《基于 Andropid 源生 AR 应用开发》

实验手册 04

版 本 1.0

文档提供：智能设备教研室 丁盟

目录

第4章　3D 图形绘制

4.1 实验目的

目的一： 掌握 3D 图形的绘制步骤。

目的二： 掌握 3D 图形的旋转方法。

目的三： 掌握 3D 图形的平移方法。

4.2 准备工作

准备一： 在 Android Studio 中根据实验手册 1 创建一个最简单的 OpenGL ES 应用。

准备二： 本次实验中绘制的为 3D 正方体图形，与之前的 2D 图形存在 Z 轴上的使用区别，也就是在 3D 绘图中会用到深度缓存信息(Z 轴数据)，因此在每次绘制前的清除操作时除了清除颜色缓存外，还需另外清除深度缓存。

```
// 清除颜色缓冲与深度缓冲
GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT |
               GLES20.GL_DEPTH_BUFFER_BIT);
```

准备三： 模型矩阵平移

```
void translateM (float[] m, int mOffset,
                 float x, float y, float z)
```

- float[] m　　//模型矩阵
- int mOffset　　//偏移量
- float x　　//X 轴移动的距离
- float y　　//Y 轴移动的距离
- float z　　//Z 轴移动的距离

准备四： 模型矩阵旋转

```
void rotateM (float[] m, int mOffset, float a,
              float x, float y, float z)
```

- float[] m　　//模型矩阵
- int mOffset　　//偏移量
- float a　　//旋转角度
- float x　　//X 轴分量
- float y　　//Y 轴分量
- float z　　//Z 轴分量

准备五： 正方体顶点数据

一个标准正方体由 8 个顶点组成，假设把个顶点位置如下：

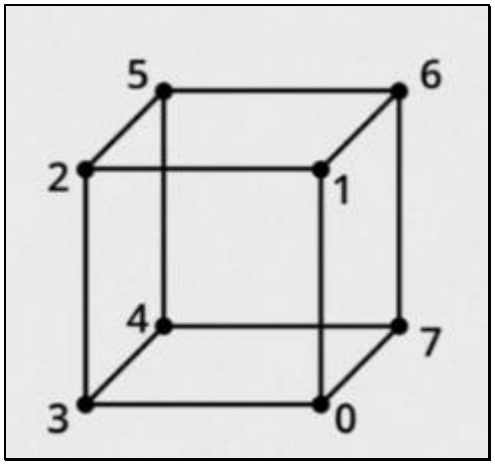

图 4.2.1

每个顶点数据如下，根据上图依次组件每个面的顶点坐标数组。注意在构造顶点数组时，每个三角形的顶点顺序应保证为逆时针（进行判断正反面）。

V0:{ 1.0, -1.0, 1.0}　　V4:{-1.0, -1.0, -1.0}

V1:{ 1.0, 1.0, 1.0}　　V5:{-1.0, 1.0, -1.0}

V2:{-1.0, 1.0, 1.0}　　V6:{ 1.0, 1.0, -1.0}

V3:{-1.0, -1.0, 1.0}　　V7:{ 1.0, -1.0, -1.0}

4.3 实验步骤

步骤一 在自定义 Renderer 类中添加相关属性成员：

```
// 正方体坐标顶点 Buffer
private FloatBuffer mCubePositions;
// 正方体颜色 Buffer
private FloatBuffer mCubeColors;

// 渲染程序句柄
private int mProgram;

// 顶点着色器中顶点变量句柄
private int mPositionHandle;
// 顶点着色器中颜色变量句柄
private int mColorHandle;
// 顶点着色器中坐标转换矩阵句柄
private int mMVPMatrixHandle;

// 总坐标转换矩阵
private float[] mMVPMatrix = new float[16];
// 模型矩阵
private float[] mModelMatrix = new float[16];
// 视图矩阵
private float[] mViewMatrix = new float[16];
// 投影矩阵
private float[] mProjectionMatrix = new float[16];
```

步骤二 接着在 Renderer 类中添加着色器代码成员变量。

```
// 顶点着色器代码
private String mVertexShaderCode =
          "uniform mat4 u_MVPMatrix;      \n"
        + "attribute vec4 a_Position;     \n"
        + "attribute vec4 a_Color;        \n"
        + "varying vec4 v_Color;          \n"
        + "void main()                    \n"
```

```
        + "{                                \n"
        + "   v_Color = a_Color;            \n"
        + "   gl_Position = u_MVPMatrix * a_Position;
\n"
        + "}                                \n";

// 片元着色器
private String mFragmentShaderCode =
          "precision mediump float;       \n"
        + "varying vec4 v_Color;          \n"
        + "void main()                    \n"
        + "{                              \n"
        + "   gl_FragColor = v_Color;     \n"
        + "}
```

步骤三 在 Renderer 类中添加加载着色器函数。

```
// 加载着色器
private int loadShader(int type, String shaderCode){

    // 顶点着色器 GLES20.GL_VERTEX_SHADER
    // 片元着色器 GLES20.GL_FRAGMENT_SHADER
    int shader = GLES20.glCreateShader(type);

    GLES20.glShaderSource(shader, shaderCode);
    GLES20.glCompileShader(shader);

    return shader;
}
```

步骤四 在 Renderer 中添加初始化图形数据函数。

```
// 初始化图形数据
private void initShapes() {
```

```
// 正方体顶点数据数组
float cubePosition[] = {
    // 正面
    -1.0f, 1.0f, 1.0f,
    -1.0f, -1.0f, 1.0f,
    1.0f, 1.0f, 1.0f,
    -1.0f, -1.0f, 1.0f,
    1.0f, -1.0f, 1.0f,
    1.0f, 1.0f, 1.0f,
    // 右面
    1.0f, 1.0f, 1.0f,
    1.0f, -1.0f, 1.0f,
    1.0f, 1.0f, -1.0f,
    1.0f, -1.0f, 1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f, 1.0f, -1.0f,
    // 背面
    1.0f, 1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    -1.0f, 1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f, -1.0f,
    -1.0f, 1.0f, -1.0f,
    // 左面
    -1.0f, 1.0f, -1.0f,
    -1.0f, -1.0f, -1.0f,
    -1.0f, 1.0f, 1.0f,
    -1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f, 1.0f,
    -1.0f, 1.0f, 1.0f,
    // 顶面
    -1.0f, 1.0f, -1.0f,
    -1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, -1.0f,
    -1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f,
    1.0f, 1.0f, -1.0f,
    // 底面
    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, 1.0f,
```

```
        -1.0f, -1.0f, -1.0f,
        1.0f, -1.0f, 1.0f,
        -1.0f, -1.0f, 1.0f,
        -1.0f, -1.0f, -1.0f,
    };

    // 正方体颜色数据数组
    float[] cubeColor = {
        // 正面（红）
        1.0f, 0.0f, 0.0f, 1.0f,
        1.0f, 0.0f, 0.0f, 1.0f,
        1.0f, 0.0f, 0.0f, 1.0f,
        1.0f, 0.0f, 0.0f, 1.0f,
        1.0f, 0.0f, 0.0f, 1.0f,
        1.0f, 0.0f, 0.0f, 1.0f,
        // 右面（绿）
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 0.0f, 1.0f,
        // 背面（蓝）
        0.0f, 0.0f, 1.0f, 1.0f,
        0.0f, 0.0f, 1.0f, 1.0f,
        0.0f, 0.0f, 1.0f, 1.0f,
        0.0f, 0.0f, 1.0f, 1.0f,
        0.0f, 0.0f, 1.0f, 1.0f,
        0.0f, 0.0f, 1.0f, 1.0f,
        // 左面（黄）
        1.0f, 1.0f, 0.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 1.0f,
        1.0f, 1.0f, 0.0f, 1.0f,
        // 顶面（青）
        0.0f, 1.0f, 1.0f, 1.0f,
        0.0f, 1.0f, 1.0f, 1.0f,
        0.0f, 1.0f, 1.0f, 1.0f,
        0.0f, 1.0f, 1.0f, 1.0f,
```

```
        0.0f, 1.0f, 1.0f, 1.0f,
        0.0f, 1.0f, 1.0f, 1.0f,
        // 底面（洋红）
        1.0f, 0.0f, 1.0f, 1.0f,
        1.0f, 0.0f, 1.0f, 1.0f,
        1.0f, 0.0f, 1.0f, 1.0f,
        1.0f, 0.0f, 1.0f, 1.0f,
        1.0f, 0.0f, 1.0f, 1.0f,
        1.0f, 0.0f, 1.0f, 1.0f
    };

    // 获取顶点 Buffer
    mCubePositions =
ByteBuffer.allocateDirect(cubePosition.length*4)
            .order(ByteOrder.nativeOrder())
            .asFloatBuffer();
    mCubePositions.put(cubePosition);
    mCubePositions.position(0);

    // 获取颜色 Buffer
    mCubeColors =
ByteBuffer.allocateDirect(cubeColor.length*4)
            .order(ByteOrder.nativeOrder())
            .asFloatBuffer();
    mCubeColors.put(cubeColor);
    mCubeColors.position(0);
}
```

步骤五 实现 Renderer 中的 onSurfaceCreated 接口，完成渲染器的相关初始化工作。

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

    // 打开背面剪裁
```

```
    GLES20.glEnable(GLES20.GL_CULL_FACE);
    // 打开深度检测
    GLES20.glEnable(GLES20.GL_DEPTH_TEST);

    // 初始化图形顶点、颜色数据 Buffer
    initShapes();

    // 加载顶点着色器
    int vertexShader =
loadShader(GLES20.GL_VERTEX_SHADER,
mVertexShaderCode);
    // 加载片元着色器
    int fragmentShader =
loadShader(GLES20.GL_FRAGMENT_SHADER,
mFragmentShaderCode);

    // 创建渲染程序
    mProgram = GLES20.glCreateProgram();
    // 将着色器添加到渲染程序中
    GLES20.glAttachShader(mProgram, vertexShader);
    GLES20.glAttachShader(mProgram, fragmentShader);
    // 链接渲染程序
    GLES20.glLinkProgram(mProgram);

    // 分别获得着色器相应变量的句柄
    mPositionHandle =
GLES20.glGetAttribLocation(mProgram, "a_Position");
    mColorHandle = GLES20.glGetAttribLocation(mProgram,
"a_Color");
    mMVPMatrixHandle =
GLES20.glGetUniformLocation(mProgram, "u_MVPMatrix");

    // 设置视图矩阵
    Matrix.setLookAtM(mViewMatrix, 0,
            0f, 0f, -0.5f,
            0f, 0f, -5.0f,
            0f, 1.0f, 0.0f);
}
```

步骤六 实现 Renderer 中的 onSurfaceChanged 接口，完成横纵屏切换时需要完成的操作。

```
@Override
public void onSurfaceChanged(GL10 gl10, int width, int
height) {
    GLES20.glViewport(0, 0, width, height);

    // 获取投影矩阵
    float ratio = (float) width / height;
    Matrix.frustumM(mProjectionMatrix, 0,
            -ratio, ratio, -1.0f, 1.0f, 1.0f, 10.0f);
}
```

步骤七 实现 onDrawFrame()方法，完成图形的绘制。

```
@Override
public void onDrawFrame(GL10 gl10) {
    // 清除颜色缓冲与深度缓冲
    GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT |
GLES20.GL_DEPTH_BUFFER_BIT);

    // 使用渲染程序
    GLES20.glUseProgram(mProgram);

    // 传入正方体顶点数据 Buffer
    GLES20.glVertexAttribPointer(mPositionHandle, 3,
GLES20.GL_FLOAT,
            false, 0, mCubePositions);
    GLES20.glEnableVertexAttribArray(mPositionHandle);

    // 传入正方体颜色数据 Buffer
    GLES20.glVertexAttribPointer(mColorHandle, 4,
GLES20.GL_FLOAT,
            false, 0, mCubeColors);
```

```
    GLES20.glEnableVertexAttribArray(mColorHandle);

    // 求取正方体旋转的角度，根据时间确定，10 秒旋转 360 度
    long time = System.currentTimeMillis() % 10000L;
    float angleInDegrees = (360.0f / 10000.0f) * ((int)
time);
    // 设置模型矩阵为单位矩阵，类似于初始化工作
    Matrix.setIdentityM(mModelMatrix, 0);
    // 将模型向远离摄像头的方向移动 5.0f
    Matrix.translateM(mModelMatrix, 0, 0.0f, 0.0f,
-5.0f);
    // 以 XY 轴中线进行旋转
    Matrix.rotateM(mModelMatrix, 0, angleInDegrees,
1.0f, 1.0f, 0.0f);

    // 求取 MV 矩阵（Model * View）
    Matrix.multiplyMM(mMVPMatrix, 0, mViewMatrix, 0,
mModelMatrix, 0);
    // 求取坐标转换总矩阵 MVP 矩阵(Model * View * Project)
    Matrix.multiplyMM(mMVPMatrix, 0, mProjectionMatrix,
0, mMVPMatrix, 0);

    // 将矩阵传递给着色器的矩阵变量
    GLES20.glUniformMatrix4fv(mMVPMatrixHandle, 1,
false, mMVPMatrix, 0);

    // 绘制图形
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, 36);
}
```

4.4 实验结论

当编码工作完成后在模拟器或真机中运行项目，会发现一个魔方正方体在绕着 XY 轴中线进行匀速旋转。效果如下：

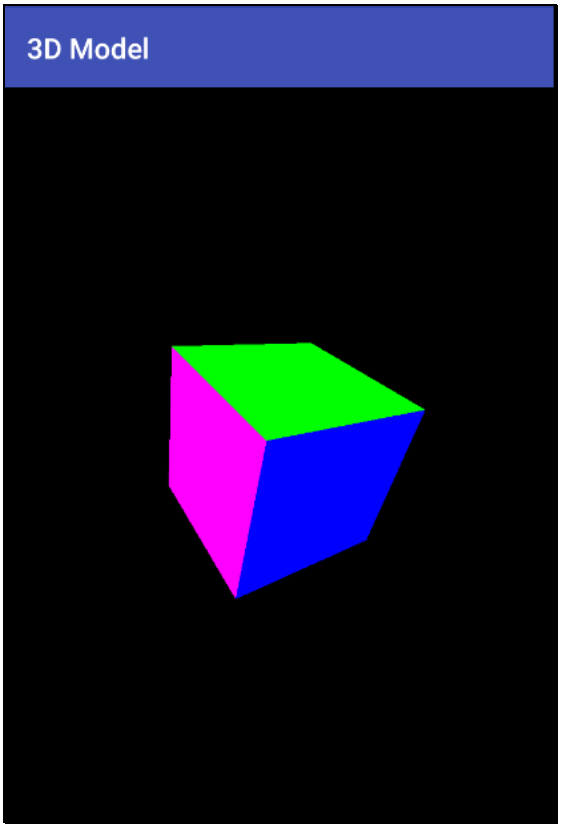

图 4.4.1