Android - OpenGL ES 2 - Tutorial 6

Apply Texture

Now let's see how to draw a square on which to apply a texture:

Create the class "ObjTexture" with the following code:

```
package opengles2.tutorial6;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLES20;
import android.opengl.GLUtils;

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

import javax.microedition.khronos.opengles.GL10;

public class ObjTexture {
    private FloatBuffer vertexBuffer;
    private ShortBuffer indexBuffer;
    private FloatBuffer textureBuffer;

    private final int COORDS_PER_VERTEX = 2;
    private final int vertexStride = COORDS_PER_VERTEX * 4;
    private float[] vertices = {
            -0.5f, -0.5f,
            0.5f, -0.5f,
            -0.5f, 0.5f,
            0.5f, 0.5f
    };
```

```
    private short[] indices = {
            0, 1, 2,
            2, 1, 3
    };

    private final int COORDS_PER_TEXTURE = 2;
    private final int textureStride = COORDS_PER_TEXTURE * 4;
    private float[] texture = {
            0.0f, 1.0f,
            1.0f, 1.0f,
            0.0f, 0.0f,
            1.0f, 0.0f
    };

    private final String vertexShaderCode =
            "uniform mat4 uMVPMatrix;" +
            "attribute vec4 vPosition;" +
            "attribute vec2 vTextCoordIn;" +
            "varying vec2 vTextCoordOut;" +
            "void main() {" +
            "  gl_Position = uMVPMatrix * vPosition;" +
            "  vTextCoordOut = vTextCoordIn;" +
            "}";

    private final String fragmentShaderCode =
            "precision mediump float;" +
            "uniform sampler2D sTexture;" +
            "varying lowp vec2 vTextCoordOut;" +
            "void main() {" +
            "  gl_FragColor = texture2D(sTexture, vTextCoordOut);" +
            "}";

    private int mProgram;

    public ObjTexture() {
        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);

        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);

        ByteBuffer itt = ByteBuffer.allocateDirect(this.texture.length * 4);
        itt.order(ByteOrder.nativeOrder());
        this.textureBuffer = itt.asFloatBuffer();
        this.textureBuffer.put(this.texture);
        this.textureBuffer.position(0);
    }
    public void SetProgramShader() {
        int vertexShader = MainRenderer.LoadShader(GLES20.GL_VERTEX_SHADER,
this.vertexShaderCode);
        int fragmentShader = MainRenderer.LoadShader(GLES20.GL_FRAGMENT_SHADER,
this.fragmentShaderCode);

        this.mProgram = GLES20.glCreateProgram();
        GLES20.glAttachShader(this.mProgram, vertexShader);
        GLES20.glAttachShader(this.mProgram, fragmentShader);
        GLES20.glLinkProgram(this.mProgram);
    }

    private final int text_count = 2; // Texture count.
    private int textures[] = new int[this.text_count]; // Texture pointer.
```

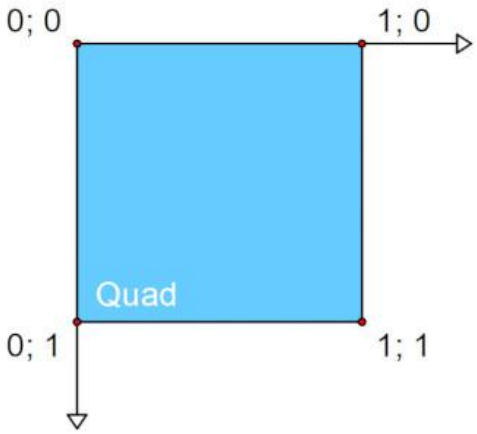
```
    public void LoadTexture(Context context) {
        // Generate texture names.
        GLES20.glGenTextures(1, textures, 0);

        Bitmap bitmap;

        // Set bitmap from resources.
        bitmap = BitmapFactory.decodeResource(context.getResources(),
R.drawable.boxwood);
        // Bind a named texture to a texturing target.
        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, textures[0]);
        // Set texture parameters.
        GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MIN_FILTER,
GLES20.GL_LINEAR);
        // Set texture parameters.
        GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MAG_FILTER,
GLES20.GL_LINEAR);
        // Load texture.
        GLUtils.texImage2D(GLES20.GL_TEXTURE_2D, 0, bitmap, 0);

        bitmap = BitmapFactory.decodeResource(context.getResources(),
R.drawable.textwood);
        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, textures[1]);
        GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MIN_FILTER,
GLES20.GL_LINEAR);
        GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MAG_FILTER,
GLES20.GL_LINEAR);
        GLUtils.texImage2D(GLES20.GL_TEXTURE_2D, 0, bitmap, 0);

        // Clean up.
        bitmap.recycle();
    }

    private int positionHandle;
    private int modelHandle;
    private int TextureCoordHandle;
    private int TextureUniformHandle;

    public void DrawTexture(float[] modelMatrix, int texture) {
        GLES20.glUseProgram(this.mProgram);

        this.positionHandle = GLES20.glGetAttribLocation(this.mProgram, "vPosition");
        GLES20.glEnableVertexAttribArray(this.positionHandle);
        GLES20.glVertexAttribPointer(this.positionHandle, COORDS_PER_VERTEX,
GLES20.GL_FLOAT, false, this.vertexStride, this.vertexBuffer);

        this.modelHandle = GLES20.glGetUniformLocation(this.mProgram, "uMVPMatrix");
        GLES20.glUniformMatrix4fv(this.modelHandle, 1, false, modelMatrix, 0);

        // Get handle to texture coordinates shader's vTextCoordIn member.
        this.TextureCoordHandle = GLES20.glGetAttribLocation(mProgram, "vTextCoordIn");
        // Enable a handle to the texture coordinates.
        GLES20.glEnableVertexAttribArray(this.TextureCoordHandle);
        // Prepare the texture coordinates data.
        GLES20.glVertexAttribPointer(this.TextureCoordHandle, COORDS_PER_TEXTURE,
GLES20.GL_FLOAT, false, textureStride, this.textureBuffer);
        // Get handle to texture uniform shader's sTexture member.
        this.TextureUniformHandle = GLES20.glGetAttribLocation(mProgram, "sTexture");
        // Tell the texture uniform sampler to use this texture in the shader by binding
to texture unit 0.
        GLES20.glUniform1i(this.TextureUniformHandle, 0);

        // Set the active texture unit to texture unit 0.
        GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
        // Bind the texture to this unit.
        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, this.textures[texture]);

        GLES20.glDrawElements(GL10.GL_TRIANGLES, this.indices.length,
```

```
GL10.GL_UNSIGNED_SHORT, this.indexBuffer);

        GLES20.glDisableVertexAttribArray(this.positionHandle);
        GLES20.glDisableVertexAttribArray(this.TextureCoordHandle);
    }

}
```

The square is that of tutorial 5, in addition we find the coordinates of the texture to be applied.

For textures (present in the project in "res \ drawable") it is preferable that they have dimensions multiple of 2, in our case they are 512x512 pixels.

The texture coordinate system is:



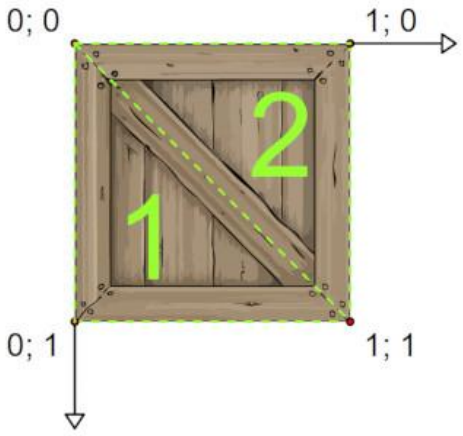
OpenGL will draw the 2 triangles of the square by applying the texture using the coordinates.

For the first triangle, indices 0, 1, 2:

- P1(-0.5;-0.5) P2(0.5;-0.5) P3(-0.5;0.5)
- TextureP1(0;1) TextureP2(1;1) TextureP3(0;0)

For the second triangle, indices 2, 1, 3:

- P1(-0.5;0.5) P2(0.5;-0.5) P3(0.5;0.5)
- TextureP1(0;0) TextureP2(1;1) TextureP3(1;0)



In the "MainRenderer" class we have:

```
package opengles2.tutorial6;

import android.content.Context;
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 {
    MainActivity activity;
    int width, height;

    private final float[] projMatrix = new float[16];
    private final float[] viewMatrix = new float[16];
    private final float[] basicMatrix = new float[16];
    private float[] modelMatrix = new float[16];

    ObjTexture objtexture;

    public MainRenderer(Context context) {
        // Set context.
        this.activity = (MainActivity) context;

        this.objtexture = new ObjTexture();
    }

    @Override
    public void onSurfaceCreated(GL10 unused, EGLConfig config) {
        GLES20.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

        // Enable texture.
        GLES20.glEnable(GLES20.GL_TEXTURE_2D);
        // Load texture.
        this.objtexture.LoadTexture(this.activity);

        this.objtexture.SetProgramShader();
    }

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

        GLES20.glViewport(0, 0, this.width, this.height);
        Matrix.orthoM(this.projMatrix, 0, 0.0f, this.width - 1.0f, 0.0f, this.height -
1.0f, 1.0f, -1.0f);
        Matrix.setLookAtM(this.viewMatrix, 0, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
1.0f, 0.0f);
        Matrix.multiplyMM(this.basicMatrix, 0, this.projMatrix, 0, this.viewMatrix, 0);
    }

    @Override
    public void onDrawFrame(GL10 unused) {
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);
```

```
        this.ResetModelMatrix();
        Matrix.translateM(this.modelMatrix, 0, this.width / 2.0f, this.height / 4.0f *
3.0f, 0.0f);
        Matrix.scaleM(this.modelMatrix, 0, 300.0f, 300.0f, 0.0f);
        this.objtexture.DrawTexture(this.modelMatrix, 0);

        this.ResetModelMatrix();
        Matrix.translateM(this.modelMatrix, 0, this.width / 2.0f, this.height / 4.0f,
0.0f);
        Matrix.scaleM(this.modelMatrix, 0, 300.0f, 300.0f, 0.0f);
        this.objtexture.DrawTexture(this.modelMatrix, 1);
    }

    public static int LoadShader(int type, String shaderCode) {
        int shader = GLES20.glCreateShader(type);
        GLES20.glShaderSource(shader, shaderCode);
        GLES20.glCompileShader(shader);

        return shader;
    }

    private void ResetModelMatrix() {
        this.modelMatrix = this.basicMatrix.clone();
    }

}
```

In the "onSurfaceCreated()" method is enabled the use of textures and the textures are loaded with the "LoadTexture()" method.

With "DrawTexture()" we draw the square specifying which texture to use through an index that coincides with the loading order.