(modelFileName)
+ ".patt");
               }
               String modelFileName2 = "chair.obj";
               BufferedReader fileReader2 =
fileUtil.getReaderFromName(modelFileName2);
               if(fileReader2 != null) {
               model2 = parser.parse("Chair", fileReader2);
               Log.w("ModelLoader", "model3d = new Model3D(model2, " +
modelName2patternName(modelFileName2) + ".patt");
                model3d2 = new Model3D(model2,
modelName2patternName(modelFileName2) + ".patt");
                } else {
                Log.w("ModelLoader", "no file reader");
                }
                String modelFileName3 = "towergreen.obj";
                BufferedReader fileReader3 =
fileUtil.getReaderFromName(modelFileName3);
                if(fileReader3 != null) {
                model3 = parser.parse("towergreen", fileReader3);
                Log.w("ModelLoader", "model3d = new Model3D(model3, " +
modelName2patternName(modelFileName3) + ".patt");
                model3d3 = new Model3D(model3,
modelName2patternName(modelFileName3) + ".patt");
                } else {
                Log.w("ModelLoader", "no file reader");
                }
                String modelFileName4 = "tower.obj";
                BufferedReader fileReader4 =
fileUtil.getReaderFromName(modelFileName4);
                if(fileReader4 != null) {
                model4 = parser.parse("tower", fileReader4);
                Log.w("ModelLoader", "model3d = new Model3D(model4, " +
modelName2patternName(modelFileName4) + ".patt");
                model3d4 = new Model3D(model4,
modelName2patternName(modelFileName4) + ".patt");
                } else {
                Log.w("ModelLoader", "no file reader");
                }
                String modelFileName5 = "plant.obj";
                BufferedReader fileReader5 =
fileUtil.getReaderFromName(modelFileName5);
                if(fileReader5 != null) {
                model5 = parser.parse("Plant", fileReader5);
                Log.w("ModelLoader", "model3d = new Model3D(model5, " +
modelName2patternName(modelFileName5) + ".patt");
                model3d5 = new Model3D(model5,
modelName2patternName(modelFileName5) + ".patt");
                } else {
                Log.w("ModelLoader", "no file reader");
                }
                }
                if(Config.DEBUG)
                Debug.stopMethodTracing();
                } catch (IOException e) {
                    e.printStackTrace();
                } catch (ParseException e) {
                    e.printStackTrace();
                }
                }
        return null;
        }
        @Override
        protected void onPostExecute(Void result) {
            super.onPostExecute(result);
            waitDialog.dismiss();

            try {
                if(model3d!=null) {
                    artoolkit.registerARObject(model3d);
                    artoolkit.registerARObject(model3d2);
                    artoolkit.registerARObject(model3d3);
                    artoolkit.registerARObject(model3d4);
                    artoolkit.registerARObject(model3d5);
                }
            } catch (AndARException e) {
                e.printStackTrace();
            }
            startPreview();
        }
    }
```

TakeAsyncScreenshot Inner Class

In the TakeAsyncScreenshot inner class we call upon AndAR's inbuilt ability to take a screenshot.

Listing 8-30. *TakeAsyncScreenshot*

```
    class TakeAsyncScreenshot extends AsyncTask<Void, Void, Void> {

        private String errorMsg = null;

        @Override
        protected Void doInBackground(Void... params) {
            Bitmap bm = takeScreenshot();
            FileOutputStream fos;
            try {
                fos = new FileOutputStream("/sdcard/AndARScreenshot"+new
Date().getTime()+".png");
                bm.compress(CompressFormat.PNG, 100, fos);
                fos.flush();
                fos.close();
            } catch (FileNotFoundException e) {
                errorMsg = e.getMessage();
                e.printStackTrace();
            } catch (IOException e) {
                errorMsg = e.getMessage();
                e.printStackTrace();
            }
            return null;
        }

        protected void onPostExecute(Void result) {
            if(errorMsg == null)
                Toast.makeText(ModelViewer.this,
getResources().getText(R.string.screenshotsaved), Toast.LENGTH_SHORT ).show();
            else
                Toast.makeText(ModelViewer.this,
getResources().getText(R.string.screenshotfailed)+errorMsg, Toast.LENGTH_SHORT
).show();
        };
    }
}
```

Parsing .mtl files

Next we have a very important class, `MtlParser.java`. This class is responsible for parsing the .mtl files that accompany the .obj files of our models.

Listing 8-31. *MtlParser.java*

```
public class MtlParser {

    private BaseFileUtil fileUtil;
```

```
public MtlParser(Model model, BaseFileUtil fileUtil) {
    this.fileUtil = fileUtil;
}

public void parse(Model model, BufferedReader is) {
    Material curMat = null;
    int lineNum = 1;
    String line;
    try {
        for (line = is.readLine();
        line != null;
        line = is.readLine(), lineNum++)
        {
            line = Util.getCanonicalLine(line).trim();
            if (line.length() > 0) {
                if (line.startsWith("newmtl ")) {
                    String mtlName = line.substring(7);
                    curMat = new Material(mtlName);
                    model.addMaterial(curMat);
                } else if(curMat == null) {
                } else if (line.startsWith("# ")) {
                } else if (line.startsWith("Ka ")) {
                     String endOfLine = line.substring(3);
                     curMat.setAmbient(parseTriple(endOfLine));
                } else if (line.startsWith("Kd ")) {
                    String endOfLine = line.substring(3);
                    curMat.setDiffuse(parseTriple(endOfLine));
                } else if (line.startsWith("Ks ")) {
                    String endOfLine = line.substring(3);
                    curMat.setSpecular(parseTriple(endOfLine));
                } else if (line.startsWith("Ns ")) {
                    String endOfLine = line.substring(3);
                    curMat.setShininess(Float.parseFloat(endOfLine));
                } else if (line.startsWith("Tr ")) {
                    String endOfLine = line.substring(3);
                    curMat.setAlpha(Float.parseFloat(endOfLine));
                } else if (line.startsWith("d ")) {
                    String endOfLine = line.substring(2);
                    curMat.setAlpha(Float.parseFloat(endOfLine));
                } else if(line.startsWith("map_Kd ")) {
                    String imageFileName = line.substring(7);
                    curMat.setFileUtil(fileUtil);
                    curMat.setBitmapFileName(imageFileName);
                } else if(line.startsWith("mapKd ")) {
                    String imageFileName = line.substring(6);
                    curMat.setFileUtil(fileUtil);
                    curMat.setBitmapFileName(imageFileName);
                 }
            }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static float[] parseTriple(String str) {
        String[] colorVals = str.split(" ");
        float[] colorArr = new float[]{
                Float.parseFloat(colorVals[0]),
                Float.parseFloat(colorVals[1]),
                Float.parseFloat(colorVals[2])};
        return colorArr;
    }
}
```

The class is not very complex. Basically, the class reads the entire file line by line, and works on every line by seeing what it starts with. The absolute first condition makes sure that the line is actually a line, and not an empty one. After that, the nested `if else` statements are there.

All the conditions mentioned from now on are from the nested statements, unless mentioned otherwise. The first such condition checks to see whether the line is the first one by seeing if it begins with “newmtl“.

```
                    if (line.startsWith("newmtl ")) {
                        String mtlName = line.substring(7);
                        curMat = new Material(mtlName);
                        model.addMaterial(curMat);
```

The next condition makes sure that our current material isn't null.

```
                    } else if(curMat == null) {
```

The third one is used to ignore comments because they start with a “#” in .mtl files.

```
                    } else if (line.startsWith("# ")) {
```

The fourth condition sees whether the line specifies the ambient light for our model, and sets it if it does.

```
                    } else if (line.startsWith("Ka ")) {
                         String endOfLine = line.substring(3);
                         curMat.setAmbient(parseTriple(endOfLine));
```

The fifth condition sees whether the line specifies the diffuse light for our model and sets it if it does.

```
                    } else if (line.startsWith("Kd ")) {
                        String endOfLine = line.substring(3);
                        curMat.setDiffuse(parseTriple(endOfLine));
```

The sixth condition sees whether the line specifies the specular light for our model and sets it if it does.

```
                    } else if (line.startsWith("Ks ")) {
                        String endOfLine = line.substring(3);
                        curMat.setSpecular(parseTriple(endOfLine));
```

The seventh condition checks whether the line specifies the shininess for our model and sets it if it does.

```
                    } else if (line.startsWith("Ns ")) {
                        String endOfLine = line.substring(3);
                        curMat.setShininess(Float.parseFloat(endOfLine));
```

The eighth and ninth conditions check whether the line specifies the alpha values for our model and sets it if it does.

```
                    } else if (line.startsWith("Tr ")) {
                        String endOfLine = line.substring(3);
                        curMat.setAlpha(Float.parseFloat(endOfLine));
                    } else if (line.startsWith("d ")) {
                        String endOfLine = line.substring(2);
                        curMat.setAlpha(Float.parseFloat(endOfLine));
```

The tenth and eleventh conditions check whether the line specifies the image for this model and sets it if it does.

```
                    } else if(line.startsWith("map_Kd ")) {
                        String imageFileName = line.substring(7);
                        curMat.setFileUtil(fileUtil);
                        curMat.setBitmapFileName(imageFileName);
                    } else if(line.startsWith("mapKd ")) {
                        String imageFileName = line.substring(6);
                        curMat.setFileUtil(fileUtil);
                        curMat.setBitmapFileName(imageFileName);
```

The `catch` statement in the end of the methods catches the `IOException` that will be triggered by something like the file not being found or if the file has unfavorable permissions.

```
            } catch (IOException e) {
                e.printStackTrace();
            }
```

The float `parseTriple()` is repeatedly called to help in parsing the file.

```
    private static float[] parseTriple(String str) {
        String[] colorVals = str.split(" ");
        float[] colorArr = new float[]{
            Float.parseFloat(colorVals[0]),
            Float.parseFloat(colorVals[1]),
            Float.parseFloat(colorVals[2])};
        return colorArr;
    }
```

Parsing the .obj files

Next up is another very important class, `ObjParser.java`. It parses the .obj files, at least to an extent. It does not support the full .obj specification. It supports the following:

- Vertices
- Vertice normals
- Texture Coordinates
- Basic Materials
- Limited Texture Support (through the map_Kd, no options)
- Faces (may not omit the face normal)

This kind of support is enough for our models.

Listing 8-32. *ObjParser.java*

```
public class ObjParser {
   private final int VERTEX_DIMENSIONS = 3;
   private final int TEXTURE_COORD_DIMENSIONS = 2;

   private BaseFileUtil fileUtil;

   public ObjParser(BaseFileUtil fileUtil) {
      this.fileUtil = fileUtil;
   }

   public Model parse(String modelName, BufferedReader is) throws IOException,
ParseException {
      ArrayList<float[]> vertices = new ArrayList<float[]>(1000);
      ArrayList<float[]> normals = new ArrayList<float[]>(1000);
      ArrayList<float[]> texcoords = new ArrayList<float[]>();

      Model model = new Model();
      Group curGroup = new Group();
      MtlParser mtlParser = new MtlParser(model,fileUtil);
      SimpleTokenizer spaceTokenizer = new SimpleTokenizer();
      SimpleTokenizer slashTokenizer = new SimpleTokenizer();
      slashTokenizer.setDelimiter("/");
      String line;
      int lineNum = 1;
      for (line = is.readLine();
      line != null;
      line = is.readLine(), lineNum++)
      {
      if (line.length() > 0) {
         if (line.startsWith("#")) {
         } else if (line.startsWith("v ")) {
            String endOfLine = line.substring(2);
            spaceTokenizer.setStr(endOfLine);
            vertices.add(new float[]{
               Float.parseFloat(spaceTokenizer.next()),
               Float.parseFloat(spaceTokenizer.next()),
               Float.parseFloat(spaceTokenizer.next())});
            }
         else if (line.startsWith("vt ")) {
            String endOfLine = line.substring(3);
            spaceTokenizer.setStr(endOfLine);
            texcoords.add(new float[]{
               Float.parseFloat(spaceTokenizer.next()),
               Float.parseFloat(spaceTokenizer.next())});
            }
         else if (line.startsWith("f ")) {
            String endOfLine = line.substring(2);
            spaceTokenizer.setStr(endOfLine);
            int faces = spaceTokenizer.delimOccurCount()+1;
            if(faces != 3) {
               throw new ParseException(modelName, lineNum, "only triangle faces
are supported");
               }
            for (int i = 0; i < 3; i++) {
               String face = spaceTokenizer.next();
               slashTokenizer.setStr(face);
               int vertexCount = slashTokenizer.delimOccurCount()+1;
               int vertexID=0;
               int textureID=-1;
               int normalID=0;
               if(vertexCount == 2) {
                  vertexID = Integer.parseInt(slashTokenizer.next())-1;
                  normalID = Integer.parseInt(slashTokenizer.next())-1;
                  throw new ParseException(modelName, lineNum, "vertex normal
needed.");
               } else if(vertexCount == 3) {
               vertexID = Integer.parseInt(slashTokenizer.next())-1;
               String texCoord = slashTokenizer.next();
               if(!texCoord.equals("")) {
                  textureID = Integer.parseInt(texCoord)-1;
               }
```

```
                normalID = Integer.parseInt(slashTokenizer.next())-1;
                } else {
                throw new ParseException(modelName, lineNum, "a faces needs
reference a vertex, a normal vertex and optionally a texture coordinate per
vertex.");
                }
                float[] vec;
                try {
                   vec = vertices.get(vertexID);
                } catch (IndexOutOfBoundsException ex) {
                throw new ParseException(modelName, lineNum, "non existing vertex
referenced.");
                }
             if(vec==null)
             throw new ParseException(modelName, lineNum, "non existing vertex
referenced.");
             for (int j = 0; j < VERTEX_DIMENSIONS; j++)
             curGroup.groupVertices.add(vec[j]);
             if(textureID != -1) {
             try {
             vec = texcoords.get(textureID);
             } catch (IndexOutOfBoundsException ex) {
             throw new ParseException(modelName, lineNum, "non existing texture
coord referenced.");
                    }
             if(vec==null)
             throw new ParseException(modelName, lineNum,  "non existing texture
coordinate referenced.");
             for (int j = 0; j < TEXTURE_COORD_DIMENSIONS; j++)
                curGroup.groupTexcoords.add(vec[j]);
             }
             try {
             vec = normals.get(normalID);
             } catch (IndexOutOfBoundsException ex) {
             throw new ParseException(modelName, lineNum, "non existing normal
vertex referenced.");
             }
             if(vec==null)
             throw new ParseException(modelName, lineNum, "non existing normal
vertex referenced.");
             for (int j = 0; j < VERTEX_DIMENSIONS; j++)
                curGroup.groupNormals.add(vec[j]);
             }
             }
       else if (line.startsWith("vn ")) {
          String endOfLine = line.substring(3);
          spaceTokenizer.setStr(endOfLine);
          normals.add(new float[]{
             Float.parseFloat(spaceTokenizer.next()),
             Float.parseFloat(spaceTokenizer.next()),
            Float.parseFloat(spaceTokenizer.next())});
        } else if (line.startsWith("mtllib ")) {
          String filename = line.substring(7);
          String[] files = Util.splitBySpace(filename);
          for (int i = 0; i < files.length; i++) {
          BufferedReader mtlFile = fileUtil.getReaderFromName(files[i]);
          if(mtlFile != null)
            mtlParser.parse(model, mtlFile);
          }
        } else if(line.startsWith("usemtl ")) {
          if(curGroup.groupVertices.size()>0) {
            model.addGroup(curGroup);
            curGroup = new Group();
          }
          curGroup.setMaterialName(line.substring(7));
        } else if(line.startsWith("g ")) {
          if(curGroup.groupVertices.size()>0) {
            model.addGroup(curGroup);
            curGroup = new Group();
          }
        }
      }
    }
    if(curGroup.groupVertices.size()>0) {
      model.addGroup(curGroup);
    }
    Iterator<Group> groupIt = model.getGroups().iterator();
    while (groupIt.hasNext()) {
      Group group = (Group) groupIt.next();
      group.setMaterial(model.getMaterial(group.getMaterialName()));
    }
    return model;
    }
    }
```

This file goes through the .obj files line by line. There are a series of if else blocks that parse the file. The following happens at each line:

1. Comments (starting with a #) are ignored
2. Vertices (starting with v) are added to the vertices `ArrayList`.
3. Texture Coordinates (starting with vt) are added to the texcoords `ArrayList`.
4. Faces (starting with f) are added to groups.
5. Normals (starting with vn) are added to the normals `ArrayList`.
6. Corresponding .mtl files (starting with `mtllib`) are parsed.

7. New materials are added and corresponding groups created (starting with usemtl).
8. New groups are created (starting with g).

Upon completion, it returns a model.

ParseException

Next is the ParseException.java class that is the ParseException that is repeatedly thrown in ObjParser.java. It is a custom exception that we have written to allow us to easily put across problems that occur during the parsing process.

Listing 8-33. *ParseException.java*

```
public class ParseException extends Exception {
   public ParseException(String file,int lineNumber, String msg) {
      super("Parse error in file "+file+"on line "+lineNumber+":"+msg);
   }
}
```

It's very simple; it just outputs a message, filling in the message specific details via parameters.

Rendering

Next is the Renderer.java file, which handles a lot of the drawing work for our graphic, including some tricky 3D stuff.

Listing 8-34. *Renderer.java*

```
public class Renderer implements GLSurfaceView.Renderer {

   private final Vector<Model3D> models;
   private final Vector3D cameraPosition = new Vector3D(0, 3, 50);
   long frame,time,timebase=0;
   public Renderer(Vector<Model3D> models) {
      this.models = models;
   }
   public void addModel(Model3D model) {
      if(!models.contains(model)) {
         models.add(model);
      }
   }
   public void onDrawFrame(GL10 gl) {
      if(ModelViewer.DEBUG) {
            frame++;
            time=System.currentTimeMillis();
            if (time - timebase > 1000) {
               Log.d("fps: ", String.valueOf(frame*1000.0f/(time-timebase)));
               timebase = time;
               frame = 0;
            }
         }
         gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
         gl.glLoadIdentity();
         GLU.gluLookAt(gl, cameraPosition.x, cameraPosition.y, cameraPosition.z,
               0, 0, 0, 0, 1, 0);
         for (Iterator<Model3D> iterator = models.iterator(); iterator.hasNext();)
{
            Model3D model = iterator.next();
            model.draw(gl);
         }
      }

      public void onSurfaceChanged(GL10 gl, int width, int height) {
         gl.glViewport(0,0,width,height);
         gl.glMatrixMode(GL10.GL_PROJECTION);
         gl.glLoadIdentity();
         GLU.gluPerspective(gl, 45.0f, ((float)width)/height, 0.11f, 100f);
         gl.glMatrixMode(GL10.GL_MODELVIEW);
         gl.glLoadIdentity();
      }

      public void onSurfaceCreated(GL10 gl, EGLConfig config) {
         gl.glClearColor(1,1,1,1);

         gl.glClearDepthf(1.0f);
         gl.glEnable(GL10.GL_DEPTH_TEST);
         gl.glDepthFunc(GL10.GL_LEQUAL);

         gl.glEnable(GL10.GL_TEXTURE_2D);

         gl.glShadeModel(GL10.GL_SMOOTH);
         gl.glDisable(GL10.GL_COLOR_MATERIAL);
         gl.glEnable(GL10.GL_BLEND);
         gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);
         gl.glEnable(GL10.GL_LIGHTING);
         float[] ambientlight = {.6f, .6f, .6f, 1f};
         float[] diffuselight = {1f, 1f, 1f, 1f};
         float[] specularlight = {1f, 1f, 1f, 1f};
         gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_AMBIENT,
MemUtil.makeFloatBuffer(ambientlight));
         gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_DIFFUSE,
MemUtil.makeFloatBuffer(diffuselight));
```

```
      gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_SPECULAR,
MemUtil.makeFloatBuffer(specularlight));
      gl.glEnable(GL10.GL_LIGHT0);

      for (Iterator<Model3D> iterator = models.iterator(); iterator.hasNext();)
{
         Model3D model = iterator.next();
         model.init(gl);
      }
   }
}
```

The constructor stores the models that are passed to it locally. The `addModel()` method adds a model to our list. The `onDrawFrame()` logs the frame rate if the app is set to Debug mode. Regardless of the mode of the app, the `onDrawFrame()` method also updates what the user is shown. The `onSurfaceChanged()` method is called whenever the surface changes and applies changes related to the new width and height. The `onSurfaceCreated()` method does the initial setup work when the surface is first created.

SDCardFileUtil.java

Next is the `SDCardFileUtil.java`. This is an extension of `BaseFileUtil` and handles the reading of the files.

Listing 8-35. *SDCardFileUtil.java*

```
public class SDCardFileUtil extends BaseFileUtil {
   public BufferedReader getReaderFromName(String name) {
   if (baseFolder != null) {
      try {
      return new BufferedReader(new FileReader(new File(baseFolder, name)));
      } catch (FileNotFoundException e) {
      return null;
      }
      } else {
      try {
      return new BufferedReader(new FileReader(new File(name)));
      } catch (FileNotFoundException e) {
      return null;
      }
      }
   }

   public Bitmap getBitmapFromName(String name) {
      if (baseFolder != null) {
         String path = new File(baseFolder,name).getAbsolutePath();
```

```
            return BitmapFactory.decodeFile(path);
         } else {
            return BitmapFactory.decodeFile(name);
         }
      }
   }
```

The first method attempts to get a `BufferedReader` by the file name, and the second one tries to get a Bitmap by the name.

SimpleTokenizer.java

Next is the `SimpleTokenizer.java class, which` class is used as a `Tokenizer` in many places to delimit strings.

Listing 8-36. *SimpleTokenizer.java*

```
public class SimpleTokenizer {
   String str = "";
   String delimiter = " ";
   int delimiterLength = delimiter.length();
   int i =0;
   int j =0;
   public final String getStr() {
      return str;
   }
   public final void setStr(String str) {
      this.str = str;
      i =0;
      j =str.indexOf(delimiter);
   }
   public final String getDelimiter() {
      return delimiter;
   }
   public final void setDelimiter(String delimiter) {
      this.delimiter = delimiter;
      delimiterLength = delimiter.length();
}
   public final boolean hasNext() {
      return j >= 0;
   }
   public final String next() {
      if(j >= 0) {
         String result = str.substring(i,j);
         i = j + 1;
         j = str.indexOf(delimiter, i);
         return result;
      } else {
            return str.substring(i);
        }
    }
    public final String last() {
        return str.substring(i);
    }

    public final int delimOccurCount() {
        int result = 0;
        if (delimiterLength > 0) {
            int start = str.indexOf(delimiter);
            while (start != -1) {
                result++;
                start = str.indexOf(delimiter, start + delimiterLength);
            }
        }
        return result;
    }
}
```

It's a simple class. Everything is from the standard Java package, and nothing needs to be imported. Strictly speaking, there is no Android API used in this class. You could copy paste it into a normal Java project and it would work perfectly.

Util.java

Next is `Util.java`. This class optimizes our .obj files so that they can be parsed faster next time.

Listing 8-37. *Util.java*

```
public class Util {
    private static final Pattern trimWhiteSpaces = Pattern.compile("[\\s]+");
    private static final Pattern removeInlineComments = Pattern.compile("#");
    private static final Pattern splitBySpace = Pattern.compile(" ");

    public static final String getCanonicalLine(String line) {
        line = trimWhiteSpaces.matcher(line).replaceAll(" ");
        if(line.contains("#")) {
            String[] parts = removeInlineComments.split(line);
            if(parts.length > 0)
                line = parts[0];
        }
        return line;
    }
    public static String[] splitBySpace(String str) {
        return splitBySpace.split(str);
    }

    public static void trim(BufferedReader in, BufferedWriter out) throws
IOException {
        String line;
        out.write("#trimmed\n");
        for (line = in.readLine();
        line != null;
        line = in.readLine()) {
            line = getCanonicalLine(line);
            if(line.length()>0) {
                out.write(line.trim());
                out.write('\n');
            }
        }
        in.close();
        out.close();
    }

    public final static List<String> fastSplit(final String text, char
separator, final boolean emptyStrings) {
        final List<String> result = new ArrayList<String>();

        if (text != null && text.length() > 0) {
            int index1 = 0;
            int index2 = text.indexOf(separator);
            while (index2 >= 0) {
                String token = text.substring(index1, index2);
                result.add(token);
                index1 = index2 + 1;
                index2 = text.indexOf(separator, index1);
            }

            if (index1 < text.length() - 1) {
                result.add(text.substring(index1));
            }
        }

        return result;
    }

}
```

This is, once again, standard Java. It simply removes whitespaces, inline comments etc. for faster parsing next time.

3D Vectors

Now we come to that last of the Java files, `Vector3D.java`. This file works with a three-dimensional vector. This class is used to position our virtual OpenGL camera. This camera is very different from the hardware camera we've been using all along. It is a virtual camera from which we see our model.

Listing 8-38. *Vector3D.java*

```
public class Vector3D implements Serializable {
    public float x=0;
    public float y=0;
    public float z=0;

    public Vector3D(float x, float y, float z) {
        super();
        this.x = x;
        this.y = y;
        this.z = z;
    }
    public float getX() {
        return x;
    }

    public void setX(float x) {
        this.x = x;
    }

    public float getY() {
        return y;
    }

    public void setY(float y) {
        this.y = y;
    }

    public float getZ() {
        return z;
    }

    public void setZ(float z) {
        this.z = z;
    }

}
```

All the methods are getting x, y, or z or setting them. That's all there is to it.

XML Files

Now that we are done with the all the Java files, we can move on to the XML files.

Strings.xml

Let's start with the `strings.xml`.

Listing 8-39. *strings.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">PAAR Chapter 8</string>
<string name="select_model_file">Select a model file:</string>
<string name="android_markt_not_avail">Android market is not available, you need
install to OI File manager manually.</string>
<string name="pickfile_intent_required">You need to install the OI File Manager
in order to use this application.</string>
<string name="file_doesnt_exist">This file doesn\'t exist!</string>
<string name="unknown_file_type">Unknown file type!</string>
<string name="rotate">rotate</string>
<string name="translate">translate</string>
<string name="scale">scale</string>

<string name="loading">Loading. Please wait...</string>
<string name="app_description">AndAR Model Viewer allows you to view wavefront
obj models on an Augmented Reality marker.</string>
<string name="take_screenshot">Take a screenshot</string><string
name="screenshotsaved">Screenshot saved!</string><string
name="screenshotfailed">Failed to save screenshot: </string>
<string name="choose_custom_model">Select a model file</string>
<string name="instructions">Instructions</string>
<string name="wrong_file">Select an obj model file. (.obj)</string>
<string name="choose_a_model">Choose a model:</string>
<string name="help">Help:</string>
<string name="custom_model">Custom model:</string>
<string name="help_file">index.html</string>
</resources>
```

This file contains all the predefined strings for our app. Each string’s contents and name should provide you with an exact description of what they do.

Layout for the Rows

Now let's see the layout files. We have `choose_model_row.xml`, which is used in the `ModelChooser`.

Listing 8-40. *choose_model_row.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="6dip">
<ImageView
        android:id="@+id/choose_model_row_icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="6dip"
        android:src="@drawable/ic_launcher" />
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/choose_model_row_text"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:gravity="center_vertical"
    android:paddingLeft="6dip"
    android:minHeight="?android:attr/listPreferredItemHeight"
/>
</LinearLayout>
```

We have an ImageView for the icon, and a TextView for the name put together side by side. That’s the entire layout for our rows.

instructions_layout.xml

Next is the `instructions_layout.xml`, which is the XML file behind our instructions Activity.

Listing 8-41. *instructions_layout.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"
    >
<WebView
        android:id="@+id/instructions_webview"
        android:layout_width="fill_parent"
        android:layout_height="0dip"
        android:layout_weight="1"
        />
</LinearLayout>
```

Here we have a linear layout that is completely filled by a WebView to display the instructions HTML file.

List Header

Next we have `list_header.xml`, which is, as the name may have given away, the header for our list.

Listing 8-42. *list_header.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/list_header_title"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:paddingTop="2dip"
    android:paddingBottom="2dip"
    android:paddingLeft="5dip"
    style="?android:attr/listSeparatorTextViewStyle" />
```

main.xml

Finally we have `main.xml`, which is used to display the information.

Listing 8-43. *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"
    >
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/InfoText">
</TextView>
</LinearLayout>
```

HTML Help File

This brings us to the end of the XML files. Now all that is left is one HTML file, which is our instructions file. It is located in /assets/help/.

Listing 8-44. *index.html*

```
<html>
    <head>
        <style type="text/css">
        h1 {font-size:20px;}
        h2 {font-size:16px;}
         </style>
    </head>
    <body>
          <p>This is the 3rd example application from the book Pro Android
Augmented Reality by Raghav Sood, published by Apress. It projects models on a
marker.
         You may either use the internal models or supply your own models in the
<a href="http://en.wikipedia.org/wiki/Obj">wavefront obj format.</a></p>
           <ol>
                         <li><a href="#firststeps">First steps</a></li>
                         <li><a href="#screenshot">Taking a screenshot</a></li>
                         <li><a href="#transforming">Transforming the
model</a></li>
                         <li><a href="#custommodels">Custom models</a></li>
          </ol>
          <h2><a name="firststeps">First steps</a></h2>
          <ul>
                         <li>First print out the marker, upon which the models
will be projected. The marker is located in the assets folder of the project
source.</li>
          </ul>
          <ul>
                         <li>Select one of the internal models.</li>
                         <li>The app will start loading the model.</li>
                         <li>After it has finished, you will see a live video
stream from the camera.</li>
                         <li>Now point the camera to the marker, you should see
the model you selected.</li>
          </ul>
          <h2><a name="screenshot">Taking a screenshot</a></h2>
          <ul>
                          <li>First press the menu key.</li>
                          <li>Next press the button "Take a screenshot".</li>
                          <li>The application will now process the image. It
will notfiy you, when it's finished.</li>
```

```
                          <li>The screenshot you just took can be found in the
root folder of your sd-card. It will be named something like
<i>AndARScreenshot1234.png</i></li>
          </ul>
          <h2><a name="transforming">Transforming the model</a></h2>
          <ul>
                          <li>Press the menu key and select the desired
transformation mode. You may either scale, rotate or translate the model.</li>
                          <li>Scale: Slide you finger up and down the touch
screen. This will enlarge and shorten the model, respectively.</li>
                          <li>Rotate: Slide your finger horizontally and
vertically, this will rotate your model correspondingly. </li>
                          <li>Translate: Slide your finger horizontally and
vertically, this will translate your model correspondingly. </li>
          </ul>
           <h2><a name="custommodels">Custom models</a></h2>
            The application is capable of showing custom wavefront obj models.
Most 3d modelling software out there can export this format(e.g. 3ds max,
Blender).
            There are currently some restrictions to the models:
            <ul>
            <li>Every face must have normals specified</li>
            <li>The object must be triangulated, this means exactly 3 vertices
per face.</li>
            <li>Basic materials and textures are supported.</li>
            </ul>
            E.g. when exporting a model from blender make sure you check
<i>Normals</i> and <i>Triangulate</i>.
            <h2>Attribution</h2>
            <ul>
            <li>This app contains code developed by the Android Open Source
Project, released under the Apache License.</li>
            <li>This app contains models from <a
href="http://resources.blogscopia.com/modelos_en.html">resources.blogscopia.com<
/a> released under the <a
href="http://creativecommons.org/licenses/by/3.0/">Creative Commons 3.0 Unported
license</a>, see also: <a href="http://www.scopia.es">www.scopia.es</a>.</li>
             <li>This product includes software developed by the <a
href="http://mij.oltrelinux.com/devel/simclist/">SimCList  project</a>.</li>
             <li>This project includes code from the <a
href=http://code.google.com/p/andar/AndAR</a> project.</li>
             </ul>
    </body>
</html>
```

Completed App

That's all the files you need that can be displayed in the book! Download this book’s source from this book's page on Apress or from the GitHub repo at http://github.com/RaghavSood/ProAndroidAugmentedReality to get the image, .patt and .obj + .mtl files that are required for the project to run smoothly.

Figures 8-3 and 8-4 show the app in action.

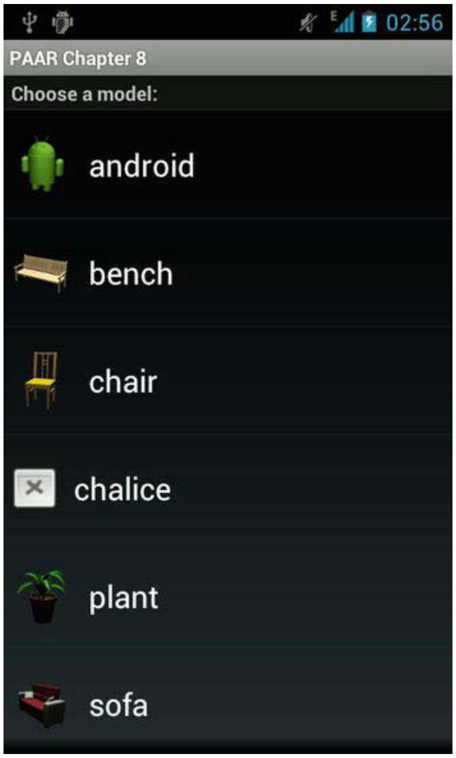


Figure 8-3. *Opening screen of the app.*

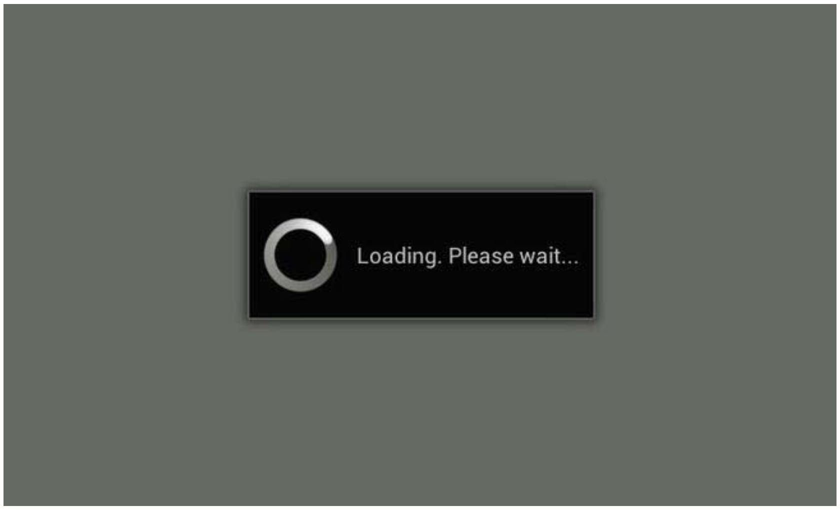

Loading. Please wait...

Figure 8-4. *Loading the Android model.*

Summary

In this chapter, we created a fully functional 3D AR object viewer using the AndAR framework. Our app has the capability to load both internal and external models; display them in 3D; and allow the user to resize, rotate, and reposition them. In the next chapter, we will build an AR app that explores the social and gaming capabilities of AR.