《基于 Andropid 源生 AR 应用开发》

实验手册 02

版 本 1.0

文档提供：智能设备教研室 丁盟

目录

第2章　2D 图形绘制

2.1 实验目的

目的一： 掌握着色器的使用流程。

目的二： 掌握数组存储顶点数据的方法。

目的三： 掌握简单 2D 图形绘制的方法。

2.2 准备工作

准备一： 在 Android Studio 中创建一个空的项目，并根据实验手册 01 完成 OpenGL ES 应用程序的基本配置工作。

准备二： 着色器的一般使用流程如下：

1) 创建着色器：glCreateShader
2) 指定着色器源代码字符串：glShaderSource
3) 编译着色器：glCompileShader
4) 创建着色器可执行程序：glCreateProgram
5) 向可执行程序中添加着色器：glAttachShader
6) 链接可执行程序：glLinkProgram

2.3 实验步骤

步骤一 在 HelloOpenGLES20Renderer 类中添加如下属性成员：

```
// 顶点数据
private FloatBuffer triangleVB;
// 着色器程序索引值
private int mProgram;
// 指向 vertex shader 的成员 vPosition 的 handle
private int mPositionHandle;
// 指向 fragment shader 的成员 vColor 的 handle
private int mColorHandle;
```

```
// 顶点着色器
private final String vertexShaderCode =
        "attribute vec4 vPosition; \n" +
        "void main(){              \n" +
        "  gl_Position = vPosition; \n" +
        "}                         \n";

// 片元着色器
private final String fragmentShaderCode =
        "precision mediump float;  \n" +
        "uniform vec4 vColor;      \n" +
        "void main(){              \n" +
        "  gl_FragColor = vColor; \n" +
        "}                         \n";

// 红色
float color[] = { 1.0f, 0.0f, 0.0f, 1.0f };
```

步骤二 添加 initShapes()方法，用来生成顶点数据 Buffer。

```
private void initShapes() {

    float triangleCoords[] = {
            -0.5f, -0.25f, 0,
            0.5f, -0.25f, 0,
            0.0f, 0.5f, 0
    };

    // 初始化三角形缓存区
    ByteBuffer vbb = ByteBuffer.allocateDirect(
            // 坐标数 * 4（float 占用4 字节）
            triangleCoords.length * 4);

    // 使用本机字节序（即大端小端）
    vbb.order(ByteOrder.nativeOrder());
    // 根据 ByteBuffer 创建一个浮点缓冲区
    triangleVB = vbb.asFloatBuffer();

    // 添加坐标
    triangleVB.put(triangleCoords);
```

```
    // 设置Buff 游标位置
    triangleVB.position(0);

}
```

步骤三 添加 loadShader()方法，用来加载着色器代码。

```
private int loadShader(int type, String shaderCode){

    // 创建一个vertex shader 类型
(GLES20.GL_VERTEX_SHADER)
    // 或fragment shader 类型(GLES20.GL_FRAGMENT_SHADER)
    int shader = GLES20.glCreateShader(type);

    // 将源码添加到shader 并编译
    GLES20.glShaderSource(shader, shaderCode);
    GLES20.glCompileShader(shader);

    return shader;
}
```

步骤四 在 onSurfaceCreated()接口中完成顶点数据 Buffer 的初始化及着色器可执行程序的创建。

```
@Override
public void onSurfaceCreated(GL10 gl10, EGLConfig
eglConfig) {
    GLES20.glClearColor(0.0f, 1.0f, 0.0f, 1.0f);

    initShapes();

    int vertexShader =
loadShader(GLES20.GL_VERTEX_SHADER, vertexShaderCode);
    int fragmentShader =
loadShader(GLES20.GL_FRAGMENT_SHADER,
fragmentShaderCode);

    // 创建一个空的OpenGL ES Program
    mProgram = GLES20.glCreateProgram();
    // 将vertex shader 添加到program
    GLES20.glAttachShader(mProgram, vertexShader);
```

```
    // 将fragment shader添加到program
    GLES20.glAttachShader(mProgram, fragmentShader);
    // 创建可执行的 OpenGL ES program
    GLES20.glLinkProgram(mProgram);

    // 获取指向vertex shader的成员vPosition的 handle
    mPositionHandle =
GLES20.glGetAttribLocation(mProgram, "vPosition");

    // 获取指向fragment shader的成员vColor的handle
    mColorHandle =
GLES20.glGetUniformLocation(mProgram, "vColor");
}
```

步骤五 在 onDrawFrame()方法中完成图形的绘制。

```
@Override
public void onDrawFrame(GL10 gl10) {
    GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT |
GLES20.GL_DEPTH_BUFFER_BIT);

    // 将program加入OpenGL ES环境中
    GLES20.glUseProgram(mProgram);

    // 准备三角形的坐标数据
    GLES20.glVertexAttribPointer(mPositionHandle, 3,
GLES20.GL_FLOAT, false, 12, triangleVB);
    // 启用一个指向三角形的顶点数组的handle
    GLES20.glEnableVertexAttribArray(mPositionHandle);

    // 设置三角形的颜色
    GLES20.glUniform4fv(mColorHandle, 1, color, 0);

    // 画三角形
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, 3);

    // 禁用指向三角形的顶点数组

GLES20.glDisableVertexAttribArray(mPositionHandle);
```

```
}
```

2.4 实验结论

当编码工作完成后在模拟器或真机中运行项目，效果如下：

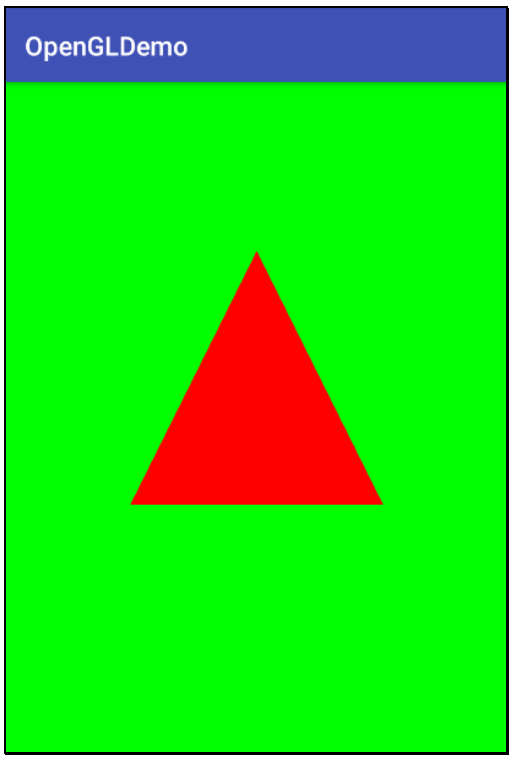

图 2.4.1