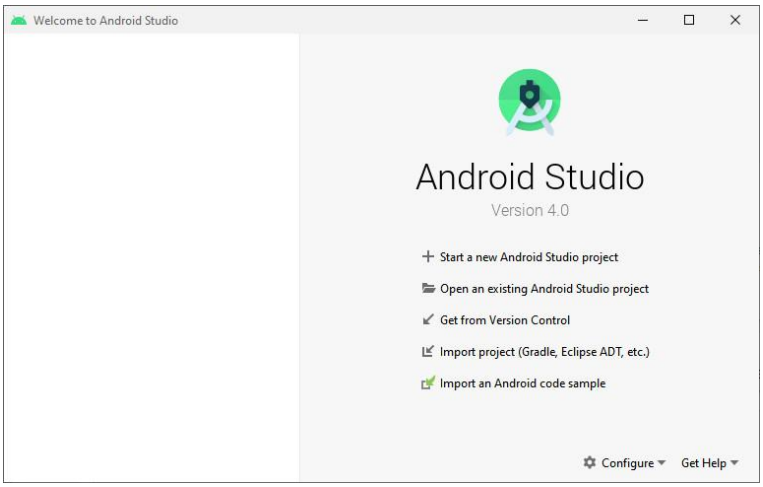
Android - OpenGL ES 1 - Tutorial 1

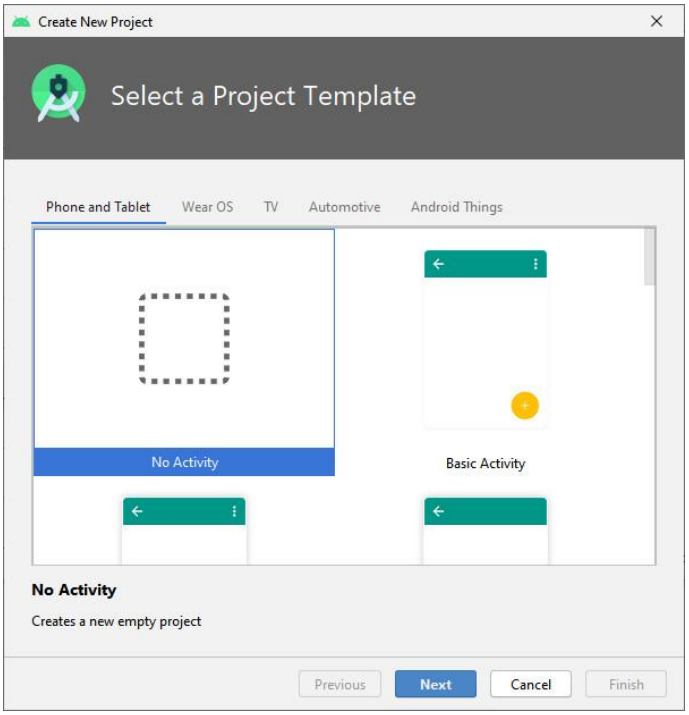
Basic 2D Project

OpenGL for Embedded Systems (OpenGL ES or GLES) is a subset of the OpenGL computer graphics rendering application programming interface (API) for rendering 2D and 3D computer graphics such as those used by video games, typically hardware-accelerated using a graphics processing unit (GPU). It is designed for embedded systems like smartphones, tablet computers, video game consoles and PDAs.

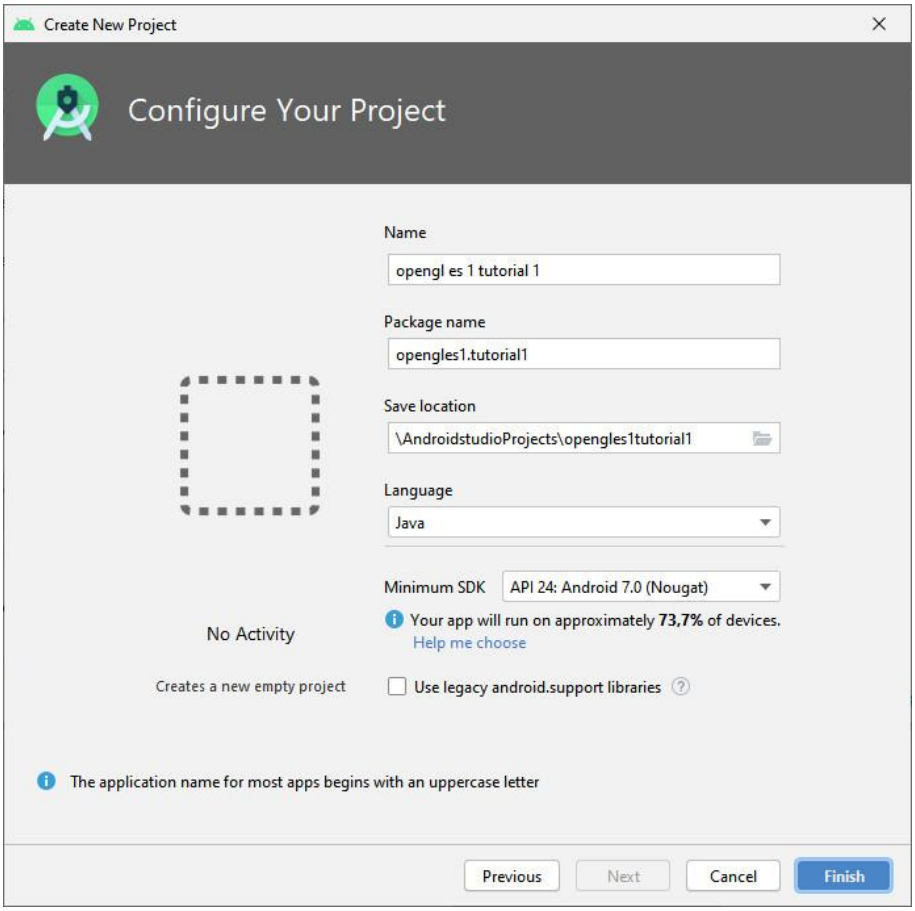
Open Android Studio, click on "Start a new Android Studio project":



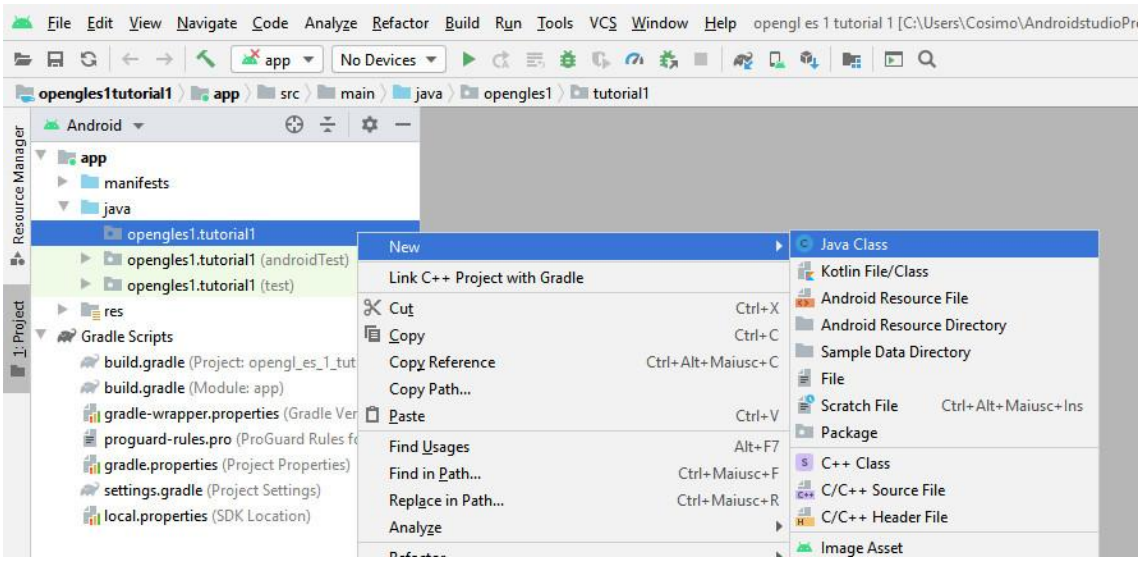
Select the project type "No Activity", click on "Next":

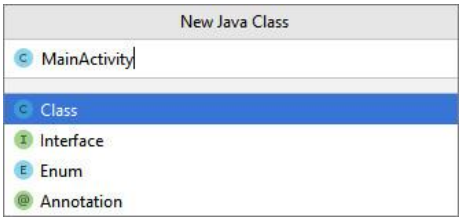


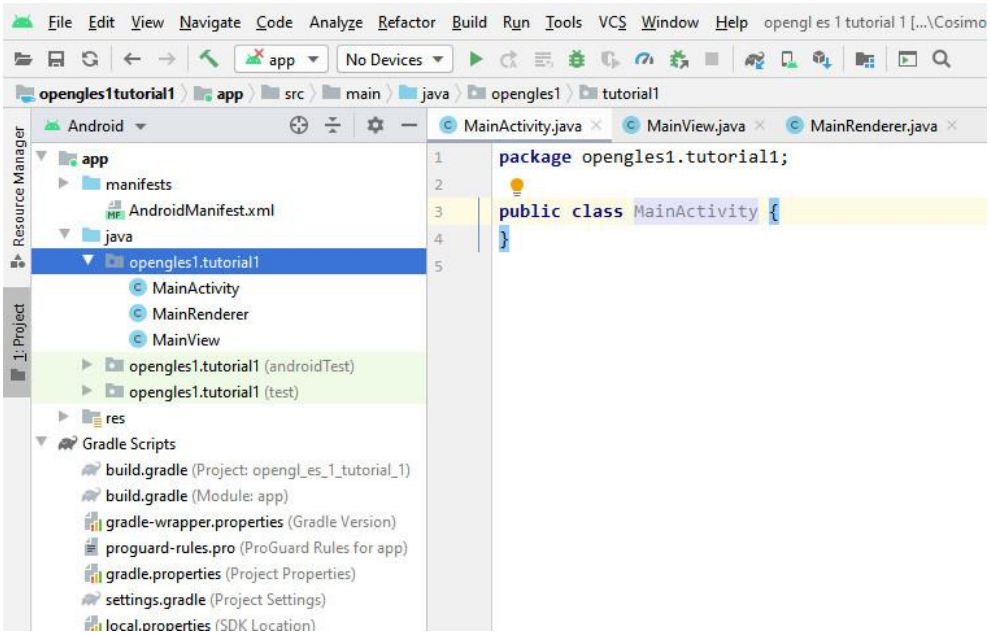
Insert the name of the project and the package, click on "Finish":



Create 3 classes, "MainActivity", "MainView" e "MainRenderer":







Replace the code of the newly created classes with the following:

- for "MainActivity":

```
package opengles1.tutorial1;

import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
import android.view.Window;
import android.view.WindowManager;

public class MainActivity extends Activity {
    // Dedicated surface for displaying OpenGL rendering.
    GLSurfaceView glSurfaceView;

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

        // Hide the activity title.
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        // Set full screen.
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);

        // Create and set the dedicated surface.
        this.glSurfaceView = new MainView(this);
        setContentView(this.glSurfaceView);
    }

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

        // Pause the rendering thread on activity pause.
        this.glSurfaceView.onPause();
    }

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

```
        // Resume the rendering thread on activity resume.
        this.glSurfaceView.onResume();
    }

}
```

- for "MainView":

```
package opengles1.tutorial1;

import android.content.Context;
import android.opengl.GLSurfaceView;

public class MainView extends GLSurfaceView {
    // Interface for drawing graphics.
    MainRenderer glRenderer;

    // Constructor.
    public MainView(Context context) {
        super(context);

        // Create and set the interface for drawing graphics.
        this.glRenderer = new MainRenderer();
        setRenderer(this.glRenderer);
    }

}
```

- for "MainRenderer":

```
package opengles1.tutorial1;

import android.opengl.GLSurfaceView;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

public class MainRenderer implements GLSurfaceView.Renderer {
    int width, height;

    // Called when the surface is created or recreated.
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // Set color's clear-value to black.
        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    }

    // Called when the surface changed size.
    // Called after the surface is created and whenever the OpenGL ES surface size
changes.
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        // Save size for future use.
        this.width = width;
        this.height = height;

        // Set the viewport (display area) to cover the entire window.
        gl.glViewport(0, 0, this.width, this.height);
        // Select projection matrix.
        gl.glMatrixMode(GL10.GL_PROJECTION);
        // Resets the matrix to its initial values to erase any previous settings.
        gl.glLoadIdentity();
        // Describes a matrix that produces a parallel projection.
        gl.glOrthof(0.0f, this.width - 1.0f, 0.0f, this.height - 1.0f, 1.0f, -1.0f);
    }

    // Called to draw the current frame.
```

```
    @Override
    public void onDrawFrame(GL10 gl) {
        // Clear buffers to preset values.
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT);

        // Your rendering code here.
    }

}
```

Complete everything by editing the file "AndroidManifest.xml" in the following way:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="opengles1.tutorial1">

    <application
        android:allowBackup="false"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity
            android:name="opengles1.tutorial1.MainActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Our project will do very little, it will simply clear the frame, but let's see what happens.

Let's start with the file "AndroidManifest.xml" in which we have specified that the activity to be performed at startup is "MainActivity" (note "extends Activity" in the class) and the orientation of the screen is "portrait", that is vertically. If we wanted to set the screen horizontally, then the mode would be "landscape", while omitting the orientation (android: screenOrientation) we will make sure that the rotation event occurs when performed by the operator with the rotation of the smartphone.

When the app starts, the method "onCreate()" is called. With "requestWindowFeature(Window.FEATURE_NO_TITLE);" we hide the title at the top of the screen and with "getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);" we set it to full screen mode.

We arrive at "GLSurfaceView", that is a dedicated surface for graphics with OpenGL.

In order to use this surface we created the class "MainView" (note "extends GLSurfaceView"), class defined when running the app with "this.glSurfaceView = new MainView(this);" and associated at the activity with "setContentView(this.glSurfaceView);".

Who really does the drawing is the "Renderer", so we have created the class "MainRenderer" (note "implements GLSurfaceView.Renderer").

When we defined "this.glSurfaceView", we called "MainView(Context context)" where we defined "Renderer" with "this.glRenderer = new MainRenderer();" and associated at "this.glSurfaceView" with "setRenderer(this.glRenderer);".

Before continuing, let's briefly review the methods "onPause()" and "onResume()" presents in the "MainActivity". Here we find "this.glSurfaceView.onPause();" and "this.glSurfaceView.onResume();", this is used to pause and restore "this.glSurfaceView" when the app is paused or restored by the user; in the absence "this.glSurfaceView" would continue to be active.

In the "Renderer" we find 3 methods:

- "onSurfaceCreated()", performed when the surface is created or recreated;
- "onSurfaceChanged()", performed after "onSurfaceCreated" and when the surface changes dimensions (for example after the rotation of the smartphone);
- "onDrawFrame()", performed after the previous methods, continuously, for drawing.

In the "onSurfaceCreated()" method:

- we set the color that we will use when we delete the frame, in this case, with "gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);", we set black without transparency. The range of values is from 0.0f to 1.0f, correspondingly to RGBA values from 0 to 255.

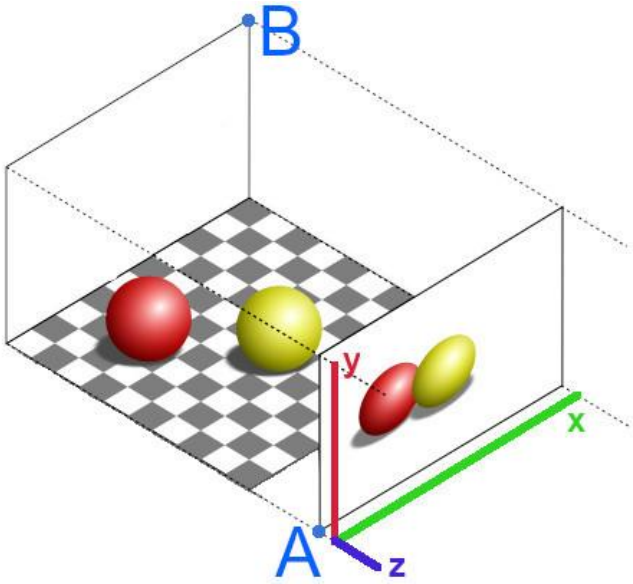
In the "onSurfaceChanged()" method:

- we set the variables "width" and "height" (width and height in pixels of our screen detected on the basis of orientation) for possible future use;
- we set the "Viewport" with "gl.glViewport(0, 0, this.width, this.height);". "Viewport" is the rectangular region of the screen where the frame will be displayed and, in our case, we are specifying the entire screen, from the 0,0 position (lower left corner) for a width (this.width) and a height (this.height);
- we set the type of projection, and in our case, for 2D, we use "glOrthof" instruction. To do this, we must first point to the projection matrix with "gl.glMatrixMode(GL10.GL_PROJECTION);", then we expect, immediately after, the "glOrthof" instruction to set it, but we find the "gl.glLoadIdentity();" instruction. When we set up a matrix with an instruction, we are basically saying: to the matrix put its value multiplied by the new value. This allows you to make changes in cascade if necessary. With "gl.glLoadIdentity();" we are resetting the projection matrix to "1" in such a way that, with the subsequent instruction, there is essentially "1" multiplied by "glOrthof", avoiding unexpected values.

In the "onDrawFrame()" method:

- we draw the frame. The "glClear" instruction clears the frame using only the color set with "glClearColor" as we are telling him to use only the one with the "GL10.GL_COLOR_BUFFER_BIT" parameter.

Let's see better the orthogonal projection that we are using.



In this system, as you notice in the front panel, the depth of one object compared to another does not affect the aspect of the dimensions, we see them as if they were all at the same distance, the difference is made only by the order in they are placed.

With "glOrthof" we defined the volume (A-B) of our projection, specifying the limits:

- for the X axis: left 0.0f, right this.width - 1.0f
- for the Y axis: low 0.0f, high this.height - 1.0f
- for the Z axis: near 1.0f, far -1.0f

The Z coordinate is irrelevant in 2D, therefore set with a unit value, all drawing operations will be placed at Z = 0.0f, in the center and, as mentioned above, the stacking order will affect the final result of the frame.

Let's review the issue "Screen"-"ViewPort"-"glOrthof" for clarity.

1. Our screen has a physical width and height (which we measure in pixels), from the bottom left corner (origin coordinate x = 0 y = 0) to the top right corner (coordinate x = width-1 y = height-1).

2. With "glOrthof" we set the drawing area (in memory) with the following limits:

- left: 0.0f
- right: this.width - 1.0f
- bottom: 0.0f
- top: this.height - 1.0f

3. With "ViewPort" we define the position and the area on the screen that will contain the frame present in memory.

Then, the frame drawn in memory is displayed on the screen in the position and in the area of the viewport and adapted if necessary, adapted if the reference system in memory is not in proportion with the viewport. In our case, we have created a coordinate system in memory that coincides with the physical size of the screen and also of the viewport, we have a correspondence of 1:1.