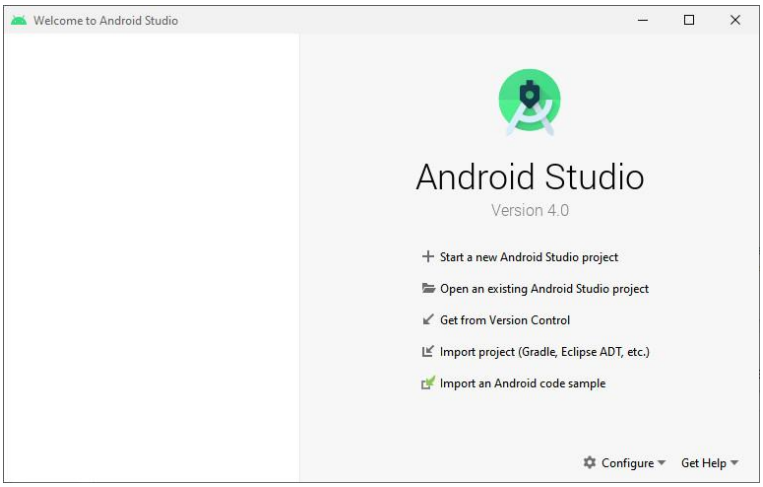
Android - OpenGL ES 2 - Tutorial 1

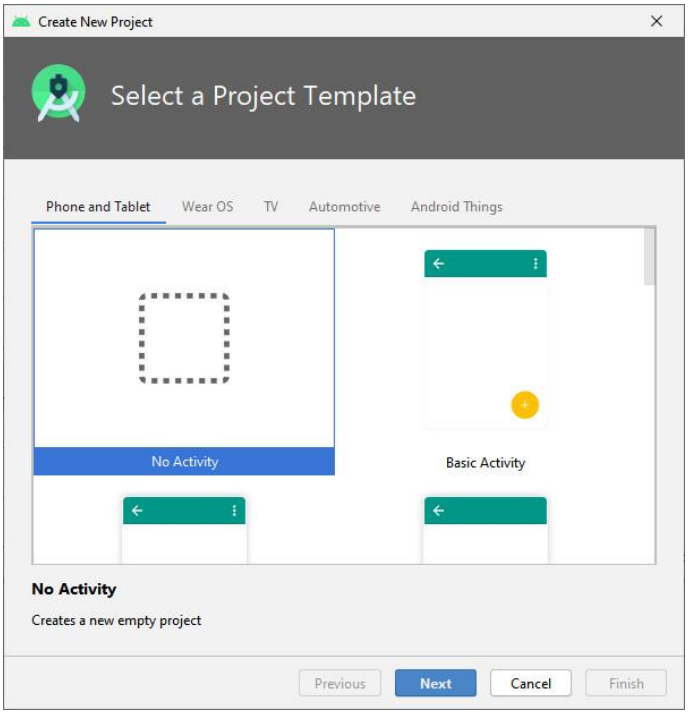
Progetto Base 2D

OpenGL ES è un sottoinsieme delle librerie grafiche OpenGL pensato per dispositivi integrati (telefoni cellulari, PDA ecc. ma anche strumentazione scientifica e industriale). Viene gestito dal consorzio no-profit Khronos Group, che cura anche lo sviluppo della libreria madre OpenGL.

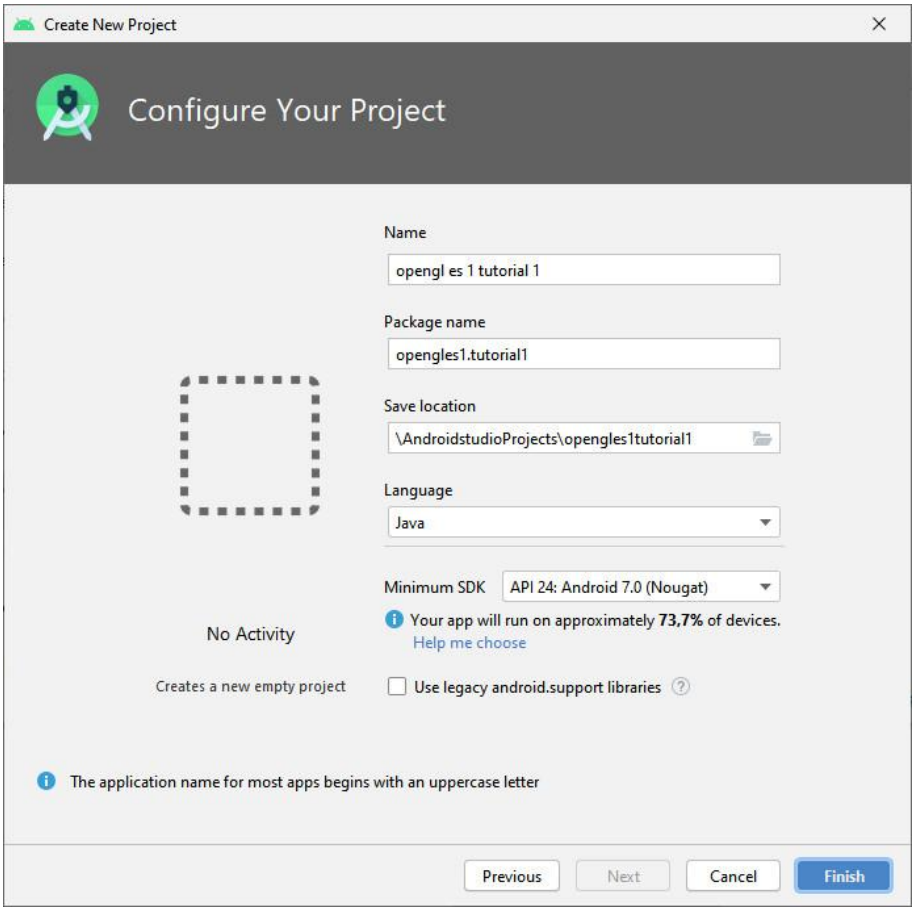
Apriamo Android Studio, clicchiamo su "Start a new Android Studio project":



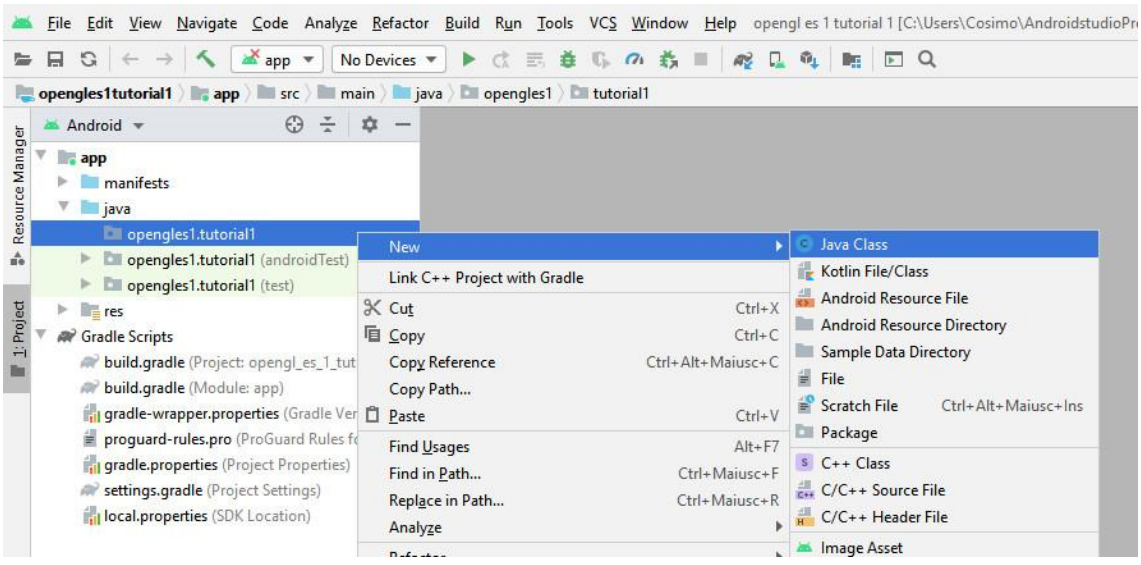
Selezioniamo il tipo progetto "No Activity", clicchiamo su "Next":

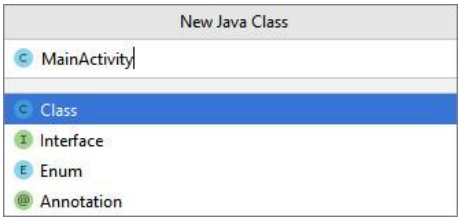


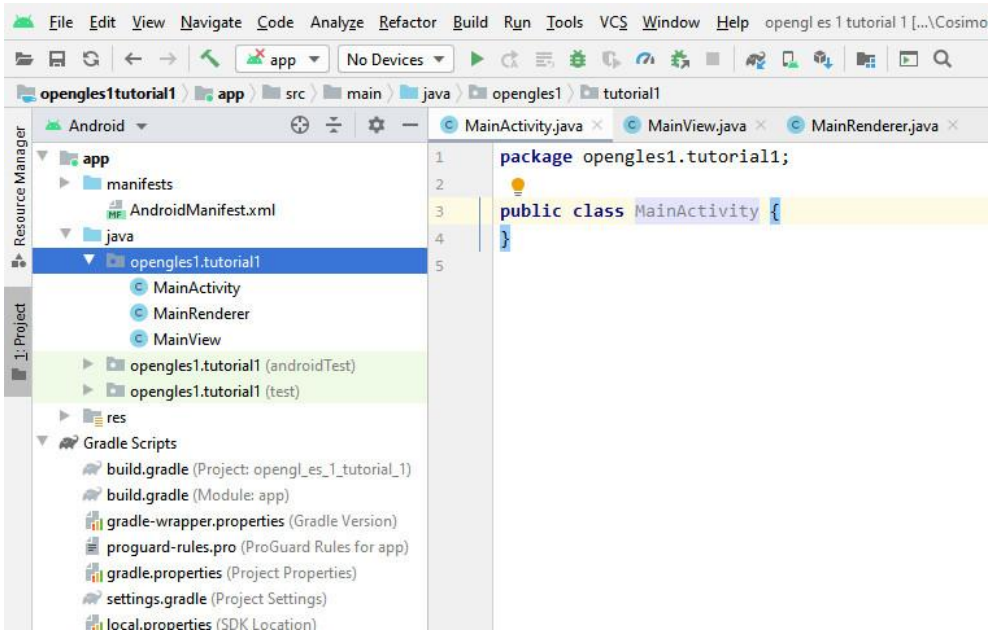
Inseriamo il nome del progetto e del package, clicchiamo su "Finish":



Ora creiamo 3 classi, "MainActivity", "MainView" e "MainRenderer":







Sostituiamo il codice delle classi appena create con i seguenti:

- per "MainActivity":

```
package opengles2.tutorial1;

import android.app.Activity;
import android.app.ActivityManager;
import android.content.Context;
import android.content.pm.ConfigurationInfo;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Toast;

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);

        if (this.SupportsGLES20()) {
            // Create and set the dedicated surface.
            this.glSurfaceView = new MainView(this);
            setContentView(this.glSurfaceView);
        }
        else {
            Toast.makeText(this, "This device does not support OpenGL ES 2.0.",
Toast.LENGTH_LONG).show();
        }
    }

    @Override
    protected void onPause() {
```

```
        super.onPause();

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

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

        // Resume the rendering thread on activity resume.
        if (this.glSurfaceView != null)
            this.glSurfaceView.onResume();
    }

    private boolean SupportsGLES20() {
        // Check if OpenGL ES 2.0 is supported.
        ActivityManager am = (ActivityManager)
getSystemService(Context.ACTIVITY_SERVICE);
        ConfigurationInfo info = am.getDeviceConfigurationInfo();
        return info.reqGlEsVersion >= 0x20000;
    }

}
```

- per "MainView":

```
package opengles2.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);

        // Set OpenGL version.
        setEGLContextClientVersion(2);
        // Create and set the interface for drawing graphics.
        this.glRenderer = new MainRenderer();
        setRenderer(this.glRenderer);
    }

}
```

- per "MainRenderer":

```
package opengles2.tutorial1;

import android.opengl.GLES20;
import android.opengl.GLSurfaceView;
import android.opengl.Matrix;

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

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

    // Projection matrix.
    private final float[] projMatrix = new float[16];
    // Camera matrix.
```

```
    private final float[] viewMatrix = new float[16];

    // Called when the surface is created or recreated.
    @Override
    public void onSurfaceCreated(GL10 unused, EGLConfig config) {
        // Set color's clear-value to black.
        GLES20.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 unused, 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.
        GLES20.glViewport(0, 0, this.width, this.height);
        // Set projection matrix to orthographic system.
        Matrix.orthoM(this.projMatrix, 0, 0.0f, this.width - 1.0f, 0.0f, this.height -
1.0f, 1.0f, -1.0f);
        // Set the camera position (View matrix).
        Matrix.setLookAtM(this.viewMatrix, 0, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
1.0f, 0.0f);
    }

    // Called to draw the current frame.
    @Override
    public void onDrawFrame(GL10 unused) {
        // Clear buffers to preset values.
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);

        // Your rendering code here.
    }

}
```

Completiamo il tutto modificando il file "AndroidManifest.xml" nel seguente modo:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="opengles2.tutorial1">

    <uses-feature android:glEsVersion="0x00020000" android:required="true" />

    <application
        android:allowBackup="true"
        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="opengles2.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>
```

Il nostro progetto farà ben poco, semplicemente cancellerà il frame, ma vediamo cosa accade.

Partiamo dal file "AndroidManifest.xml" nel quale abbiamo specificato che l'activity da eseguire all'avvio è "MainActivity" (notare "extends Activity" nella classe) e che l'orientamento dello schermo è "portrait" ovvero in verticale. Se volessimo impostare lo schermo in orizzontale, allora la modalità sarebbe "landscape", mentre omettendo l'orientamento (android:screenOrientation) faremo in modo che si generi l'evento della rotazione quando eseguito dall'operatore con la rotazione dello smartphone.

All'avvio dell'app verrà richiamato il metodo "onCreate()" dove con "requestWindowFeature(Window.FEATURE_NO_TITLE);" nascondiamo il titolo nella parte superiore dello schermo e con "getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);" impostiamo la modalità a pieno schermo.

Arriviamo a "GLSurfaceView", ovvero una superficie dedicata per la grafica con OpenGL.

Per poter utilizzare questa superficie abbiamo creato la classe "MainView" (notare "extends GLSurfaceView"), classe che definiamo durante l'esecuzione dell'app con "this.glSurfaceView = new MainView(this);" e che associamo all'activity con "setContentView(this.glSurfaceView);".

Chi si occupa realmente di disegnare è il "Renderer". Per questo abbiamo creato la classe "MainRenderer" (notare "implements GLSurfaceView.Renderer").

Quando abbiamo definito "this.glSurfaceView", abbiamo richiamato "MainView(Context context)" dove, a sua volta, è stato definito il "Renderer" con "this.glRenderer = new MainRenderer();" e che abbiamo associato a "this.glSurfaceView" con "setRenderer(this.glRenderer);".

Prima di proseguire, vediamo brevemente i metodi "onPause()" e "onResume()" presenti nel "MainActivity". Qui troviamo "this.glSurfaceView.onPause();" e "this.glSurfaceView.onResume();", ciò serve a mettere in pausa ed a ripristinare "this.glSurfaceView" quando l'app viene messa in pausa o ripristinata dall'operatore; in assenza "this.glSurfaceView" continuerebbe ad essere attivo.

Nel "Renderer" troviamo 3 metodi:

- "onSurfaceCreated()", eseguito quando la superficie viene creata o ricreata;
- "onSurfaceChanged()", eseguito dopo "onSurfaceCreated" e quando la superficie cambia dimensioni (ad esempio dopo la rotazione dello smartphone);
- "onDrawFrame()", eseguito dopo i metodi precedenti, di continuo, per il disegno.

Nel metodo "onSurfaceCreated()":

- impostiamo il colore che utilizzeremo quando cancelleremo il frame, in questo caso, con "GLES20.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);", impostiamo il nero senza trasparenza. L'intervallo dei valori va da 0.0f ad 1.0f, in corrispondenza dei valori RGBA da 0 a 255.

Nel metodo "onSurfaceChanged()":

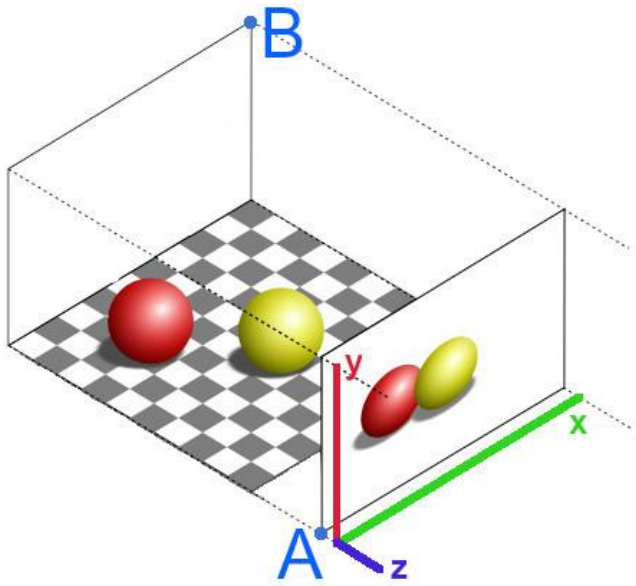
- impostiamo le variabili "width" ed "height" (larghezza ed altezza in pixel del nostro schermo rilevati in base all'orientamento) per un eventuale utilizzo futuro;

- impostiamo il "Viewport" con "GLES20.glViewport(0, 0, this.width, this.height);". Il "Viewport" è la regione rettangolare dello schermo nel quale verrà visualizzato il frame e, nel nostro caso, stiamo specificando l'intero schermo, dalla posizione 0,0 (angolo in basso a sinistra) per una larghezza this.width ed un'altezza this.height;
- impostiamo la matrice di proiezione, e nel nostro caso, per il 2D, con "Matrix.orthoM(this.projMatrix, 0, 0.0f, this.width - 1.0f, 0.0f, this.height - 1.0f, 1.0f, -1.0f);";
- impostiamo la matrice della camera con "Matrix.setLookAtM(this.viewMatrix, 0, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);".

Nel metodo "onDrawFrame()":

- disegniamo il frame. L'istruzione "glClear" cancella il frame utilizzando solo il colore impostato con "glClearColor" in quanto gli stiamo dicendo di usare solo quello con il parametro "GLES20.GL_COLOR_BUFFER_BIT".

Vediamo meglio la proiezione ortogonale che stiamo utilizzando.



In tale sistema, come notate nel pannello frontale, la profondità di un oggetto rispetto ad un altro non influisce sull'aspetto delle dimensioni, li vediamo come se fossero tutti alla stessa distanza, la differenza la fa solo l'ordine in cui sono posti.

Con l'istruzione "Matrix.orthoM" abbiamo definito il volume (A-B) della nostra proiezione, specificandone i limiti:

- per l'asse X: sinistra 0.0f, destra this.width - 1.0f
- per l'asse Y: basso 0.0f, alto this.height - 1.0f
- per l'asse Z: vicino 1.0f, lontano -1.0f

La coordinata Z è ininfluente nel 2D, quindi impostata con valore unitario, tutte le operazioni di disegno verranno poste a Z=0.0f, al centro e, come detto pocanzi, inciderà l'ordine di sovrapposizione al risultato finale del frame.

Rivediamo la questione "Schermo"-"ViewPort"-"orthoM" per maggior chiarezza.

1. Il nostro schermo ha una larghezza ed un'altezza fisica (che misuriamo in pixel), dall'angolo in basso a sinistra (coordinata origine x=0 y=0) all'angolo in alto a destra (coordinata x=larghezza-1 y=altezza-1).

2. Con "Matrix.orthoM" impostiamo l'area di disegno (in memoria) con i seguenti limiti:

- a sinistra: 0.0f
- a destra: this.width - 1.0f
- in basso: 0.0f
- in alto: this.height - 1.0f

3. Con il "ViewPort" definiamo la posizione e l'area sullo schermo che conterrà il frame presente in memoria.

Quindi, il frame disegnato in memoria viene visualizzato sullo schermo nella posizione e nell'area del viewport ed adattato se necessario. Nel nostro caso, abbiamo creato in memoria un sistema di coordinate che coincide con la dimensione fisica dello schermo ed anche del viewport, abbiamo una corrispondenza di 1:1.