第 5 章 利用 OpenGL 绘制三维图形

我们生活的世界就是三维的世界，所谓三维空间是指我们所处的空间，可以理解为有前后—上下—左右三个维度。

三维即是坐标轴的三个轴，即 x 轴、y 轴、z 轴，其中 x 表示左右空间，y 表示上下空间，z 表示前后空间，这样就形成了人的视觉立体感。如图 5-1 所示。

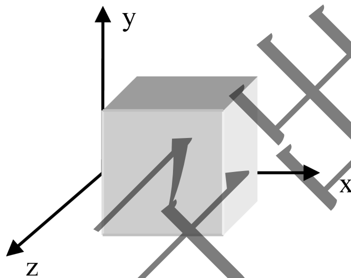

图 5-1 三维坐标系

利用计算机进行三维图形的绘制程序中，如果直接去调用计算机硬件去绘制三维图形，难度很大。所幸的是，很多软件产商提供了图形库，利用这些图形库，我们可以非常容易的绘制出令人满意的三维画面。其中包含 direct3D 和 OpenGL 包，我们接下来的章节将会学习 OpenGL，介绍在手机上如何绘制三维图形。

由于 OpenGL 专门为工作站设计，相对于移动设备来说，体积太大了。所以 Android 实现了 OpenGL 的一个子集——OpenGL ES，它包含了 OpenGL 的主要功能，但是体积却大大地缩小。

5.1 三维图形基础

5.1.1 三维图形程序框架

我们小时候，玩过一种活动书，书里面画有小人的各种动作，当我们以极快的速度翻阅时，看起来小人就像动起来一样，这主要是人眼的视觉残像造成的。我们观看电影也是同样的道理，电影胶片以每秒 24 幅画面显现的时候，我们就感觉画面动起来了。其中，每一幅画面，我们称为一帧。

我们绘制三维动画也一样，其框架如图 5-2 所示：

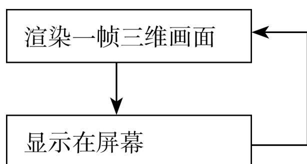

图 5-2 三维图形程序框架

如上图 5-2 所示，这个程序是一个死循环，每循环一次，就渲染、显示一帧画面，如果每秒钟至少循环 24 次，那画面就会很流畅。

5.1.2 三维图形渲染流水线

工厂里面的生产流水线，生产原料通过一端进入流水线，经过一道一道工序的处理之后，最终产出成品。如图 5-2 中“渲染一帧三维画面”的过程，就如同一条生产流水线。

三维图形渲染流水线，也有输入的原料和最终生产出来的产品。三维图形渲染流水线最终生产出来的产品就是一帧三维画面，如前所述，这条流水线每秒钟至少要生产 24 帧画面。那么流水线的生产原料是什么？在流水线中，我们又需要做一些什么控制工作呢？

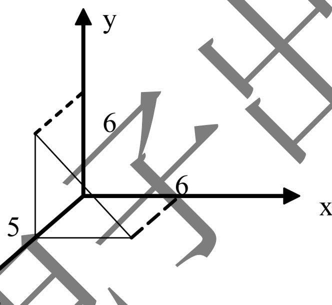

图 5-3 三角形顶点

首先，我们来看一下原料。原料就是各个顶点的数据，包括三维坐标、纹理坐标等。如我们要绘制一个如图 5-3 所示的最基本的多边形——三角形，要应该设置三个顶点的数据，如表 5-1 所示：

表 5-1 顶点数据

序号	x	y	z
0	0	0	5
1	0	6	5
2	6	0	5

我们需要把这些数据存放在一个数组中，然后把数组放入生产线就可以了。如果我们能够画一个三角形，那么在复杂的三维图形也不在话下，因为复杂的图形也是由一个一个基本的三角形构建而成。

把原料放入渲染流水线之后，流水线的机器就可以开始启动，生成一帧一帧的三维画面了，但是在这之前，我们还需要对流水线的机器做一些简单的设置。如图 5-4，显示了三维图形渲染流水线的各道基本工序。

而显示硬件像流水线的工人一样，把上游送来的中间件处理之后，送往下一道工序。当然

东软电子出版社

这个过程是由系统自动完成的，我们只需要设置一些基本的参数即可。

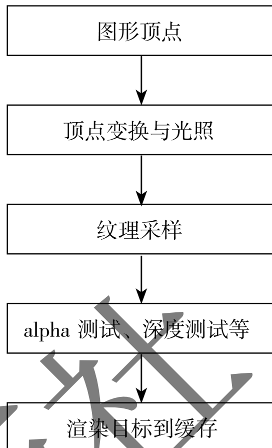

图 5-4　渲染流水线

【课堂实训 5-1】搭建三维程序

在 Android 手机上面创建一个三维图形绘制项目，利用子线程来进行三维图形的具体绘制工作，并且初始化硬件设备。

(1)创建三维图形绘制项目，使用如下信息：

```
Project name:myOpenGL
Package name:org.example.myOpenGL
Activity name:myOpenGL
Application name:myOpenGL
```

打开 myOpenGL.java 文件，窗体显示视图由默认的 main.xml 修改为 GLView，代码如下所示：

```
package org.example.myOpenGL;

import android.app.Activity;
import android.os.Bundle;

public class myOpenGL extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new GLView(this));  //把显示视图改为 GLView
    }
}
```

(2)创建绘制子线程。

GLView 是用于三维图形绘制的视图类，功能类似于第 4 章的 pView 类。创建 GLView 类，GLView 类要继承自 SurfaceView 类，而不是 View 类(这是和 pView 类不同的地方)，GLView 类中因为继承自 SurfaceView，所以需要三个回调函数 surfaceCreated、surfaceDestroy 和 surfaceChanged。修改 GLView.java 代码如下：

```
package org.example.myOpenGL;

import android.content.Context;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.util.Log;

//注意 GLView 要继承自 SurfaceView 类
public class GLView extends SurfaceView implements SurfaceHolder.Callback{
    private GLThread glThread;  //创建 GLThread 类实例
    GLView(Context context){
```

华软电子出版社

```
        super(context);
        getHolder().addCallback(this);
        getHolder().setType(SurfaceHolder.SURFACE_TYPE_GPU);
    }

    public void surfaceCreated(SurfaceHolder holder){
        glThread=new GLThread(this);
        glThread.start();
    }
    public void surfaceDestroyed(SurfaceHolder holder){
        glThread.requestExitAndWait();
        glThread=null;
    }
    public void surfaceChanged(SurfaceHolder holder,int format,int w,int h){
    }
}
```

在以上代码中,创建了一个新的类实例 glThread,注意,我们在调用 glThread 的时候使用了 glThread.start()方法开启这个线程。

(3)创建绘制子线程 GLThread 类,修改代码如下:

```
package org.example.myOpenGL;

import android.app.Activity;
import android.content.Context;

//注意 GLThread 类要继承自 Thread 父类
public class GLThread extends Thread{
    private final GLView view;
    private boolean dlength=false;

    GLThread(GLView view){
        this.view=view;
    }

    @Override
    public void run(){
    }
    public void requestExitAndWait(){
        dlength=true;
        try{
            join();
        }catch(InterruptedException ex){
        }
```

东软电子出版社

```
    }
}
```

GLThread 类要继承自 Thread 类，这样它才能作为一个子线程，而我们需要编写的是 run 函数。当启动以 GLThread 作为入口函数的子线程的时候，程序就会转到 run 函数执行，如图 5-5 所示：

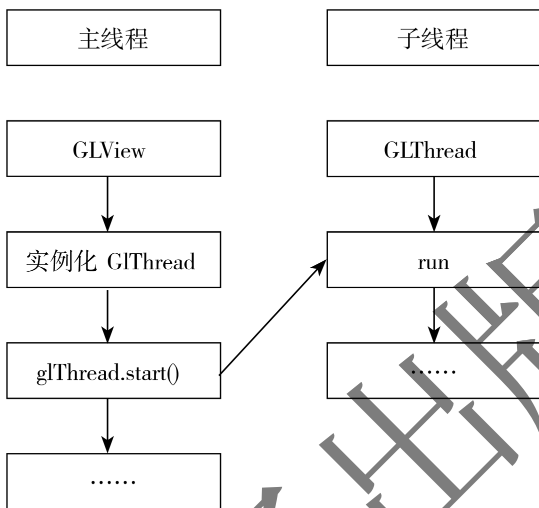

图 5-5　多线程管理

GLView 是主线程，GLThread 是子线程，是真正的绘制函数，绘制代码就写在 GLThread 类中。

也许有人会有疑问，那我们在本节创建 GLThread 类作子线程有何意义，start 和直接函数调用又有什么分别。

而我们在这一节所学习多线程，就是让我们的程序学会多线程同时运行。在我们以前的单线程程序中，主函数 GLView 调用 GLThread 类 run 函数时，主函数会在调用处一直处于等待状态，等待 run 函数调用结束，才能执行接下来的代码，也就是说程序在任一时刻只能执行一个函数。而通过多线程处理，主线程和子线程可以同时运行，主线程不必等待子线程的调用结束。

而这对于三维图形绘制是非常重要的，大家回顾一下图 5-2 的三维图形程序框架，知道三维图形程序的绘制代码是一个死循环，所以如果我们让主线程 GLView 陷入到死循环中显然不是一个好主意，因此我们需要创建一个子线程 GLThread，让子线程去做死循环，从而把主线程解放出来继续响应用户。这也是我们为什么大费周折创建子线程的原因。

(4)初始化设备。

在进行三维图形绘制之前，需要对显示设备进行一些初始化工作。打开 GLThread.java，修改代码如下：

```
package org.example.myopengl;

//需要导入一些 OpenGL 包
import javax.microedition.khronos.egl.EGL10;
import javax.microedition.khronos.egl.EGL11;
import javax.microedition.khronos.egl.EGLConfig;
```

东软电子出版社

```
import javax.microedition.khronos.egl.EGLContext;
import javax.microedition.khronos.egl.EGLDisplay;
import javax.microedition.khronos.egl.EGLSurface;
import javax.microedition.khronos.opengles.GL10;
import android.app.Activity;
import android.content.Context;
import android.opengl.GLU;

public class GLThread extends Thread{
    private final GLView view;
    private boolean dlength=false;
    GLThread(GLView view){
    this.view=view;
    }

    @Override
    public void run(){
        EGL10 egl=(EGL10) EGLContext.getEGL();
        EGLDisplay display=
                                            egl.eglGetDisplay(EGL10.EGL_DEFAULT_DISPLAY);
        int[] version=new int[2];
        egl.eglInitialize(display,version);
        int[] configSpec={EGL10.EGL_RED_SIZE,5,
            EGL10.EGL_GREEN_SIZE,6,EGL10.EGL_BLUE_SIZE,5,
            EGL10.EGL_DEPTH_SIZE,16,EGL10.EGL_NLENGTH };
        EGLConfig[]  configs=new EGLConfig[1];
        int[] numConfig=new int[1];
        egl.eglChooseConfig(display,configSpec,configs,1,numConfig);
        EGLConfig config=configs[0];
        EGLContext glc=
                        egl.eglCreateContext(display,config,EGL10.EGL_NO_CONTEXT,null);
        EGLSurface surface=
                        egl.eglCreateWindowSurface(display,config,view.getHolder(),null);
        egl.eglMakeCurrent(display,surface,surface,glc);

        GL10 gl=(GL10)(glc.getGL());   //绘制设备
        init(gl);
        while(! dlength){
            drawFrame(gl);   //绘制一帧图像
            egl.eglSwapBuffers(display,surface);   //把图像翻转到屏幕上显示
            if(egl.eglGetError() == EGL11.EGL_CONTEXT_LOST){
                Context c=view.getContext();
```

东软电子出版社

```
                if(c instanceof Activity){
                    ((Activity) c).finish();
                }
            }
        }
        egl.eglMakeCurrent(display,EGL10.EGL_NO_SURFACE,
            EGL10.EGL_NO_SURFACE,EGL10.EGL_NO_CONTEXT);
        egl.eglDestroySurface(display,surface);
        egl.eglDestroyContext(display,glc);
        egl.eglTerminate(display);
    }
    private void init(GL10 gl){
        float ratio=(float) view.getWidth() / view.getHeight();
        GLU.gluPerspective(gl,45.0f,ratio,1,100f);  //设置摄像机的宽高比等
        gl.glEnable(GL10.GL_DEPTH_TEST);
        gl.glDepthFunc(GL10.GL_LEQUAL);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    }
    private void drawFrame(GL10 gl){
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT|GL10.GL_DEPTH_BUFFER_BIT);
    }
    public void requestExitAndWait(){
        dlength=true;
        try{
            join();
        }catch(InterruptedException ex){
        }
    }
}
```

在以上代码中，首先在 run 函数中获取本机的 OpenGL 版本以及配置信息；接着在 run 函数中定义一个名为 gl 的 GL10 变量，这相当于绘制设备，接下来的绘制工作就要依靠 gl 变量；然后利用 init 函数对 gl 变量进行一些设置，包括屏幕的宽高比、深度测试等；最后在 run 函数中进入到一个 while 循环，每循环一次就利用 gl 设备进行一帧三维图形的绘制，在这里我们把绘制代码放到 drawframe 函数中，draeframe 函数在内存中绘制完一帧图像就将其翻转到屏幕上进行显示。

在这里需要介绍 init 函数中的系统函数 GLU.gluPerspective()。在三维图形渲染中，显示在屏幕上的不是场景全部，而是通过一个虚拟摄像机去观察场景，然后把观察到的部分显示在屏幕上，如图 5-6 所示。

因此场景中的哪些地方能够被摄像机看见，取决于摄像机的位置和朝向。除此以外，还和摄像机的可视范围有关，摄像机的可视范围其形状犹如一个金字塔去掉了塔顶，称之为平截台体，如图 5-7 所示，GLU.gluPerspective()函数的功能就是设置平截台体。位于平截台体范围

东软电子出版社

内的物体就是最终显示在屏幕上的物体。

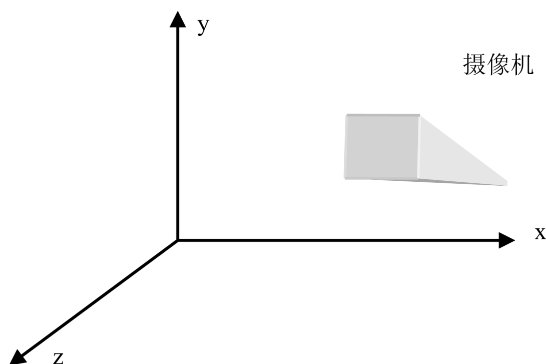

图 5-6 虚拟摄像机

GLU. gluPerspective()函数原型如下：

```
void GLU.gluPerspective(GL10 gl,
float fovy,
float aspect,
float zNear,
float zFar)
```

其中 fovy 表示摄像机张开的角度(一般设为 45 度)；aspect 表示宽高比，这里应该和手机屏幕的宽高比一致，否则图像会产生变形；zNear 表示摄像机能看到的最近距离；zFar 表示摄像机能看到的最远距离。

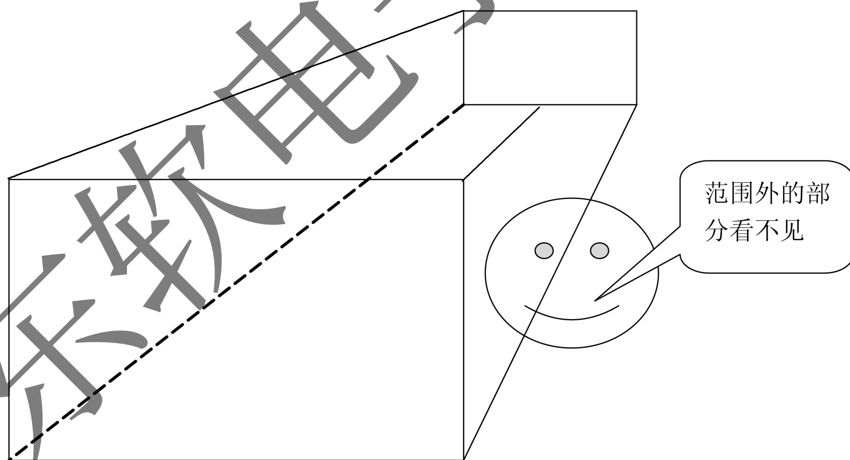

图 5-7 平截台体

最后我们运行程序，如果出现黑屏，则说明运行成功。

东软电子出版社

5.2　绘制三维图形

5.2.1　设置立方体顶点

三维渲染就像是一条流水线，在本节设置的立方体顶点数据，就相当于流水线的生产原料。

以实训 5-1 的项目为例，创建名为 GLCube 的 java 类。该类的主要作用是描述立方体的基本顶点数据。

然后，打开 GLCube.java 文件，编写代码如下：

```
package org.example.myopengl;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.IntBuffer;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
public class GLCube {
    private final IntBuffer mVertexBuffer;
    public GLCube(){
        int length=60000;
        int vertices[]={
            // FRONT
            -length/2, -length/2, length/2, length/2, -length/2, length/2,
            -length/2, length/2, length/2, length/2, length/2, length/2,
            // BACK
            -length/2, -length/2, -length/2, -length/2, length/2, -length/2,
            length/2, -length/2, -length/2, length/2, length/2, -length/2,
            // LEFT
            -length/2, -length/2, length/2, -length/2, length/2, length/2,
            -length/2, -length/2, -length/2, -length/2, length/2, -length/2,
            // RIGHT
            length/2, -length/2, -length/2, length/2, length/2, -length/2,
            length/2, -length/2, length/2, length/2, length/2, length/2,
            // TOP
            -length/2, length/2, length/2, length/2, length/2, length/2,
            -length/2, length/2, -length/2, length/2, length/2, -length/2,
            // BOTTOM
            -length/2, -length/2, length/2, -length/2, -length/2, -length/2,
            length/2, -length/2, length/2, length/2, -length/2, -length/2, };
```

东软电子出版社

```
        ByteBuffer vbb=ByteBuffer.allocateDirect(vertices.length/2 * 4);
        vbb.order(ByteOrder.nativeOrder());
        mVertexBuffer=vbb.asIntBuffer();
        mVertexBuffer.put(vertices);
        mVertexBuffer.position(0);
    }
    public void draw(GL10 gl){
        gl.glVertexPointer(3,GL10.GL_FIXED,0,mVertexBuffer);
        gl.glColor4f(1,1,1,1);
        gl.glNormal3f(0,0,1);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP,0,4);
        gl.glNormal3f(0,0,-1);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP,4,4);
        gl.glColor4f(1,1,1,1);
        gl.glNormal3f(-1,0,0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP,8,4);
        gl.glNormal3f(1,0,0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP,12,4);
        gl.glColor4f(1,1,1,1);
        gl.glNormal3f(0,1,0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP,16,4);
        gl.glNormal3f(0,-1,0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP,20,4);
        }
    }
```

在 GLCube 类中，创建了两个函数，其中一个是构造函数，把设置顶点数据的代码放在其中；另一个是 draw 函数，把调用顶点数据进行绘制的代码放在其中。

以绘制立方体的前面为例，说明顶点数据应该如何设置。前面共有 4 个顶点，每个顶点包括 x、y、z 三维坐标，所以一共 12 个数据。每三个数据为一组，表示一个顶点的三维坐标，其对应关系如图 5-8 所示。其余各面以此类推。

接下来我们再以前面的绘制为例，说明在 draw 函数中如何调用顶点缓冲区的数据进行绘制。

```
gl.glColor4f(1,1,1,1);
gl.glNormal3f(0,0,1);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP,0,4);
```

其中 gl.glColor4f(1,1,1,1)，表示设置顶点颜色为白色；

gl.glNormal3f(0,0,1)，设置该面的法线向量方向为(0,0,1)。

gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP,0,4)中第一个参数 GL10.GL_TRIANGLE_STRIP 表示绘制三角形，第二个参数 0 和第三个参数 4，表示从顶点缓冲区的第 0 号顶点开始连续取出 4 个顶点数据(即图 5-8 所示的 4 个顶点)来进行三角形的绘制，共计 2 个三角形。其余面以此类推。

华东电子出版社

```
// FRONT
        -length, -length, length, length, -length, length,
        -length, length, length, length, length, length,
```

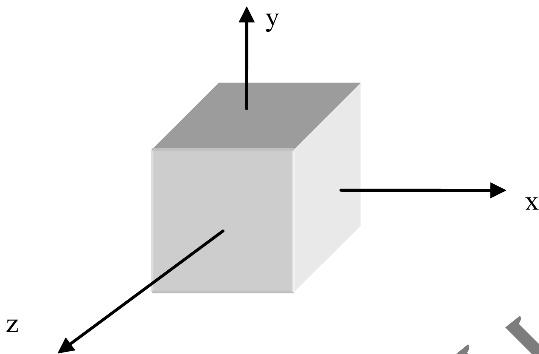

图 5-8　顶点数据的设置

最后，我们在 GLThread 类中实例化 GLCube，然后再调用其绘制函数即可，修改 GLThread.java 代码如下所示：

```
public class GLThread extends Thread{
    ……
private final GLCube cube=new GLCube();   //实例化立方体类
    ……
    @Override
    public void run(){
    ……
    }
    private void init(GL10 gl){
    ……
    }
    private void drawFrame(GL10 gl){
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT|GL10.GL_DEPTH_BUFFER_BIT);
        //添加立方体的绘制代码
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glTranslatef(0f,0,-3.0f);
        cube.draw(gl);
    }
    public void requestExitAndWait(){
        ……
    }
}
```

东软电子出版社

在代码中，函数 glTranslatef 表示在 x，y，z 方向平移图形，如果以后我们需要让图形移动，就需要使用此函数。

最后运行程序，结果如图 5-9 所示。

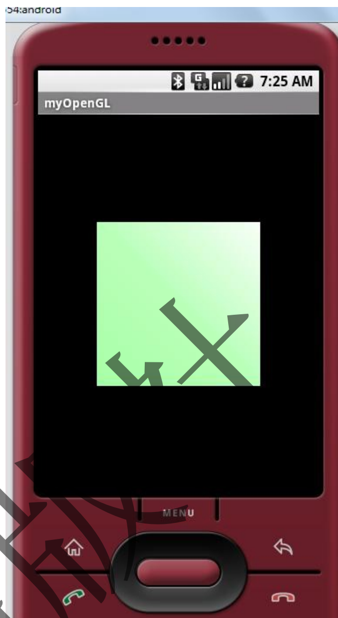

图 5-9　绘制出来的立方体

5.2.2　让立方体动起来

让立方体动起来，方法有很多。第一种就是直接修改立方体的坐标，但是这种方法每一次都要重新设置顶点缓冲区。更为科学的方法是让立方体的坐标乘上相应的变化矩阵就可以，变化矩阵包括平移矩阵、旋转矩阵和缩放矩阵，这其中的原理，大家可以自行查看计算机图形学。OpenGL 已经把相关的矩阵运算封装起来了，我们要做的仅仅是设置平移的位移、旋转的角度等几个参数就可以了。接下来，就让立方体旋转起来。

之前，已经通过 gl.glTranslatef(0f,0,－3.0f)让立方体进行了平移，接下来，我们就让立方体旋转。立方体的旋转，是通过 gl.glRotatef 函数实现的，该函数原型如下：

```
void GL10.glRotatef(float angle,
float x,
float y,
float z)
```

其中第一个参数 angle 表示旋转的角度，第 2,3,4 个参数 x、y、z 表示绕着哪个轴进行旋转，我们让立方体绕着 y 轴旋转，修改 GLThread.java 中的 drawframe 函数。

```
private void drawFrame(GL10 gl){
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_
BUFFER_BIT);
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0.f,0,－3.0f);
    gl.glRotatef(60,0,1,0);
    cube.draw(gl);
}
```

注意，glRotatef 和 glTranslatef 等代码要写在 cube.draw(gl)之前，这样平移和旋转的效果才会施加在立方体上面；glLoadIdentity 函数是导入单位矩阵，它的几何意义在于消除前面平移、旋转矩阵的影响（如果有的话），否则将与现有的平移、旋转效果相叠加。运行程序，效果如图 5-10 所示。

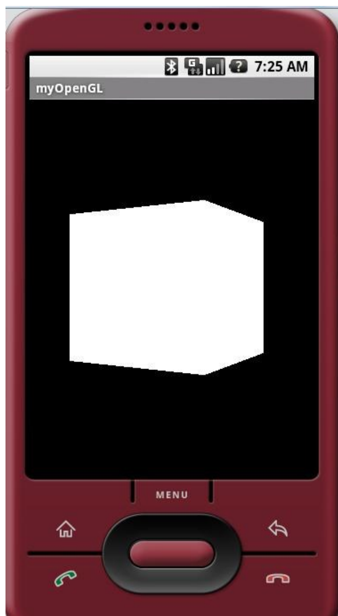

图 5-10　立方体旋转

5.2.3　给立方体加上纹理贴图

现在立方体已经绘制完成了，但是表面的颜色依然显得很单调，不够真实。接下来，需要给立方体加上纹理贴图。所谓纹理，其实就是图片；所谓贴图，就是把图片贴在物体上。图片的形状是矩形，因此，只需要把图片 4 个顶点对应到立方体某个面的 4 个顶点，剩下的映射工作交由系统自动完成。以立方体的前面为例，纹理图片的对应关系，如图 5-11 所示。

所以，我们需要按照顶点缓冲区中顶点的次序，依次设置好对应的图片的纹理坐标就可以了。

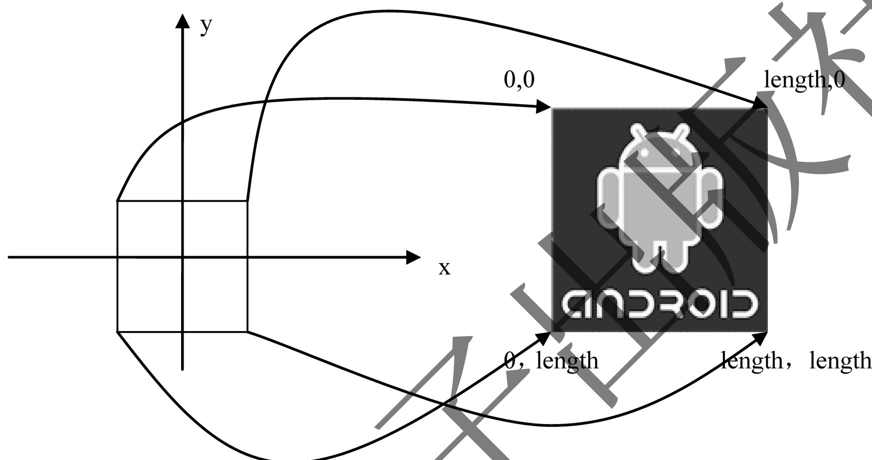

图 5-11　纹理坐标对应关系

修改 GLCube.java，代码如下：

```
public class GLCube {
    ……
    private final IntBuffer mTextureBuffer;
    public GLCube(){
        ……
        int texCoords[]={
            // FRONT
            0, length, length, length, 0, 0, length, 0,
            // BACK
            length, length, length, 0, 0, length, 0, 0,
            // LEFT
            length, length, length, 0, 0, length, 0, 0,
            // RIGHT
            length, length, length, 0, 0, length, 0, 0,
            // TOP
            length, 0, 0, 0, length, length, 0, length,
            // BOTTOM
            0, 0, 0, length, length, 0, length, length, };
```

东软电子出版社

```
        ByteBuffer tbb=
        ByteBuffer.allocateDirect(texCoords.length * 4);
        tbb.order(ByteOrder.nativeOrder());
        mTextureBuffer=tbb.asIntBuffer();
        mTextureBuffer.put(texCoords);
        mTextureBuffer.position(0);
    }

    static void loadTexture(GL10 gl,Context context,int resource){
        Bitmap bmp =
        BitmapFactory.decodeResource(context.getResources(),resource);
        ByteBuffer bb=extract(bmp);
        load(gl,bb,bmp.getWidth(),bmp.getHeight());
    }

    private static ByteBuffer extract(Bitmap bmp){
        ByteBuffer bb=
        ByteBuffer.allocateDirect(bmp.getHeight() * bmp.getHeight() * 4);
        bb.order(ByteOrder.BIG_ENDIAN);
        IntBuffer ib=bb.asIntBuffer();
        for(int y=bmp.getHeight() - 1;y > -1; y--)
            for(int x=0;x < bmp.getWidth();x++){
                int pix=bmp.getPixel(x,bmp.getHeight() - y - 1);
                int red=((pix >> 16) & 0xFF);
                int green=((pix >> 8)& 0xFF);
                int blue=((pix ) & 0xFF);
                ib.put(red << 24 | green << 16 | blue << 8 |
                ((red + blue + green)/3));
            }
        bb.position(0);
        return bb;
    }
    private static void load(GL10 gl, ByteBuffer bb,int width,int height){
        int[] tmp_tex=new int[1];
        gl.glGenTextures(1,tmp_tex,0);
        int tex=tmp_tex[0];
        gl.glBindTexture(GL10.GL_TEXTURE_2D,tex);
        gl.glTexImage2D(GL10.GL_TEXTURE_2D,
        0,
        GL10.GL_RGBA,
        width,
        height,
```

东软电子出版社

```
            0,
            GL10.GL_RGBA,
            GL10.GL_UNSIGNED_BYTE,bb);
            gl.glTexParameterx(GL10.GL_TEXTURE_2D,
            GL10.GL_TEXTURE_MIN_FILTER,
            GL10.GL_LINEAR);
            gl.glTexParameterx(GL10.GL_TEXTURE_2D,
            GL10.GL_TEXTURE_MAG_FILTER,
            GL10.GL_LINEAR);
        }
        public void draw(GL10 gl){
            gl.glTexCoordPointer(2,GL10.GL_FIXED,0,mTextureBuffer);
            ……
        }
    }
```

其中,loadTexture 函数的作用是把磁盘上的一张图片作为纹理,装载到内存中准备贴图。mTextureBuffer 是设置的纹理坐标缓冲区。

然后,我们再修改 GLThread.java,调用纹理贴图函数 loadTexture。代码如下:

```
    public class GLThread extends Thread{
    ……
        private void init(GL10 gl){
        ……
            gl. glEnableClientState ( GL10. GL _ TEXTURE _ COORD _
        ARRAY);
            gl.glEnable(GL10.GL_TEXTURE_2D);
            GLCube.loadTexture(gl,
            view.getContext(),
            R.drawable.android);
        }
    }
```

注意,loadTexture 的第 3 个参数是图片的路径和名称,如果图片名称不是 android.png,那么请更换图片名称。

最后,运行程序,观察结果如图 5-12 所示。

图 5-12　纹理贴图效果

5.2.4　给立方体加上光照

要让物体呈现出更加真实的效果,我们可以再加上光照的效果。那么什么是光照呢?所谓光照就是把光的颜色值按照某种计算方式进行运算后,得到的真实照射在物体表面的光线颜色值。按照光照的计算方式,光照效果可以分为环境光、漫反射和镜面反射。所谓环境光,则是指光的颜色均匀的分布在场景中;所谓漫反射,指光线照射在物体表面,由于表面的不光滑而呈现出漫反射的效果;所谓镜面反射,指物体表面很光滑,比如金属,则此时光照的效果还和观察者

东软电子出版社

的角度相关。对于具体这三种光照效果的计算方式，我们在这里不必深究，我们只需要设置好三种光照效果的颜色即可，具体的计算交由渲染管线自动完成。

我们在一个场景中，可以有一种或者多种光照效果，叠加出来的结果就是最后的光照效果值。得到光照效果值之后，我们还需要设置物体表面的材质，所谓材质就是判定光线照射在物体表面，被吸收了多少？又被反射了多少？最后被反射的光线颜色值再叠加到物体表面的纹理颜色值，就是物体表面真实的色彩了。

接下来，我们就给物体表面加上环境光和漫反射两种效果，打开 GLThread.java，修改 init()函数中的代码如下：

```
public class GLThread extends Thread{
……

    private void init(GL10 gl){
    ……
        //设置两种光照效果的颜色值
        float lightAmbient[]=new float[]{0.2f,1f,0.2f,1};
        float lightDiffuse[]=new float[]{1,1,1,1};
        //设置漫反射的光源位置
        float[] lightPos=new float[]{1,1,1,1};
        //启动光照效果
        gl.glEnable(GL10.GL_LIGHTING);
        gl.glEnable(GL10.GL_LIGHT0);
        //应用设置的光照颜色和位置
        gl.glLightfv(GL10.GL_LIGHT0,
        GL10.GL_AMBIENT,
        lightAmbient,
        0);
        gl.glLightfv(GL10.GL_LIGHT0,
        GL10.GL_DIFFUSE,
        lightDiffuse,
        0);
        gl.glLightfv(GL10.GL_LIGHT0,GL10.GL_POSITION,lightPos,0);
        //设置材质
        float matAmbient[]=new float[]{1,1,1,1};
        float matDiffuse[]=new float[]{1,0,1,1};
        //应用材质
        gl.glMaterialfv(GL10.GL_FRONT_AND_BACK,
        GL10.GL_AMBIENT,
        matAmbient,
        0);
        gl.glMaterialfv(GL10.GL_FRONT_AND_BACK,
        GL10.GL_DIFFUSE,
```

东软电子出版社

```
            matDiffuse,
            0);
            ……
        }
    ……
}
```

其中,lightAmbient为环境光颜色值,lightDiffuse为漫反射颜色值,matAmbient为针对环境光的材质,matDiffuse为针对漫反射的材质,最后运行程序,观察光照效果。大家也可以根据自己的需要来设定颜色值和材质,并观察最终效果的变换。

【课堂实训5-2】创建两个立方体

绘制两个立方体,分别绕着x轴和y轴进行旋转。利用GLCube类,实例化两个立方体,利用平移矩阵,各自移开一定的距离,然后分别给每个立方体实例应用旋转矩阵。如果需要立方体不停地旋转,则旋转角度应该设置为一个不断自加的变量,此处可以采用系统时间。

相关修改的核心代码如下:

```
private void drawFrame(GL10 gl){
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT|GL10.GL_DEPTH_BUFFER_BIT);
    GLU.gluLookAt(gl,0,0,0,0,0,-1,0,1,0);
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(1.5f,0,-9.0f);
    long elapsed=System.currentTimeMillis() - startTime;
    gl.glRotatef(60,0,1,0);
    gl.glRotatef(elapsed * (30f / 1000f),0,1,0);
    gl.glRotatef(elapsed * (15f / 1000f),1,0,0);
    cube.draw(gl);
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(-1.5f,0,-9);
    gl.glRotatef(elapsed * (30f / 1000f),0,1,0);
    othercube.draw(gl);
}
```

本章小结

本章主要介绍了Android平台下的三维图形编程的相关知识。三维图形编程在手机应用程序开发中有着比较重要的地位,特别是在手机游戏开发中。本章对Android平台下的OpenGL图形函数库进行了一些简单的介绍,用户可以据此绘制三维图形。

东软电子出版社

单元实训　实现两个立方体的碰撞检测

绘制两个立方体，利用平移矩阵，各自移开一定的距离，然后同时向中间运动，两个立方体接触后向相反的方向弹出，立方体的外边缘触到屏幕的边缘后弹回，然后再向中间运动，依此循环下去，实现两个立方体的碰撞运动。

本章习题

一、填空题

1. 在 Android 中，三维图形函数库采用______________。

2. 要使得图形视觉效果流畅，每秒钟至少渲染______________帧三维画面。

3. 子线程类要继承自______________父类。

4. 基本光照模型包括______________、______________、______________。

二、简答题

1. 请简单说明采用子线程进行渲染的好处。

2. 某物体，其材质为反射 80%红光、70%绿光、90%蓝光，应该如何设置其材质。

3. 某摄像机张开角度为 60，视口比例为 4:3，最近距离为 5，最远距离为 100，应该如何设置。

4. 让物体沿着 x 轴移动 5 个单位，沿着 z 轴负方向移动 7 个单位，应该如何设置平移矩阵。

东软电子出版社