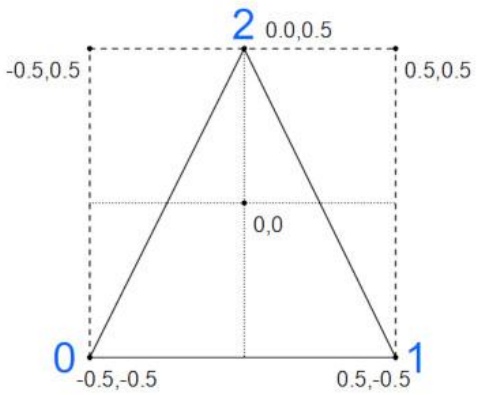
Android - OpenGL ES 1 - Tutorial 2

Disegnare un Triangolo

Consideriamo il triangolo dell'immagine seguente:



Continuando sul progetto base 2D, creiamo una nuova classe "ObjTriangle" e sostituiamo il codice con il seguente:

```
package opengles1.tutorial2;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import java.nio.ShortBuffer;

import javax.microedition.khronos.opengles.GL10;

public class ObjTriangle {
    private FloatBuffer vertexBuffer;
    private ShortBuffer indexBuffer;

    private float[] vertices = {
            -0.5f, -0.5f,
            0.5f, -0.5f,
            0.0f, 0.5f
    };

    private short[] indices = {
            0, 1, 2
    };

    public ObjTriangle() {
        // Set vertex buffer from data.
        ByteBuffer vbb = ByteBuffer.allocateDirect(this.vertices.length * 4);
        vbb.order(ByteOrder.nativeOrder());
        this.vertexBuffer = vbb.asFloatBuffer();
        this.vertexBuffer.put(this.vertices);
        this.vertexBuffer.position(0);

        // Set index buffer from data.
        ByteBuffer ibb = ByteBuffer.allocateDirect(this.indices.length * 2);
        ibb.order(ByteOrder.nativeOrder());
        this.indexBuffer = ibb.asShortBuffer();
        this.indexBuffer.put(this.indices);
        this.indexBuffer.position(0);
```

```
    }

    public void DrawTriangle(GL10 gl) {
        // Enable client state.
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        // Point to buffer.
        gl.glVertexPointer(2, GL10.GL_FLOAT, 0, this.vertexBuffer);
        // Set color for all vertex (red).
        gl.glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
        // Draw.
        gl.glDrawElements(GL10.GL_TRIANGLES, this.indices.length, GL10.GL_UNSIGNED_SHORT,
this.indexBuffer);
        // Disable the client state.
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
    }

}
```

Modifichiamo la classe "MainRenderer" nel seguente modo:

```
package opengles1.tutorial2;

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;

    ObjTriangle objtriangle;

    // Constructor.
    public MainRenderer() {
        // Set class.
        this.objtriangle = new ObjTriangle();
    }

    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    }

    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        this.width = width;
        this.height = height;

        gl.glViewport(0, 0, this.width, this.height);
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        gl.glOrthof(0.0f, this.width - 1.0f, 0.0f, this.height - 1.0f, 1.0f, -1.0f);

        // Select model-view matrix.
        gl.glMatrixMode(GL10.GL_MODELVIEW);
    }

    @Override
    public void onDrawFrame(GL10 gl) {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT);

        gl.glLoadIdentity();
        // Translate triangle.
        gl.glTranslatef(this.width / 2.0f, this.height / 2.0f, 0.0f);
        // Scale triangle.
        gl.glScalef(100.0f, 100.0f, 0.0f);
        // Draw triangle.
        this.objtriangle.DrawTriangle(gl);
```

```
    }
}
```

Notiamo la presenza del costruttore "MainRenderer()" in quanto abbiamo la necessità di definire la classe "ObjTriangle" nel momento in cui il renderer viene definito in "MainView".

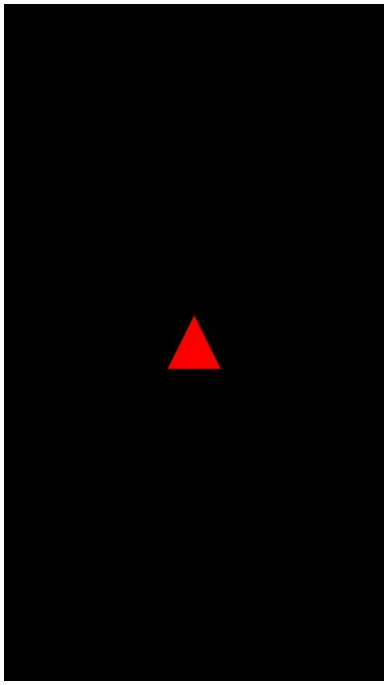
Con "this.objtriangle = new ObjTriangle();" richiamiamo il costruttore della classe "ObjTriangle" dove settiamo i dati del nostro triangolo. I vertici del triangolo presenti in "vertices" e "indices" (scritti in senso antiorario) non sono in un formato utile ad OpenGL, nel costruttore vengono converititi e memorizzati in "vertexBuffer" e "indexBuffer".

Nel metodo "onSurfaceChanged()", dopo aver impostato la matrice di proiezione, utilizziamo l'istruzione "gl.glMatrixMode(GL10.GL_MODELVIEW);" per dire ad OpenGL che d'ora innanzi lavoreremo sulla matrice dei modelli (modelview), tutte le successive istruzioni influiranno su tale matrice.

Nel metodo "onDrawFrame()":

1. cancello il frame;
2. resetto la matrice corrente (modelview);
3. effettuo una traslazione sulla matrice corrente;
4. effettuo uno scalo sulla matrice corrente;
5. disegno il triangolo richiamando la funzione presente nella classe "ObjTriangle":
    a. abilito l'uso dei vertici;
    b. specifico dove sono i vertici;
    c. specifico il colore di tutti i vertici (rosso);
    d. disegno;
    e. disabilito l'uso dei vertici.

Il risultato è:



Durante la fase di disegno in "DrawTriangle()", i vertici e la posizione del triangolo subiranno l'influenza della matrice modelview che precedentemente è stata traslata e scalata.

Sapendo che il nostro sistema di coordinate ortogonali è di tipo "1:1" con lo schermo ed il triangolo di dimensioni "1x1", traslare di 200 equivale ad uno spostamento di 200 e scalare di 10 equivale ad un ridimensionamento di 10. Ma fate attenzione all'ordine! Se prima scaliamo di 10, quando trasleremo di 200 ci sposteremo di 2000, cioè la nuova "unità di base" (10) per 200 volte.