Leia CNSDK For Simple Java Android Applications

Purpose

Provide a tutorial that shows how to create a simple lightfield application in Java using the Leia CNSDK.

What is the Leia CNSDK?

The Leia CNSDK is middleware in the form of C++ and Java libraries and headers. It was created for application developers targeting Leia lightfield devices such as the LumePad 2, and provides a simple API that performs all necessary eye-tracking and graphics.

While not absolutely required, the CNSDK will allow you to harness the ability of your lightfield device in a simple manner. Any developer attempting to create Leia lightfield 3D experiences without CNSDK will encounter significant technical challenges. We therefore highly suggest using CNSDK for your application.

The CNSDK provides the following mechanisms to access functionality depending on your situation:

1. For Windows C++ applications, use classes `ILeiaSDK` and `IThreadedInterlacer`.
2. For simple Android Java applications, use Leia-provided Java classes such as `LeiaSDK`, `InterlacedSurfaceView`, and `InputViewsAsset`.
3. For advanced Android Java applications, use Java JNI to access C++ classes directly.

In this tutorial we will focus on simple Java applications for Android.

Project Setup

The provided sample project is a simple Java activity created in Android Studio 2021.2.1. It was created in the following way:

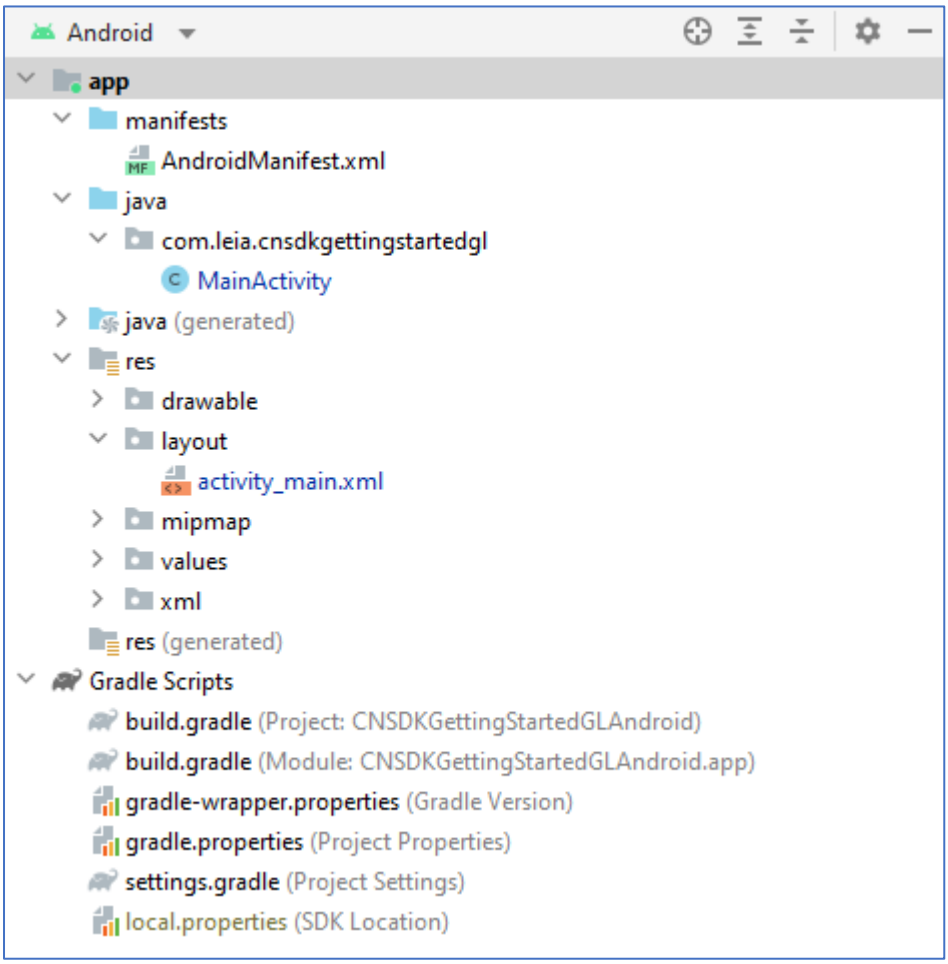
1. Start Android Studio and create an empty Java project.
2. Add empty MainActivity.java file and modify AndroidManifest.xml to include the activity. The activity block in the manifest file should look similar to the block below:

```
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

3. Add a layout file by clicking File/New/XML/Layout XML File and name it activity_main.xml.
4. Add the Leia CNSDK aar file to the dependencies block of the application build.gradle file. This should look similar to the block below, depending on the location of the aar file relative to the project.

```
dependencies {
    ...
    implementation files("../CNSDK/sdk-final-aar-debug.aar")
}
```

Once setup, your project should look like this:



Project Implementation

Now that the project is setup, we need to create a layout and implement the MainActivity.

The project will do just one thing – display an interlaced stereo image.

First, we need to setup the layout for the application. To do this, double-click the activity_main.xml file and add a view based on `com.leia.sdk.views.InterlacedSurfaceView` with id `interlacedView`.

Next, we need to implement MainActivity.java. This class implements `LeiaSDK.Delegate` and is where all Leia CNSDK functionality is located. We only need to implement the following seven methods:

```
void onCreate(Bundle savedInstanceState)
void didInitialize(LeiaSDK leiaSDK)
```

```
void onResume()
void onPause()
void onFaceTrackingStarted(LeiaSDK leiaSDK)
void onFaceTrackingStopped(LeiaSDK leiaSDK)
void onFaceTrackingFatalError(LeiaSDK leiaSDK)
```

The methods above are all trivial except `onCreate`, which is where all the Leia logic resides. Please see the sample code for the other methods.

There are 3 parts to `onCreate` – setting the layout view, initializing the Leia SDK, and filling the `InterlacedSurfaceView` with content.

To set the layout view, simply call:

```
setContentView(R.layout.activity_main);
```

Next, we need to initialize the Leia SDK and enable face-tracking. This is done as follows:

```
try {
    LeiaSDK.InitArgs initArgs = new LeiaSDK.InitArgs();
    initArgs.delegate = this;
    initArgs.platform.logLevel = LogLevel.Trace;
    initArgs.platform.context = getApplicationContext();
    initArgs.faceTrackingServerLogLevel = LogLevel.Trace;
    initArgs.enableFaceTracking = true;
    leiaSDK = LeiaSDK.createSDK(initArgs);
} catch (Exception e) {
    Log.e(LogTag, String.format("Error: %s", e.toString()));
    e.printStackTrace();
}
```

Finally, we need to fill the `InterlacedSurfaceView` with content. We create an `InputViewsAsset` object and load a stereo image into it. In this particular case, we are setting a built-in test image but you can set the image of your choice.

```
interlacedView = findViewById(R.id.interlacedView);
InputViewsAsset viewsAsset = new InputViewsAsset();
viewsAsset.LoadBitmapFromPathIntoSurface("image_0.jpg", this, null);
interlacedView.setViewAsset(viewsAsset);
```

Now you are ready to run the application. It should load and display a stereo image showing a beer glass.

Enhancements

The `InterlacedSurfaceView` class provides additional methods to help control the output. In this section we will discuss a method that changes the perceived 3D effect, called *reconvergence,* and a method to apply 2D scaling,

Your application might want to provide a user-adjustable reconvergence value. This will affect the 3d strength and correctness of the final interlaced image. Use the `setReconvergenceAmount` method below to achieve this:

```
interlacedView.setReconvergenceAmount(0.02f);
```

This method changes the separation of the stereo images. It affects the 3d strength and too much reconvergence will break the 3d effect completely.

Scaling the interlaced output is also a commonly used feature. Depending on how you wish to display your output, there are several scaling options available. You can use the following method for scaling:

```
interlacedView.setScaleType(ScaleType.FIT_CENTER);
```

There are four scaling types you can use:

- FILL: Stretch result to the view (default)
- FIT_CENTER: Scale result to fit view while maintaining aspect ratio
- CROP_FILL: Crops result to fit view while maintaining aspect ratio
- CROP_FIT_SQUARE: Crops result to fit view with a 1:1 aspect ratio