

# Android OpenGL ES 简明开发教程小结

1. [Android OpenGL ES 简明开发教程一：概述](#)
2. [Android OpenGL ES 简明开发教程二：构造 OpenGL ES View](#)
3. [Android OpenGL ES 简明开发教程三：3D 绘图基本概念](#)
4. [Android OpenGL ES 简明开发教程四：3D 坐标变换](#)
5. [Android OpenGL ES 简明开发教程五：添加颜色](#)
6. [Android OpenGL ES 简明开发教程六：真正的 3D 图形](#)
7. [Android OpenGL ES 简明开发教程七：材质渲染](#)

## Android OpenGL ES 简明开发教程一：概述

ApiDemos 的 Graphics 示例中含有 OpenGL ES 例子，OpenGL ES 主要用来开发 3D 图形应用的。OpenGL ES (OpenGL for Embedded Systems) 是 OpenGL 三维图形 API 的子集，针对手机、PDA 和游戏主机等嵌入式设备而设计。

下面是维基百科中对应 OpenGL ES 的简介：

OpenGL ES 是从 OpenGL 裁剪定制而来的，去除了 glBegin/glEnd，四边形（GL\_QUADS）、多边形（GL\_POLYGONS）等复杂图元等许多非绝对必要的特性。经过多年发展，现在主要有两个版本，OpenGL ES 1.x 针对固定管线硬件的，OpenGL ES 2.x 针对可编程管线硬件。OpenGL ES 1.0 是以 OpenGL 1.3 规范为基础的，OpenGL ES 1.1 是以 OpenGL 1.5 规范为基础的，它们分别又支持 common 和 common lite 两种 profile。lite profile 只支持定点实数，而 common profile 既支持定点数又支持浮点数。OpenGL ES 2.0 则是参照 OpenGL 2.0 规范定义的，common profile 发布于 2005-8，引入了对可编程管线的支持。

在解析 Android ApiDemos 中 OpenGL ES 示例前，有必要对 OpenGL ES 开发单独做个简明开发教程，可以帮助从未接触过 3D 开发的程序员了解 OpenGL 的开发的基本概念和方法，很多移动手机平台都提供了对 OpenGL ES 开发包的支持，因此尽管这里使用 Android 平台介绍 OpenGL ES，但基本概念和步骤同样适用于其它平台。

简明开发教程主要参考 [Jayway Team Blog 中 OpenGL ES 开发教程](#)，这是一个写的比较通俗易懂的开发教程，适合 OpenGL ES 初学者。

除了这个 OpenGL ES 简明开发教程外，以后将专门针对 OpenGL ES 写个由浅入深的开发教程，敬请关注。

## Android OpenGL ES 简明开发教程二：构造 OpenGL ES View

在 Android 平台上构造一个 OpenGL View 非常简单，主要有两方面的工作：

### GLSurfaceView

Android 平台提供的 OpenGL ES API 主要定义在包 `android.opengl`、`javax.microedition.khronos.egl`、`javax.microedition.khronos.opengles`、`java.nio` 等几个包中，其中类 `GLSurfaceView` 为这些包中的核心类：

- 起到连接 OpenGL ES 与 Android 的 View 层次结构之间的桥梁作用。
- 使得 OpenGL ES 库适应于 Android 系统的 Activity 生命周期。
- 使得选择合适的 Frame buffer 像素格式变得容易。
- 创建和管理单独绘图线程以达到平滑动画效果。
- 提供了方便使用的调试工具来跟踪 OpenGL ES 函数调用以帮助检查错误。

因此编写 OpenGL ES 应用的起始点是从类 `GLSurfaceView` 开始，设置 `GLSurfaceView` 只需调用一个方法来设置 OpenGLView 用到的 `GLSurfaceView.Renderer`。

### [帮助](#)

```
1 public void setRenderer(GLSurfaceView.Renderer renderer)
```

### GLSurfaceView.Renderer

`GLSurfaceView.Renderer` 定义了一个统一图形绘制的接口，它定义了如下三个接口函数：

## [帮助](#)

```
1      // Called when the surface is created or recreated.

2      public void onSurfaceCreated(GL10 gl, EGLConfig config)

3

4      // Called to draw the current frame.

5      public void onDrawFrame(GL10 gl)

6

7      // Called when the surface changed size.

8      public void onSurfaceChanged(GL10 gl, int width, int height)
```

- **onSurfaceCreated**：在这个方法中主要用来设置一些绘制时不常变化的参数，比如：背景色，是否打开 **z-buffer** 等。
- **onDrawFrame**：定义实际的绘图操作。
- **onSurfaceChanged**：如果设备支持屏幕横向和纵向切换，这个方法将发生在横向<->纵向互换时。此时可以重新设置绘制的纵横比率。

有了上面的基本定义，可以写出一个 **OpenGL ES** 应用的通用框架。

创建一个新的 **Android** 项目:**OpenGLESTutorial**, 在项目添加两个类 **TutorialPartI.java** 和 **OpenGLRenderer.java**.

具体代码如下：

### **TutorialPartI.java**

## [帮助](#)

```
1      public class TutorialPartI extends Activity {

2          // Called when the activity is first created.
```

```

3      @Override

4      public void onCreate(Bundle savedInstanceState) {

5          super.onCreate(savedInstanceState);

6          this.requestWindowFeature(Window.FEATURE_NO_TITLE); // (NEW)

7          getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,

8          WindowManager.LayoutParams.FLAG_FULLSCREEN); // (NEW)

```

## Tutorial Part I

```

9

10     GLSurfaceView view = new GLSurfaceView(this);

11     view.setRenderer(new OpenGLRenderer());

12     setContentView(view);

13 }

14 }

```

### OpenGLRenderer.java

#### [帮助](#)

```

1      public class OpenGLRenderer implements Renderer {

2

3          public void onSurfaceCreated(GL10 gl, EGLConfig config) {

4              // Set the background color to black ( rgba ).

5              gl.glClearColor(0.0f, 0.0f, 0.0f, 0.5f); // OpenGL docs.

6              // Enable Smooth Shading, default not really needed.

```

```
7      gl.glShadeModel(GL10.GL_SMOOTH);// OpenGL docs.

8      // Depth buffer setup.

9      gl.glClearDepthf(1.0f);// OpenGL docs.

10     // Enables depth testing.

11     gl.glEnable(GL10.GL_DEPTH_TEST);// OpenGL docs.

12     // The type of depth testing to do.

13     gl.glDepthFunc(GL10.GL_LEQUAL);// OpenGL docs.

14     // Really nice perspective calculations.

15     gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, // OpenGL docs.

16     GL10.GL_NICEST);

17     }

18

19     public void onDrawFrame(GL10 gl) {

20         // Clears the screen and depth buffer.

21         gl.glClear(GL10.GL_COLOR_BUFFER_BIT | // OpenGL docs.

22         GL10.GL_DEPTH_BUFFER_BIT);

23     }

24

25     public void onSurfaceChanged(GL10 gl, int width, int height) {

26         // Sets the current view port to the new size.
```

```
27     gl.glViewport(0, 0, width, height);// OpenGL docs.

28     // Select the projection matrix

29     gl.glMatrixMode(GL10.GL_PROJECTION);// OpenGL docs.

30     // Reset the projection matrix

31     gl.glLoadIdentity();// OpenGL docs.

32     // Calculate the aspect ratio of the window

33     GLU.gluPerspective(gl, 45.0f,

34         (float) width / (float) height,

35         0.1f, 100.0f);

36     // Select the modelview matrix

37     gl.glMatrixMode(GL10.GL_MODELVIEW);// OpenGL docs.

38     // Reset the modelview matrix

39     gl.glLoadIdentity();// OpenGL docs.

40 }

41 }
```

编译后运行，屏幕显示一个黑色的全屏。这两个类定义了 **Android OpenGL ES** 应用的最基本的类和方法，可以看作是 **OpenGL ES** 的”Hello ,world”应用，后面将逐渐丰富这个例子来画出 **3D** 图型。

框架代码[下载](#)：可以作为你自己的 **OpenGL 3D** 的初始代码。

# Android OpenGL ES 简明开发教程三：3D 绘图基本概念

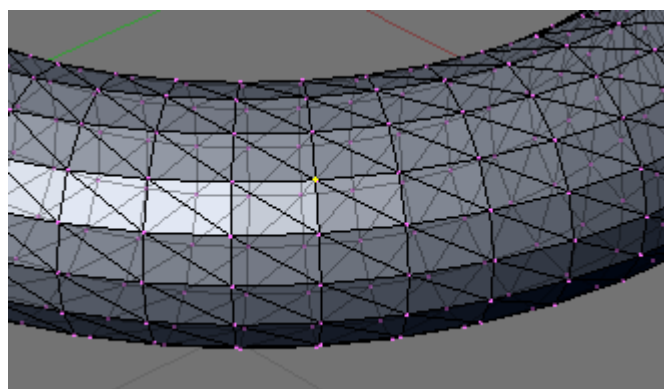
发表于 2011 年 05 月 30 日 由 [guidebee](#)

前面介绍了使用 Android 编写 OpenGL ES 应用的程序框架，本篇介绍 3D 绘图的一些基本构成要素，最终将实现一个多边形的绘制。

一个 3D 图形通常是由一些小的基本元素（顶点，边，面，多边形）构成，每个基本元素都可以单独来操作。

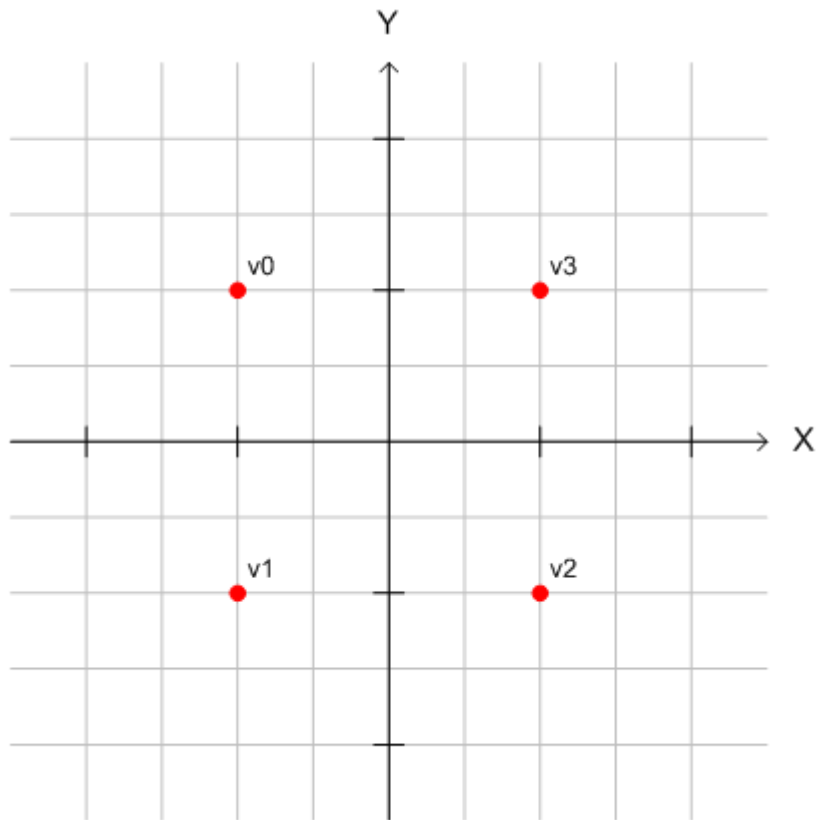
## Vertex （顶点）

顶点是 3D 建模时用到的最小构成元素，顶点定义为两条或是多条边交会的地方。在 3D 模型中一个顶点可以为多条边，面或是多边形所共享。一个顶点也可以代表一个点光源或是 Camera 的位置。下图中标识为黄色的点为一个顶点(Vertex)。



在 Android 系统中可以使用一个浮点数数组来定义一个顶点，浮点数数组通常放在一个 Buffer (java.nio)中来提高性能。

比如：下图中定义了四个顶点对应的 Android 顶点定义：



[帮助](#)

```
1      private float vertices[] = {  
2          -1.0f,  1.0f, 0.0f,  // 0, Top Left  
3          -1.0f, -1.0f, 0.0f,  // 1, Bottom Left  
4          1.0f, -1.0f, 0.0f,  // 2, Bottom Right  
5          1.0f,  1.0f, 0.0f,  // 3, Top Right  
6      };
```

为了提高性能，通常将这些数组存放到 `java.io` 中定义的 `Buffer` 类中：

[帮助](#)

```
1      // a float is 4 bytes, therefore we multiply the
```



```

2      //number if vertices with 4.

3      ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);

4      vbb.order(ByteOrder.nativeOrder());

5      FloatBuffer vertexBuffer = vbb.asFloatBuffer();

6      vertexBuffer.put(vertices);

7      vertexBuffer.position(0);

```

有了顶点的定义，下面一步就是如何将它们传给 OpenGL ES 库，OpenGL ES 提供一个成为“管道 Pipeline”的机制，这个管道定义了一些“开关”来控制 OpenGL ES 支持的某些功能，缺省情况这些功能是关闭的，如果需要使用 OpenGL ES 的这些功能，需要明确告知 OpenGL “管道”打开所需功能。因此对于我们的这个示例，需要告诉 OpenGL 库打开 Vertex buffer 以便传入顶点坐标 Buffer。要注意的使用完某个功能之后，要关闭这个功能以免影响后续操作：

#### [帮助](#)

```

1      // Enabled the vertex buffer for writing and to be used during rendering.

2      gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);// OpenGL docs.

3      // Specifies the location and data format of an array of vertex

4      // coordinates to use when rendering.

5      gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer); // OpenGL docs.

6      When you are done with the buffer don't forget to disable it.

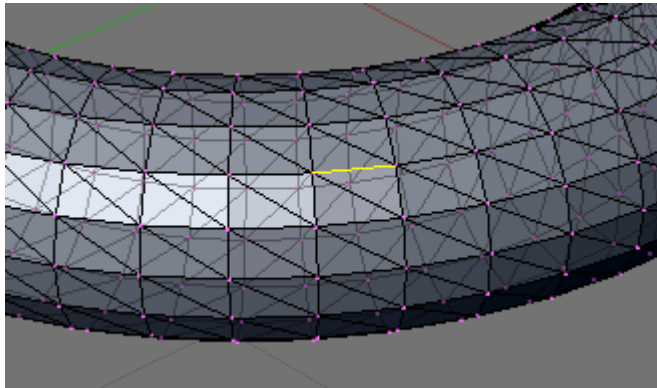
7      // Disable the vertices buffer.

8      gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);// OpenGL docs.

```

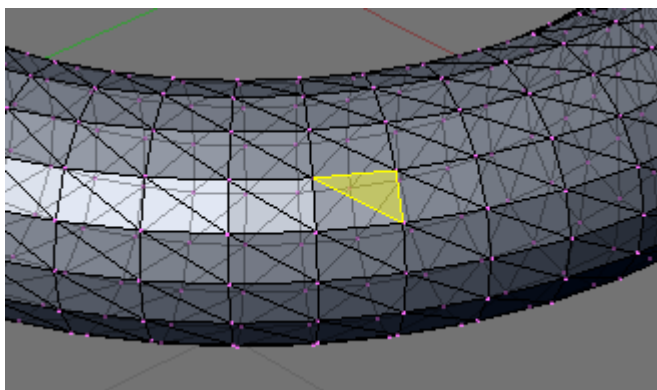
## Edge (边)

边定义为两个顶点之间的线段。边是面和多边形的边界线。在 3D 模型中，边可以被相邻的两个面或是多边形共享。对一个边做变换将影响边相接的所有顶点，面或多边形。在 OpenGL 中，通常无需直接来定义一个边，而是通过顶点定义一个面，从而由面定义了其所对应的三条边。可以通过修改边的两个顶点来更改一条边，下图黄色的线段代表一条边：



## Face (面)

在 OpenGL ES 中，面特指一个三角形，由三个顶点和三条边构成，对一个面所做的变化影响到连接面的所有顶点和边，面多边形。下图黄色区域代表一个面。



定义面的顶点的顺序很重要 在

拼接曲面的时候，用来定义面的顶点的顺序非常重要，因为顶点的顺序定义了面的朝向（前向或是后向），为了获取绘制的高性能，一般情况不会绘制面的前面和后面，只绘制面的“前面”。虽然“前面”“后面”的定义可以应人而易，但一般为所有的“前面”定义统一的顶点顺序(顺时针或是逆时针方向)。

下面代码设置逆时针方法为面的“前面”：

[帮助](#)

```
1          gl.glFrontFace(GL10.GL_CCW);
```

打开 忽略“后面”设置:

[帮助](#)

```
1          gl.glEnable(GL10.GL_CULL_FACE);
```

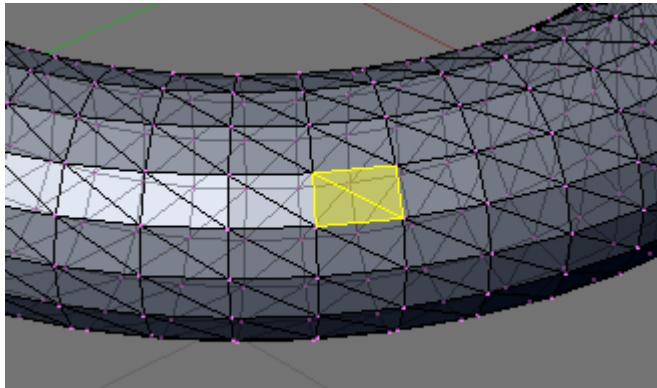
明确指明“忽略“哪个面的代码如下:

[帮助](#)

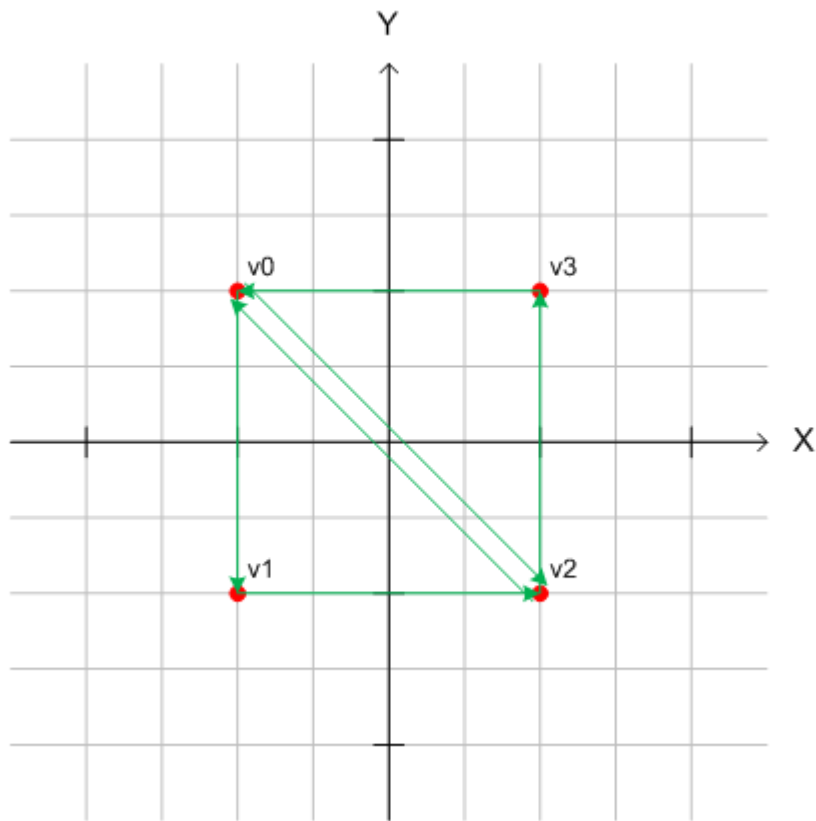
```
1          gl.glCullFace(GL10.GL_BACK);
```

## Polygon （多边形）

多边形由多个面（三角形）拼接而成，在三维空间上，多边形并不一定表示这个 Polygon 在同一平面上。这里我们使用缺省的逆时针方向代表面的“前面 Front)，下图黄色区域为一个多边形。



来看一个多边形的示例在 Android 系统如何使用顶点和 buffer 来定义，如下图定义了一个正方形:



对应的顶点和 buffer

定义代码:

[帮助](#)

```
1 private short[] indices = { 0, 1, 2, 0, 2, 3 };  
2  
2 To gain some performance we also put this ones in a byte buffer.  
3  
3 // short is 2 bytes, therefore we multiply the number if vertices with  
3 2.  
4  
4 ByteBuffer ibb = ByteBuffer.allocateDirect(indices.length * 2);  
5  
5 ibb.order(ByteOrder.nativeOrder());  
6  
6 ShortBuffer indexBuffer = ibb.asShortBuffer();  
7  
7 indexBuffer.put(indices);  
8  
8 indexBuffer.position(0);
```

**Render (渲染)**

我们已定义好了多边形，下面就要了解如和使用 OpenGL ES 的 API 来绘制（渲染）这个多边形了。OpenGL ES 提供了两类方法来绘制一个空间几何图形：

- `public abstract void glDrawArrays(int mode, int first, int count)` 使用 `VertexBuffer` 来绘制，顶点的顺序由 `vertexBuffer` 中的顺序指定。
- `public abstract void glDrawElements(int mode, int count, int type, Buffer indices)`，可以重新定义顶点的顺序，顶点的顺序由 `indices Buffer` 指定。

前面我们已定义里顶点数组，因此我们将采用 `glDrawElements` 来绘制多边形。

同样的顶点，可以定义的几何图形可以有所不同，比如三个顶点，可以代表三个独立的点，也可以表示一个三角形，这就需要使用 `mode` 来指明所需绘制的几何图形的基本类型。

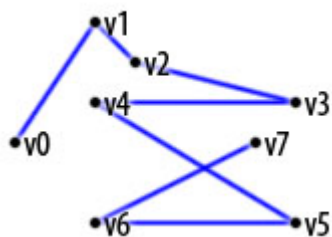
## GL\_POINTS

绘制独立的点。



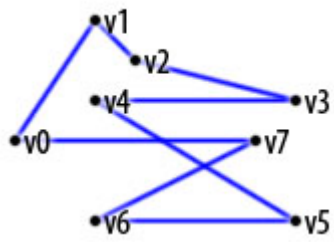
## GL\_LINE\_STRIP

绘制一系列线段。



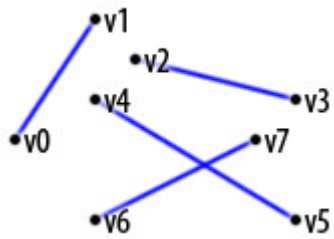
## GL\_LINE\_LOOP

类同上，但是首尾相连，构成一个封闭曲线。



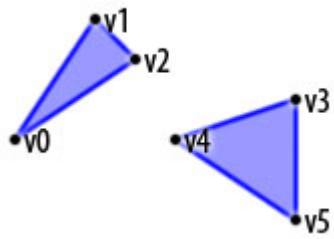
## GL\_LINES

顶点两两连接，为多条线段构成。



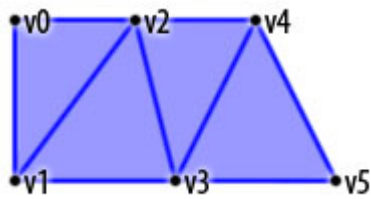
## GL\_TRIANGLES

每隔三个顶点构成一个三角形，为多个三角形组成。



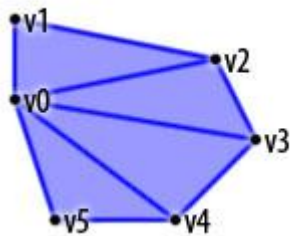
## GL\_TRIANGLE\_STRIP

每相邻三个顶点组成一个三角形，为一系列相接三角形构成。



## GL\_TRIANGLE\_FAN

以一个点为三角形公共顶点，组成一系列相邻的三角形。



下面可以来绘制正方形了，在项目中添加一个 `Square.java` 定义如下：

### [帮助](#)

```
1      package se.jayway.opengl.tutorial;

2

3      import java.nio.ByteBuffer;

4      import java.nio.ByteOrder;

5      import java.nio.FloatBuffer;

6      import java.nio.ShortBuffer;

7

8      import javax.microedition.khronos.opengles.GL10;

9
```

```
10     public class Square {
11
12         // Our vertices.
13
14         private float vertices[] = {
15
16             -1.0f,  1.0f, 0.0f,  // 0, Top Left
17
18             -1.0f, -1.0f, 0.0f,  // 1, Bottom Left
19
20             1.0f, -1.0f, 0.0f,  // 2, Bottom Right
21
22             1.0f,  1.0f, 0.0f,  // 3, Top Right
23
24         };
25
26         // The order we like to connect them.
27
28         private short[] indices = { 0, 1, 2, 0, 2, 3 };
29
30
31         // Our vertex buffer.
32
33         private FloatBuffer vertexBuffer;
34
35
36         // Our index buffer.
37
38         private ShortBuffer indexBuffer;
39
40
41         public Square() {
42
43             // a float is 4 bytes, therefore we
```



```
30         // multiply the number if
31
32         // vertices with 4.
33
34         ByteBuffer vbb
35
36         = ByteBuffer.allocateDirect(vertices.length * 4);
37
38         vbb.order(ByteOrder.nativeOrder());
39
40         vertexBuffer = vbb.asFloatBuffer();
41
42         vertexBuffer.put(vertices);
43
44         vertexBuffer.position(0);
45
46
47
48
49         // short is 2 bytes, therefore we multiply
50
51         //the number if
52
53         // vertices with 2.
54
55         ByteBuffer ibb
56
57         = ByteBuffer.allocateDirect(indices.length * 2);
58
59         ibb.order(ByteOrder.nativeOrder());
60
61         indexBuffer = ibb.asShortBuffer();
62
63         indexBuffer.put(indices);
64
65         indexBuffer.position(0);
66
67
68     }
69
```

```
50      /**
51      * This function draws our square on screen.
52      * @param gl
53      */
54      public void draw(GL10 gl) {
55          // Counter-clockwise winding.
56          gl.glFrontFace(GL10.GL_CCW);
57          // Enable face culling.
58          gl.glEnable(GL10.GL_CULL_FACE);
59          // What faces to remove with the face culling.
60          gl.glCullFace(GL10.GL_BACK);
61
62          // Enabled the vertices buffer for writing
63          //and to be used during
64          // rendering.
65          gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
66          // Specifies the location and data format of
67          //an array of vertex
68          // coordinates to use when rendering.
69          gl.glVertexPointer(3, GL10.GL_FLOAT, 0,
```

```

70         vertexBuffer);

71

72         gl.glDrawElements(GL10.GL_TRIANGLES, indices.length,

73         GL10.GL_UNSIGNED_SHORT, indexBuffer);

74

75         // Disable the vertices buffer.

76         gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);

77         // Disable face culling.

78         gl.glDisable(GL10.GL_CULL_FACE);

79     }

80

81 }
```

在 `OpenGLRenderer` 中添加 `Square` 成员变量并初始化:

[帮助](#)

```

1         // Initialize our square.

2         Square square = new Square();
```

并在 `public void onDrawFrame(GL10 gl)` 添加

[帮助](#)

```

1         // Draw our square.
```

```
2           square.draw(gl);
```

来绘制这个正方形,编译运行,什么也没显示,这是为什么呢?这是因为 **OpenGL ES** 从当前位置开始渲染,缺省坐标为(0,0,0),和 **View port** 的坐标一样,相当于把画面放在眼前,对应这种情况 **OpenGL** 不会渲染离 **view Port** 很近的画面,因此我们需要将画面向后退一点距离:

#### [帮助](#)

```
1           // Translates 4 units into the screen.

2           gl.glTranslatef(0, 0, -4);
```

在编译运行,这次倒是有显示了,当正方形迅速后移直至看不见,这是因为每次调用 **onDrawFrame** 时,每次都再向后移动 4 个单位,需要加上重置 **Matrix** 的代码。

#### [帮助](#)

```
1           // Replace the current matrix with the identity matrix

2           gl.glLoadIdentity();
```

最终 **onDrawFrame** 的代码如下:

#### [帮助](#)

```
1           public void onDrawFrame(GL10 gl) {

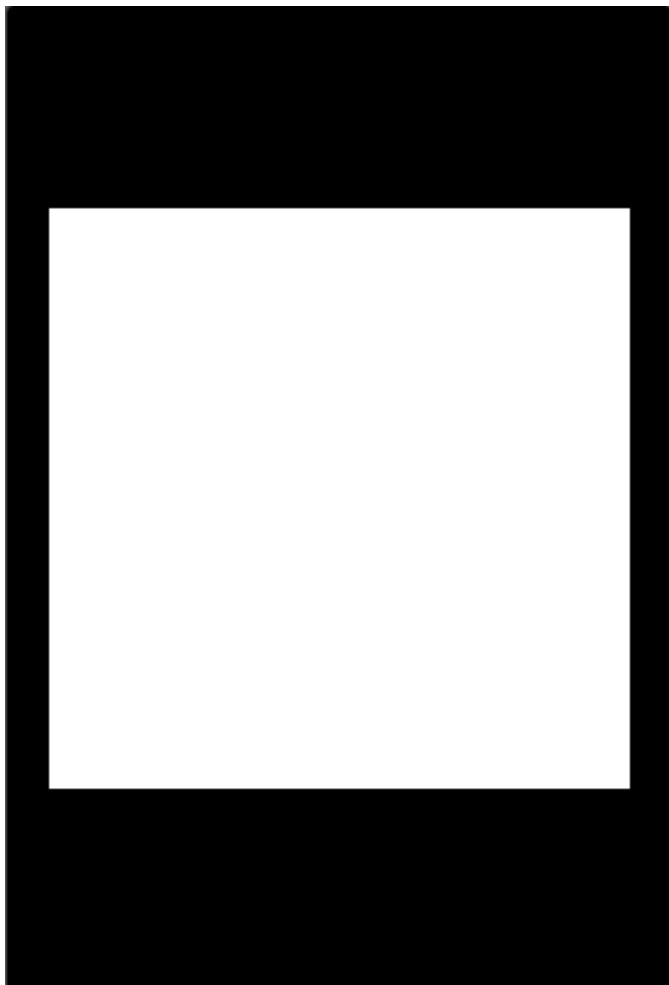
2           // Clears the screen and depth buffer.

3           gl.glClear(GL10.GL_COLOR_BUFFER_BIT |

4           GL10.GL_DEPTH_BUFFER_BIT);

5           gl.glLoadIdentity();
```

```
6         gl.glTranslatef(0, 0, -4);  
  
7         // Draw our square.  
  
8         square.draw(gl); // ( NEW )  
  
9  
  
10    }
```



本篇代码[下载](#)

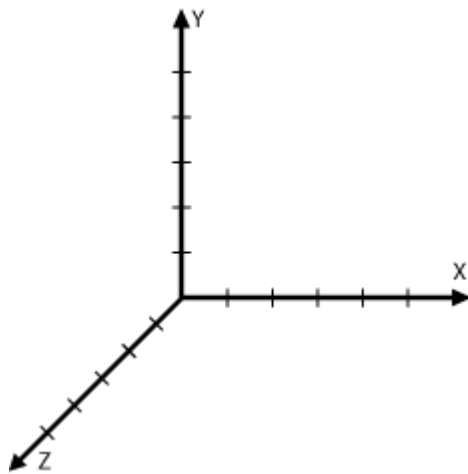
# Android OpenGL ES 简明开发教程四：3D 坐标变换

发表于 2011 年 05 月 31 日 由 [guidebee](#)

本篇介绍 3D 坐标系下的坐标变换 transformations。

## Coordinate System 坐标系

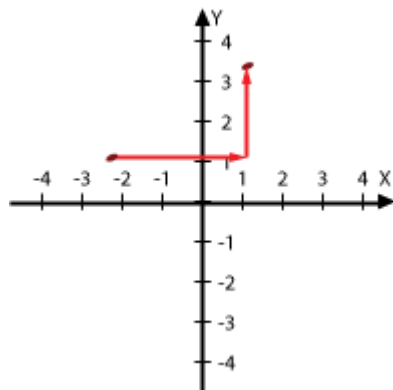
OpenGL 使用了右手坐标系，右手坐标系判断方法：在空间直角坐标系中，让右手拇指指向 x 轴的正方向，食指指向 y 轴的正方向，如果中指能指向 z 轴的正方向，则称这个坐标系为右手直角坐标系。



### Translate 平移变换

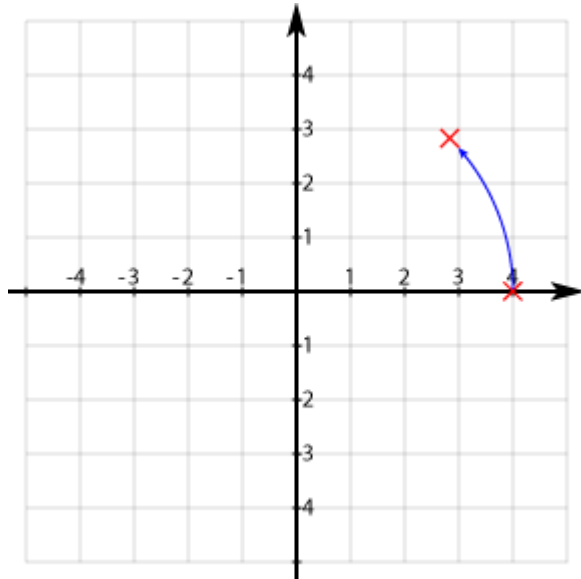
方法 `public abstract void glTranslatef (float x, float y, float z)` 用于坐标平移变换。

在上个例子中我们把需要显示的正方形后移了 4 个单位，就是使用的坐标的平移变换，可以进行多次平移变换，其结果为多个平移矩阵的累计结果，矩阵的顺序不重要，可以互换。



### Rotate 旋转

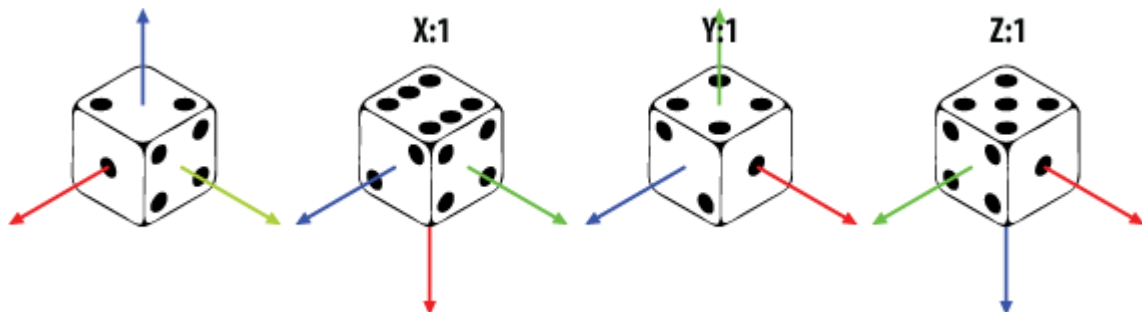
方法 `public abstract void glRotatef(float angle, float x, float y, float z)` 用来实现坐标变换，单位为角度。 $(x,y,z)$  定义旋转的参照矢量方向。多次旋转的顺序非常重要。



比如你选择一个骰子，首先按下列顺序选择 3 次：

#### [帮助](#)

```
1      gl.glRotatef(90f, 1.0f, 0.0f, 0.0f);  
  
2      gl.glRotatef(90f, 0.0f, 1.0f, 0.0f);  
  
3      gl.glRotatef(90f, 0.0f, 0.0f, 1.0f);
```



然后打算逆向旋转回原先的初始状态，需要有如下旋转：

#### [帮助](#)

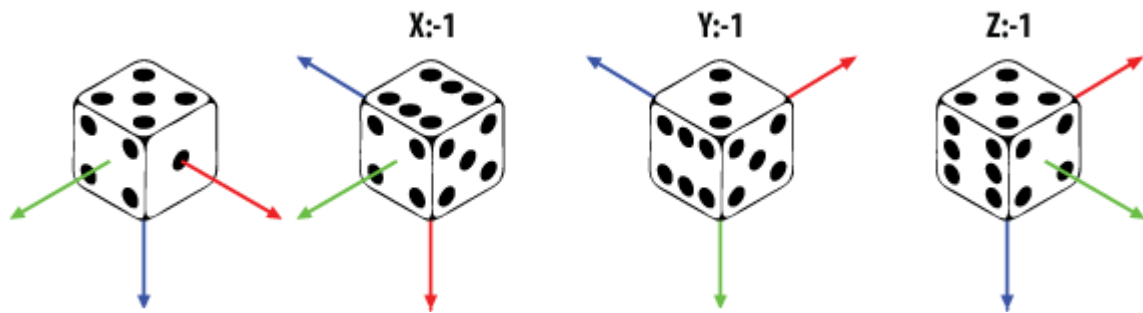
```

1      gl.glRotatef(90f, -1.0f, 0.0f, 0.0f);

2      gl.glRotatef(90f, 0.0f, -1.0f, 0.0f);

3      gl.glRotatef(90f, 0.0f, 0.0f, -1.0f);

```



或者如下旋转：

[帮助](#)

```

1      gl.glRotatef(90f, 0.0f, 0.0f, -1.0f);

2      gl.glRotatef(90f, 0.0f, -1.0f, 0.0f);

3      gl.glRotatef(90f, -1.0f, 0.0f, 0.0f);

```

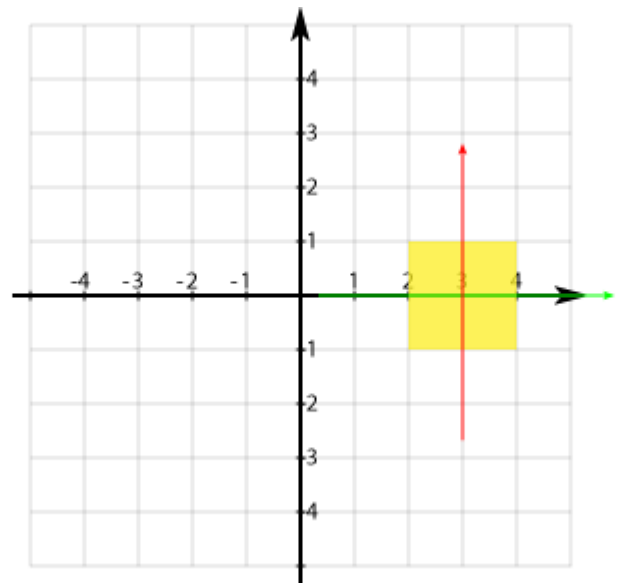
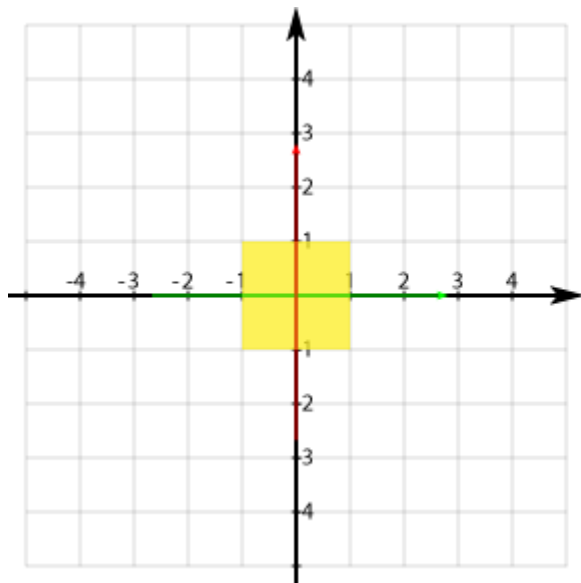
旋转变换 `glRotatef(angle, -x, -y, -z)` 和 `glRotatef(-angle, x, y, z)` 是等价的，但选择变换的顺序直接影响最终坐标变换的结果。 角度为正时表示逆时针方向。

## Translate & Rotate （平移和旋转组合变换）

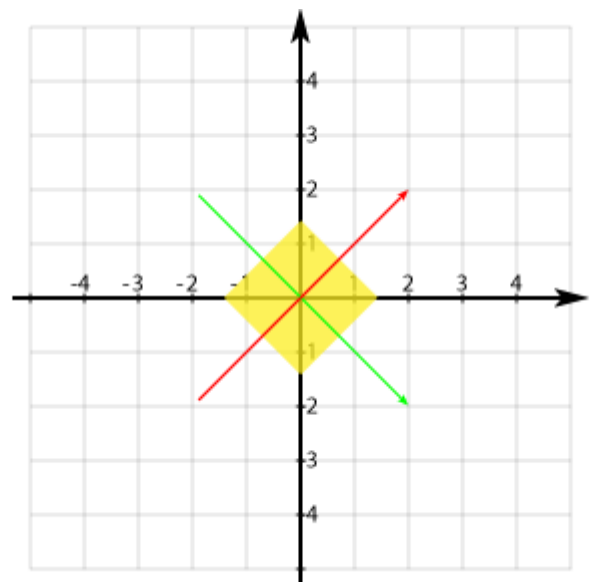
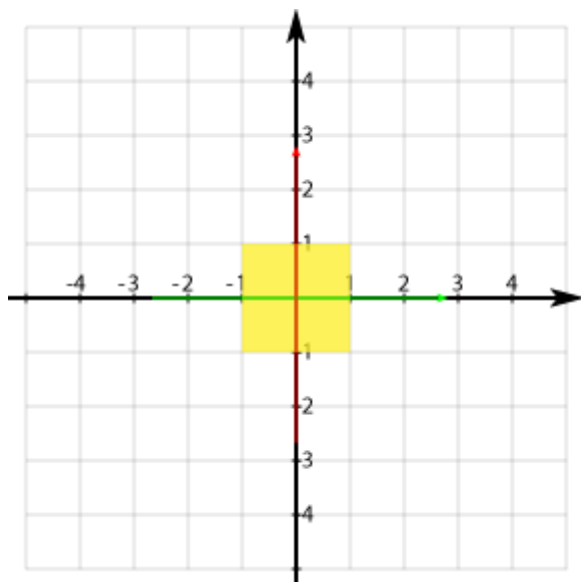
在对 **Mesh**（网格，构成三维形体的基本单位）同时进行平移和选择变换时，坐标变换的顺序也直接影响最终的结果。

比如：先平移后旋转， 旋转的中心为平移后的坐标。





先选择后平移： 平移在则相对于旋转后的坐标系：

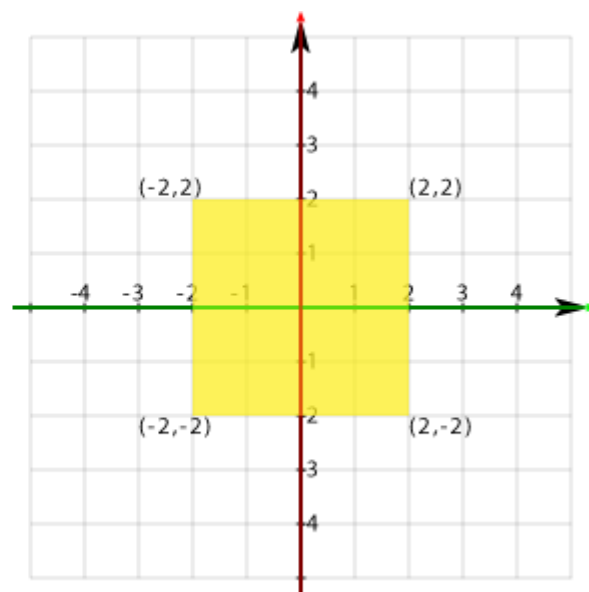
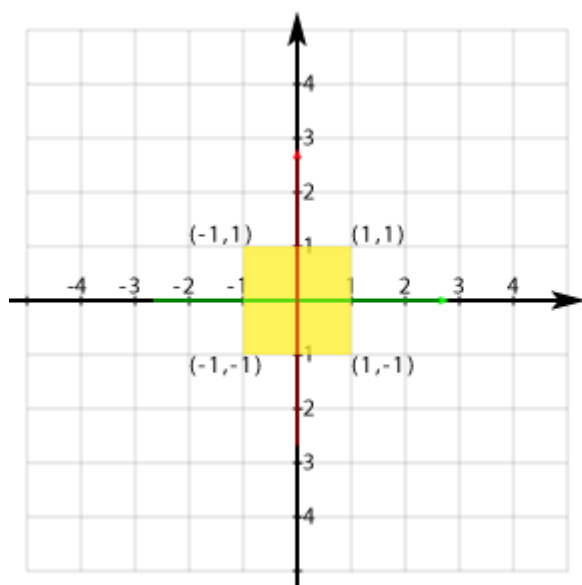


一个基本原则是，坐标变换都是相对于变换的 Mesh 本身的坐标系而进行的。

## Scale（缩放）

方法 `public abstract void glScalef (float x, float y, float z)` 用于缩放变换。

下图为使用 `gl.glScalef(2f, 2f, 2f)` 变换后的基本，相当于把每个坐标值都乘以 2。



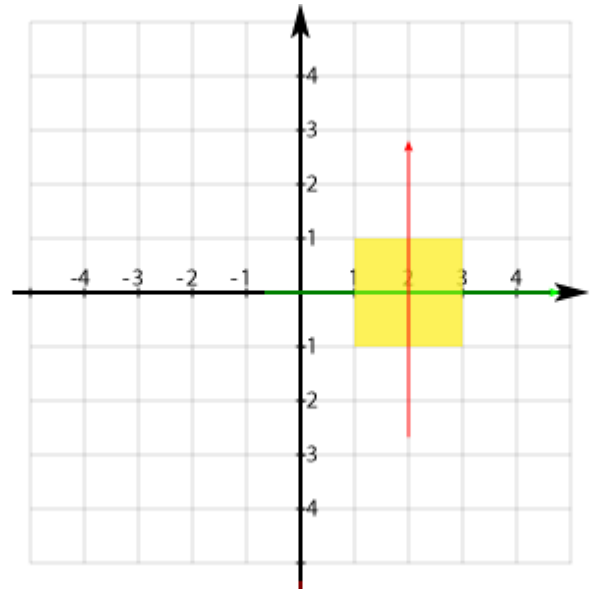
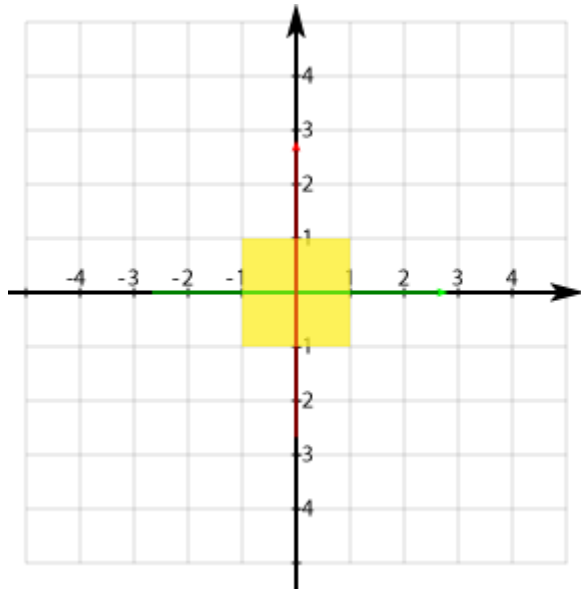
### Translate & Scale（平移和缩放组合变换）

同样当需要平移和缩放时，变换的顺序也会影响最终结果。

比如先平移后缩放：

[帮助](#)

```
1      glTranslatef(2, 0, 0);  
  
2      glScalef(0.5f, 0.5f, 0.5f);
```



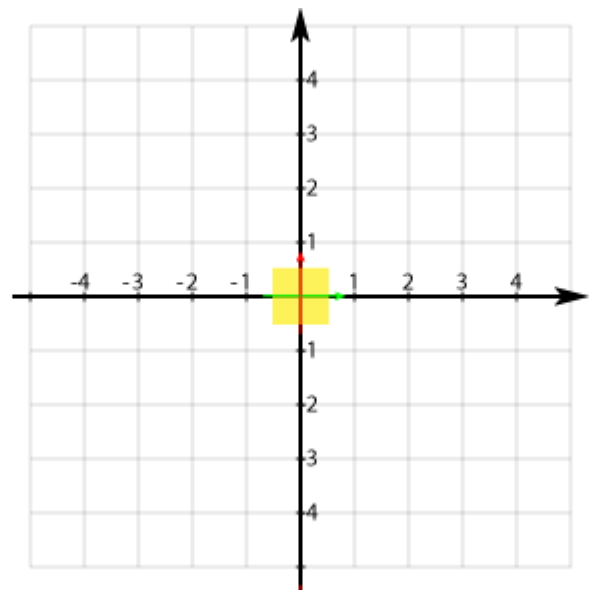
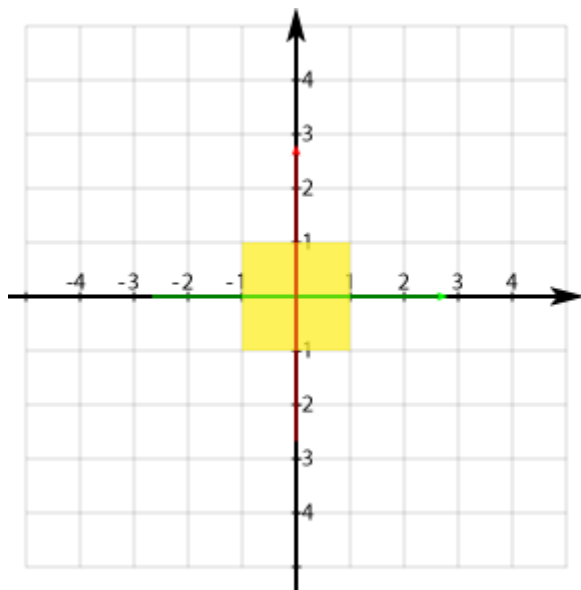
如果调换一下顺序：

[帮助](#)

```
1          gl.glScalef(0.5f, 0.5f, 0.5f);

2          gl.glTranslatef(2, 0, 0);
```

结果就有所不同：



矩阵操作，单位矩阵

在进行平移，旋转，缩放变换时，所有的变换都是针对当前的矩阵（与当前矩阵相乘），如果需要将当前矩阵回复最初的无变换的矩阵，可以使用单位矩阵（无平移，缩放，旋转）。

public abstract void [glLoadIdentity\(\)](#)。

在栈中保存当前矩阵和从栈中恢复所存矩阵，可以使用

public abstract void [glPushMatrix\(\)](#)

和

public abstract void [glPopMatrix\(\)](#)。

在进行坐标变换的一个好习惯是在变换前使用 `glPushMatrix` 保存当前矩阵，完成坐标变换操作后，再调用 `glPopMatrix` 恢复原先的矩阵设置。

最后利用上面介绍的坐标变换知识，来绘制 3 个正方形 A,B,C。进行缩放变换，使的 B 比 A 小 50%，C 比 B 小 50%。然后以屏幕中心逆时针旋转 A，B 以 A 为中心顺时针旋转，C 以 B 为中心顺时针旋转同时以自己中心高速逆时针旋转。

修改 `onDrawFrame` 代码如下：

#### [帮助](#)

```
1      public void onDrawFrame(GL10 gl) {  
2          // Clears the screen and depth buffer.  
3          gl.glClear(GL10.GL_COLOR_BUFFER_BIT  
4              | GL10.GL_DEPTH_BUFFER_BIT);  
5          // Replace the current matrix with the identity matrix  
6          gl.glLoadIdentity();  
7          // Translates 10 units into the screen.
```

```
8      gl.glTranslatef(0, 0, -10);

9

10     // SQUARE A

11     // Save the current matrix.

12     gl.glPushMatrix();

13     // Rotate square A counter-clockwise.

14     gl.glRotatef(angle, 0, 0, 1);

15     // Draw square A.

16     square.draw(gl);

17     // Restore the last matrix.

18     gl.glPopMatrix();

19

20     // SQUARE B

21     // Save the current matrix

22     gl.glPushMatrix();

23     // Rotate square B before moving it,

24     //making it rotate around A.

25     gl.glRotatef(-angle, 0, 0, 1);

26     // Move square B.

27     gl.glTranslatef(2, 0, 0);
```

```
28         // Scale it to 50% of square A

29         gl.glScalef(.5f, .5f, .5f);

30         // Draw square B.

31         square.draw(gl);

32

33         // SQUARE C

34         // Save the current matrix

35         gl.glPushMatrix();

36         // Make the rotation around B

37         gl.glRotatef(-angle, 0, 0, 1);

38         gl.glTranslatef(2, 0, 0);

39         // Scale it to 50% of square B

40         gl.glScalef(.5f, .5f, .5f);

41         // Rotate around it's own center.

42         gl.glRotatef(angle*10, 0, 0, 1);

43         // Draw square C.

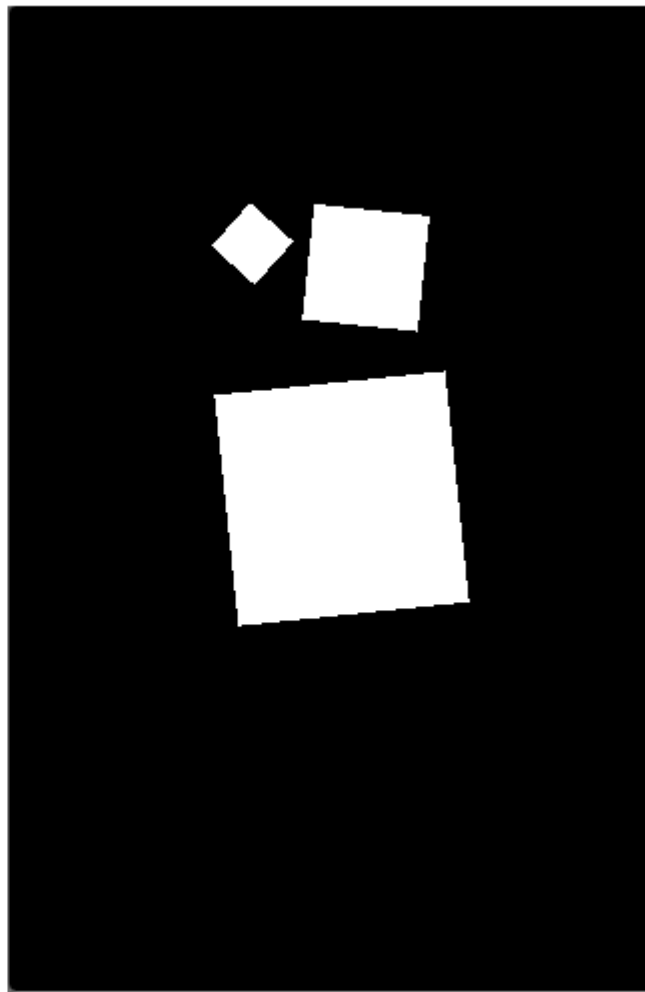
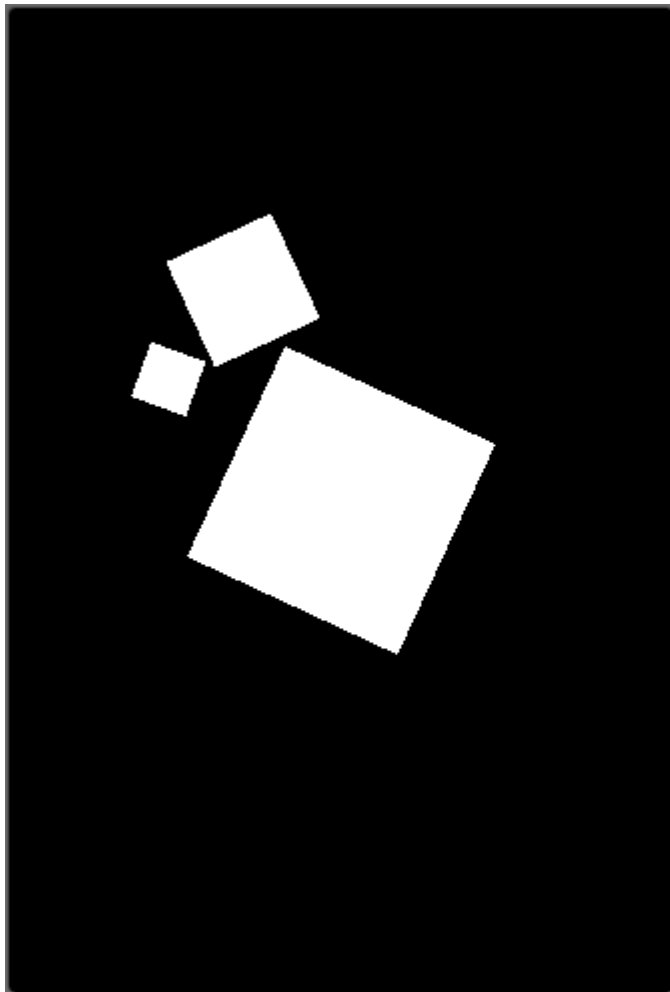
44         square.draw(gl);

45

46         // Restore to the matrix as it was before C.

47         gl.glPopMatrix();
```

```
48      // Restore to the matrix as it was before B.  
  
49      gl.glPopMatrix();  
  
50  
  
51      // Increse the angle.  
  
52      angle++;  
  
53  
  
54      }
```



本例代码[下载](#)

# Android OpenGL ES 简明开发教程五：添加颜色

发表于 2011 年 05 月 31 日 由 guidebee

前面的例子显示的正方形都是白色, 看来不是很吸引人, 本篇介绍如何给 Mesh (网格) 添加颜色。OpenGL ES 使用颜色是我们熟知的 RGBA 模式 (红, 绿, 蓝, 透明度)。

颜色的定义通常使用 Hex 格式 `0xFF00FF` 或十进制格式 `(255,0,255)`, 在 OpenGL 中却是使用 `0...1` 之间的浮点数表示。0 为 0, 1 相当于 255 (`0xFF`)。

最简单的上色方法叫做顶点着色(Vertex coloring), 可以使用单色, 也可以定义颜色渐变或者使用材质 (类同于二维图形中各种 Brush 类型)。

## Flat coloring (单色)

是通知 OpenGL 使用单一的颜色来渲染, OpenGL 将一直使用指定的颜色来渲染直到你指定其它的颜色。

指定颜色的方法为

`public abstract void glColor4f(float red, float green, float blue, float alpha)`。

缺省的 red, green, blue 为 1, 代表白色。这也是为什么前面显示的正方形都是白色的缘故。

我们创建一个新的类为 FlatColoredSquare, 作为 Square 的子类, 将它的 draw 重定义如下:

```
public void draw(GL10 gl) {  
  
    gl.glColor4f(0.5f, 0.5f, 1.0f, 1.0f);  
  
    super.draw(gl);  
  
}
```

将 OpenGLRenderer 的 square 的类型改为 FlatColoredSquare。



```
private FlatColoredSquare square=new FlatColoredSquare();
```

编译运行，正方形颜色变成了蓝色：

## Smooth coloring

（平滑颜色过渡）

当给每个顶点定义一个颜色时，OpenGL 自动为不同顶点颜色之间生成中间过渡颜色（渐变色）。

在项目中添加一个 **SmoothColoredSquare** 类，作为 **Square** 子类，为每个顶点定义一个颜色值。

```
// The colors mapped to the vertices.  
  
float[] colors = {  
  
    1f, 0f, 0f, 1f, // vertex 0 red
```

```

0f, 1f, 0f, 1f, // vertex 1 green

0f, 0f, 1f, 1f, // vertex 2 blue

1f, 0f, 1f, 1f, // vertex 3 magenta

};

```

颜色定义的顺序和顶点的顺序是一致的。为了提高性能，和顶点坐标一样，我们也把颜色数组放到 **Buffer** 中：

```

// float has 4 bytes, colors (RGBA) * 4 bytes

ByteBuffer cbb

    = ByteBuffer.allocateDirect(colors.length * 4);

cbb.order(ByteOrder.nativeOrder());

colorBuffer = cbb.asFloatBuffer();

colorBuffer.put(colors);

colorBuffer.position(0);

```

最后修改 **draw** 方法，如下：

```

public void draw(GL10 gl) {

    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);


    // Enable the color array buffer to be

    //used during rendering.

    gl.glEnableClientState(GL10.GL_COLOR_ARRAY);

    // Point out the where the color buffer is.

```

```
gl.glColorPointer(4, GL10.GL_FLOAT, 0, colorBuffer);

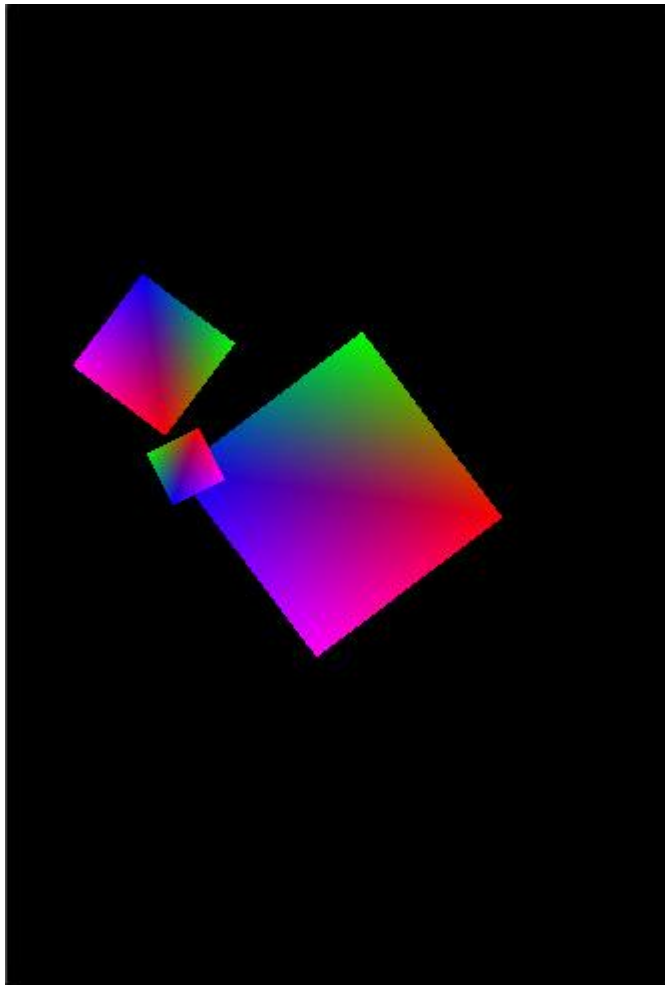
super.draw(gl);

// Disable the color buffer.

gl.glDisableClientState(GL10.GL_COLOR_ARRAY);

}
```

将 OpenGLRenderer 中的 Square 类型改成 SmoothColoredSquare，编译运行结果如下：



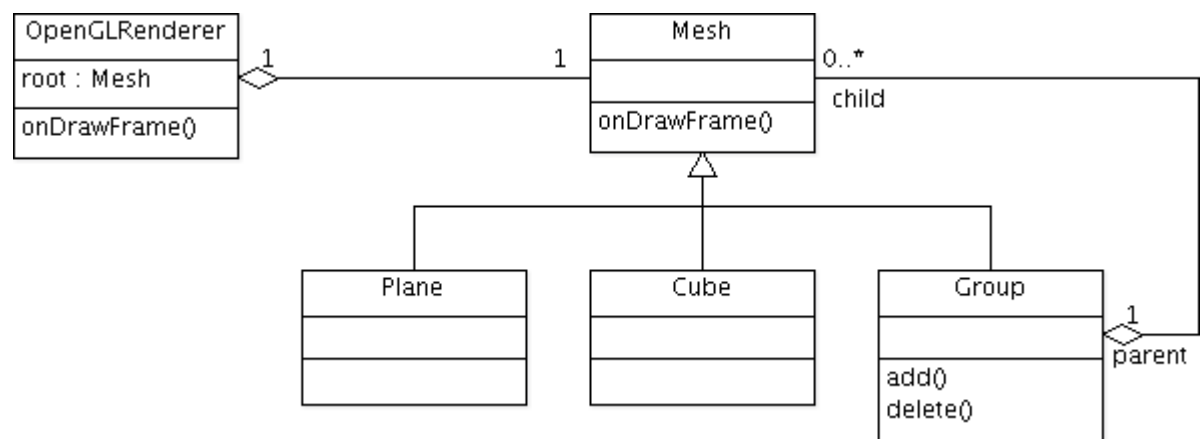
本地示例代码[下载](#)

## Android OpenGL ES 简明开发教程六：真正的 3D 图形

前面的例子尽管使用了 OpenGL ES 3D 图形库，但绘制的还是二维图形（平面上的正方形）。Mesh（网格，三角面）是构成空间形体的基本元素，前面的正方形也是有两个 Mesh 构成的。本篇将介绍使用 Mesh 构成四面体，椎体等基本空间形体。

### Design 设计

在使用 OpenGL 框架时一个好的设计原则是使用“[Composite Pattern](#)”，本篇采用如下设计：



### Mesh

首先定义一个基类 **Mesh**,所有空间形体最基本的构成元素为 **Mesh**（三角形网格），其基本定义如下：

#### [帮助](#)

```
1      public class Mesh {
2
3          // Our vertex buffer.
4
5          private FloatBuffer verticesBuffer = null;
6
7          // Our index buffer.
```

```
6         private ShortBuffer indicesBuffer = null;

7

8         // The number of indices.

9         private int numOfIndices = -1;

10

11        // Flat Color

12        private float[] rgba

13        = new float[] { 1.0f, 1.0f, 1.0f, 1.0f };

14

15        // Smooth Colors

16        private FloatBuffer colorBuffer = null;

17

18        // Translate params.

19        public float x = 0;

20

21        public float y = 0;

22

23        public float z = 0;

24

25        // Rotate params.
```

```
26         public float rx = 0;

27

28         public float ry = 0;

29

30         public float rz = 0;

31

32         public void draw(GL10 gl) {

33             // Counter-clockwise winding.

34             gl.glFrontFace(GL10.GL_CCW);

35             // Enable face culling.

36             gl.glEnable(GL10.GL_CULL_FACE);

37             // What faces to remove with the face culling.

38             gl.glCullFace(GL10.GL_BACK);

39             // Enabled the vertices buffer for writing and

40             //to be used during

41             // rendering.

42             gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);

43             // Specifies the location and data format

44             //of an array of vertex

45             // coordinates to use when rendering.
```

```
46     gl.glVertexPointer(3, GL10.GL_FLOAT, 0, verticesBuffer);

47     // Set flat color

48     gl.glColor4f(rgba[0], rgba[1], rgba[2], rgba[3]);

49     // Smooth color

50     if(colorBuffer != null) {

51         // Enable the color array buffer to be

52         //used during rendering.

53         gl.glEnableClientState(GL10.GL_COLOR_ARRAY);

54         gl.glColorPointer(4, GL10.GL_FLOAT, 0, colorBuffer);

55     }

56

57     gl.glTranslatef(x, y, z);

58     gl.glRotatef(rx, 1, 0, 0);

59     gl.glRotatef(ry, 0, 1, 0);

60     gl.glRotatef(rz, 0, 0, 1);

61

62     // Point out the where the color buffer is.

63     gl.glDrawElements(GL10.GL_TRIANGLES, numOfIndices,

64     GL10.GL_UNSIGNED_SHORT, indicesBuffer);

65     // Disable the vertices buffer.
```

```
66         gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);

67         // Disable face culling.

68         gl.glDisable(GL10.GL_CULL_FACE);

69     }

70

71     protected void setVertices(float[] vertices) {

72         // a float is 4 bytes, therefore

73         //we multiply the number if

74         // vertices with 4.

75         ByteBuffer vbb

76         = ByteBuffer.allocateDirect(vertices.length * 4);

77         vbb.order(ByteOrder.nativeOrder());

78         verticesBuffer = vbb.asFloatBuffer();

79         verticesBuffer.put(vertices);

80         verticesBuffer.position(0);

81     }

82

83     protected void setIndices(short[] indices) {

84         // short is 2 bytes, therefore we multiply

85         //the number if
```



```
86         // vertices with 2.

87         ByteBuffer ibb

88         = ByteBuffer.allocateDirect(indices.length * 2);

89         ibb.order(ByteOrder.nativeOrder());

90         indicesBuffer = ibb.asShortBuffer();

91         indicesBuffer.put(indices);

92         indicesBuffer.position(0);

93         numIndices = indices.length;

94     }

95

96     protected void setColor(float red, float green,

97         float blue, float alpha) {

98         // Setting the flat color.

99         rgba[0] = red;

100        rgba[1] = green;

101        rgba[2] = blue;

102        rgba[3] = alpha;

103    }

104

105    protected void setColors(float[] colors) {
```

```
106         // float has 4 bytes.

107         ByteBuffer cbb

108         = ByteBuffer.allocateDirect(colors.length * 4);

109         cbb.order(ByteOrder.nativeOrder());

110         colorBuffer = cbb.asFloatBuffer();

111         colorBuffer.put(colors);

112         colorBuffer.position(0);

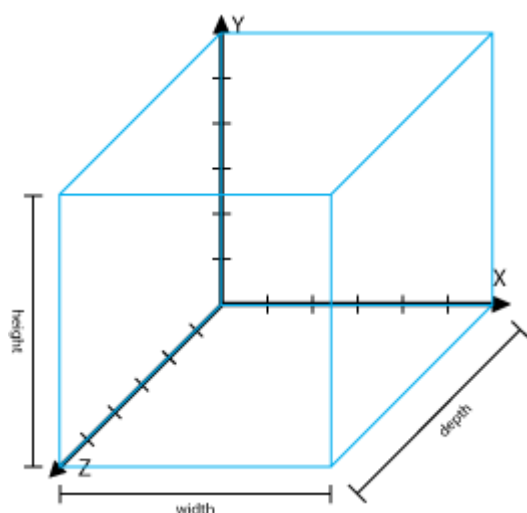
113     }

114 }
```

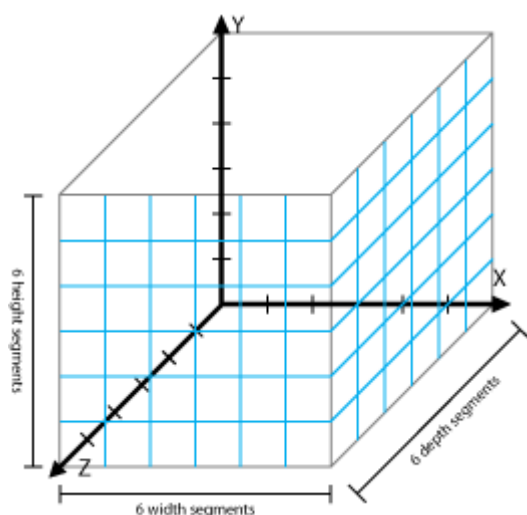
- **setVertices** 允许子类重新定义顶点坐标。
- **setIndices** 允许子类重新定义顶点的顺序。
- **setColor /setColors** 允许子类重新定义颜色。
- **x,y,z** 定义了平移变换的参数。
- **rx,ry,rz** 定义旋转变换的参数。

## Plane

有了 **Mesh** 定义之后，再来构造 **Plane**，**plane** 可以有宽度，高度和深度，宽度定义为沿 **X** 轴方向的长度，深度定义为沿 **Z** 轴方向长度，高度为 **Y** 轴方向。



Segments 为形体宽度，高度，深度可以分成的份数。Segments 在构造一个非均匀分布的 Surface 特别有用，比如在一个游戏场景中，构造地貌，使的 Z 轴的值随机分布在-0.1 到 0.1 之间，然后给它渲染好看的材质就可以造成地图凹凸不平的效果。



上面图形中 Segments 为一正方形，但在 OpenGL 中我们需要使用三角形，所有需要将 Segments 分成两个三角形。为 Plane 定义两个构造函数：

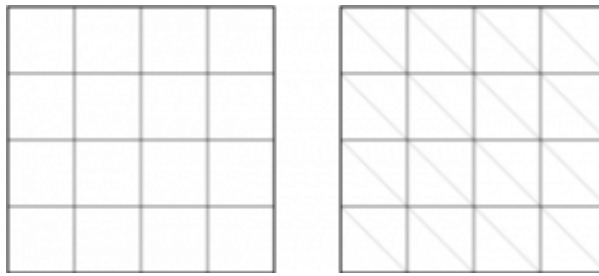
```
// Let you decide the size of the plane but still only one segment.
```

```
public Plane(float width, float height)
```

```
// For alla your settings.
```

```
public Plane(float width, float height, int widthSegments, int heightSegments)
```

比如构造一个 1 unit 宽和 1 unit 高，并分成 4 个 Segments，使用图形表示如下：



左边的图显示了 segments ,右边的图  
为需要创建的 Face（三角形）。

Plane 类的定义如下：

[帮助](#)

```
1      public class Plane extends Mesh {  
  
2          public Plane() {  
  
3              this(1, 1, 1, 1);  
  
4          }  
  
5  
6          public Plane(float width, float height) {  
  
7              this(width, height, 1, 1);  
  
8          }  
  
9  
10         public Plane(float width, float height, int widthSegments,  
11             int heightSegments) {  
  
12             float[] vertices  
  
13             = new float[(widthSegments + 1)  
14                 * (heightSegments + 1) * 3];
```

```
15     short[] indices
16     = new short[(widthSegments + 1)
17     * (heightSegments + 1)* 6];
18
19     float xOffset = width / -2;
20
21     float yOffset = height / -2;
22
23     float xWidth = width / (widthSegments);
24
25     float yHeight = height / (heightSegments);
26
27     int currentVertex = 0;
28
29     int currentIndex = 0;
30
31     short w = (short) (widthSegments + 1);
32
33     for (int y = 0; y < heightSegments + 1; y++) {
34
35         for (int x = 0; x < widthSegments + 1; x++) {
36
37             vertices[currentVertex] = xOffset + x * xWidth;
38
39             vertices[currentVertex + 1] = yOffset + y * yHeight;
40
41             vertices[currentVertex + 2] = 0;
42
43             currentVertex += 3;
44
45
46
47
48             int n = y * (widthSegments + 1) + x;
```

```
35     if (y < heightSegments && x < widthSegments) {  
36  
37         // Face one  
38  
39         indices[currentIndex] = (short) n;  
40  
41         indices[currentIndex + 1] = (short) (n + 1);  
42  
43         indices[currentIndex + 2] = (short) (n + w);  
44  
45         // Face two  
46  
47         indices[currentIndex + 3] = (short) (n + 1);  
48  
49         indices[currentIndex + 4] = (short) (n + 1 + w);  
50  
51         indices[currentIndex + 5] = (short) (n + 1 + w - 1);  
52  
53  
54         currentIndex += 6;  
55  
56     }  
57  
58 }  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

## Cube

下面来定义一个正方体（Cube），为简单起见，这个四面体只可以设置宽度，高度，和深度，没有和 **Plane** 一样提供 **Segments** 支持。

#### [帮助](#)

```
1      public class Cube extends Mesh {  
  
2          public Cube(float width, float height, float depth) {  
  
3              width /= 2;  
  
4              height /= 2;  
  
5              depth /= 2;  
  
6  
7              float vertices[] = { -width, -height, -depth, // 0  
8  
9              width, -height, -depth, // 1  
10  
11              width, height, -depth, // 2  
12  
13              -width, height, -depth, // 3  
14  
15              -width, -height, depth, // 4  
16  
17              width, -height, depth, // 5  
18  
19              width, height, depth, // 6  
20  
21              -width, height, depth, // 7  
22  
23          };  
  
24  
25          short indices[] = { 0, 4, 5,  
26  
27          0, 5, 1,
```

```
19         1, 5, 6,  
20         1, 6, 2,  
21         2, 6, 7,  
22         2, 7, 3,  
23         3, 7, 4,  
24         3, 4, 0,  
25         4, 7, 6,  
26         4, 6, 5,  
27         3, 0, 1,  
28         3, 1, 2, };  
29  
30         setIndices(indices);  
31         setVertices(vertices);  
32     }  
33 }
```

## Group

**Group** 可以用来管理多个空间几何形体，如果把 **Mesh** 比作 **Android** 的 **View**, **Group** 可以看作 **Android** 的 **ViewGroup**, **Android** 的 **View** 的设计也是采用的“**Composite Pattern**”。

**Group** 的主要功能是把针对 **Group** 的操作（如 **draw**）分发到 **Group** 中的每个成员对应的操作（如 **draw**）。



Group 定义如下:

[帮助](#)

```
1      public class Group extends Mesh {  
  
2          private Vector<Mesh> children = new Vector<Mesh>();  
  
3  
4          @Override  
  
5          public void draw(GL10 gl) {  
  
6              int size = children.size();  
  
7              for( int i = 0; i < size; i++)  
  
8                  children.get(i).draw(gl);  
  
9          }  
  
10  
11          /**  
12              * @param location  
13              * @param object  
14              * @see java.util.Vector#add(int, java.lang.Object)  
15              */  
16          public void add(int location, Mesh object) {  
  
17              children.add(location, object);  
  
18          }
```

```
19
20      /**
21      * @param object
22      * @return
23      * @see java.util.Vector#add(java.lang.Object)
24      */
25      public boolean add(Mesh object) {
26          return children.add(object);
27      }
28
29      /**
30      *
31      * @see java.util.Vector#clear()
32      */
33      public void clear() {
34          children.clear();
35      }
36
37      /**
38      * @param location
```

```
39      * @return
40
41      * @see java.util.Vector#get(int)
42
43      */
44
45      public Mesh get(int location) {
46
47          return children.get(location);
48
49      }
50
51
52
53
54
55
56
57
58      /**
59
60      * @param location
61
62      * @return
63
64      * @see java.util.Vector#remove(int)
65
66      */
67
68      public Mesh remove(int location) {
69
70          return children.remove(location);
71
72      }
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```

59      */

60      public boolean remove(Object object) {

61          return children.remove(object);

62      }

63

64      /**

65       * @return

66       * @see java.util.Vector#size()

67      */

68      public int size() {

69          return children.size();

70      }

71

72      }

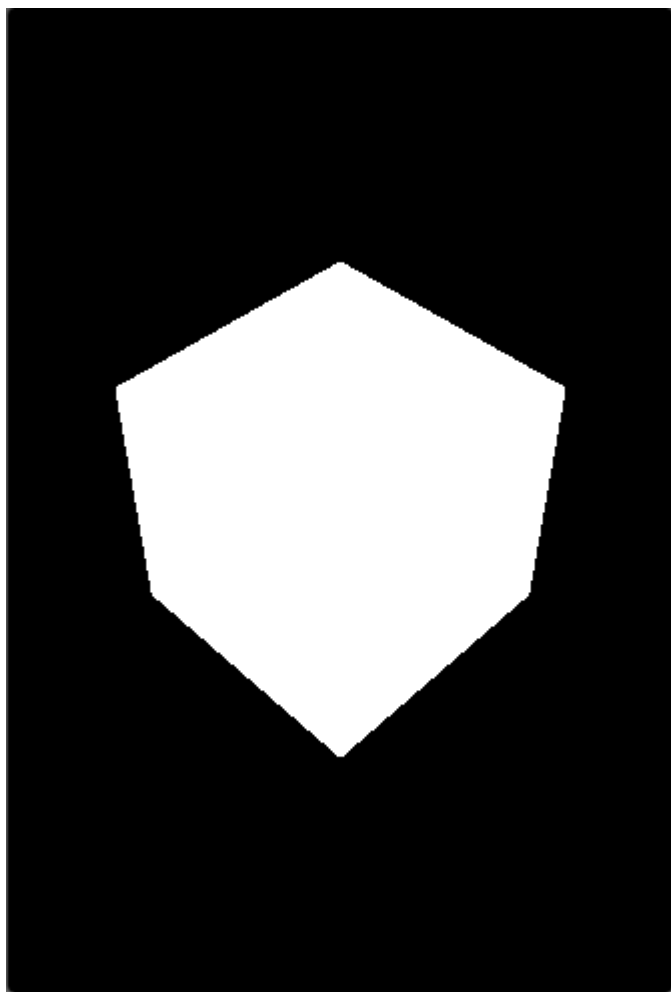
```

## 其它建议

上面我们定义里 **Mesh**, **Plane**, **Cube** 等基本空间几何形体,对于构造复杂图形(如人物),可以预先创建一些通用的几何形体,如果在组合成较复杂的形体。除了上面的基本形体外,可以创建如 **Cone**, **Pryamid**, **Cylinder** 等基本形体以备后用。



本例示例代码[下载](#), 显示结果如下:



## Android OpenGL ES 简明开发教程七：材质渲染

前面讨论了如何给 3D 图形染色，更一般的情况是使用位图来给 Mesh 上色（渲染材质）。主要步骤如下：

### 创建 Bitmap 对象

使用材质渲染，首先需要构造用来渲染的 Bitmap 对象，Bitmap 对象可以从资源文件中读取或是从网络下载或是使用代码构造。为简单起见，本例从资源中读取：

### [帮助](#)

```
1      Bitmap bitmap = BitmapFactory.decodeResource(context.getResources(),  
2      R.drawable.icon);
```

要注意的是，有些设备对使用的 **Bitmap** 的大小有要求，要求 **Bitmap** 的宽度和长度为 2 的几次幂（1, 2, 4, 8, 16, 32, 64。。。），如果使用不和要求的 **Bitmap** 来渲染，可能只会显示白色。

## 创建材质(Generating a texture)

下一步使用 OpenGL 库创建一个材质(Texture)，首先是获取一个 Texture Id。

### [帮助](#)

```
1      // Create an int array with the number of textures we want,  
2      // in this case 1.  
3      int[] textures = new int[1];  
4      // Tell OpenGL to generate textures.  
5      gl.glGenTextures(1, textures, 0);
```

textures 中存放了创建的 Texture ID，使用同样的 Texture Id，也可以来删除一个 Texture：

### [帮助](#)

```
1      // Delete a texture.  
2      gl.glDeleteTextures(1, textures, 0)
```

有了 Texture Id 之后，就可以通知 OpenGL 库使用这个 Texture：

### [帮助](#)

```
1      gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);
```

## 设置 Texture 参数 glTexParameter

下一步需要给 Texture 填充设置参数，用来渲染的 Texture 可能比要渲染的区域大或者小，这是需要设置 Texture 需要放大或是缩小时 OpenGL 的模式：

### [帮助](#)

```
1      // Scale up if the texture if smaller.

2      gl.glTexParameterf(GL10.GL_TEXTURE_2D,

3          GL10.GL_TEXTURE_MAG_FILTER,

4          GL10.GL_LINEAR);

5

6      // scale linearly when image smalled than texture

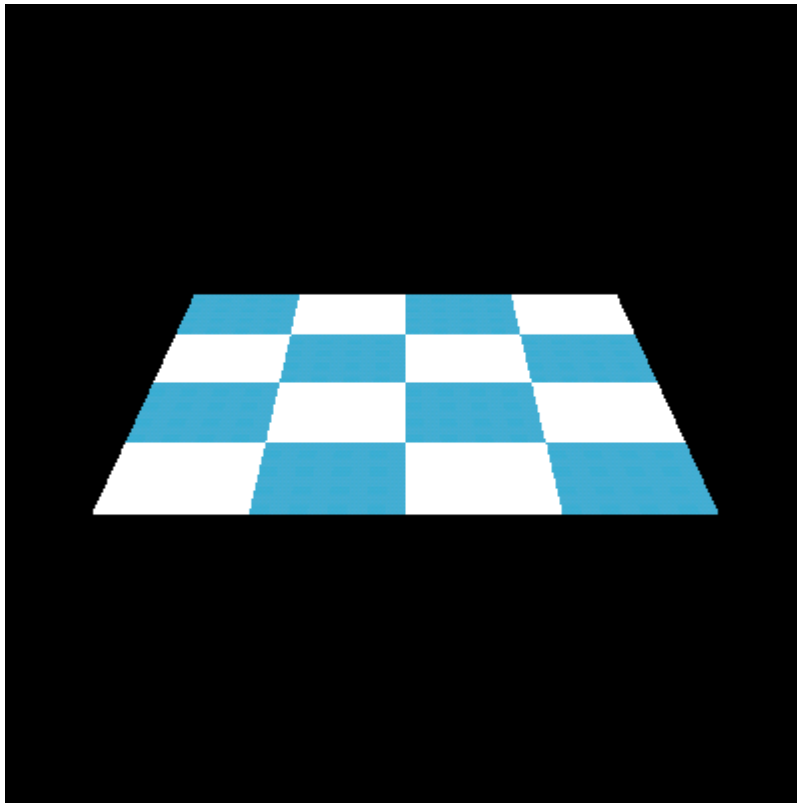
7      gl.glTexParameterf(GL10.GL_TEXTURE_2D,

8          GL10.GL_TEXTURE_MIN_FILTER,

9          GL10.GL_LINEAR);
```

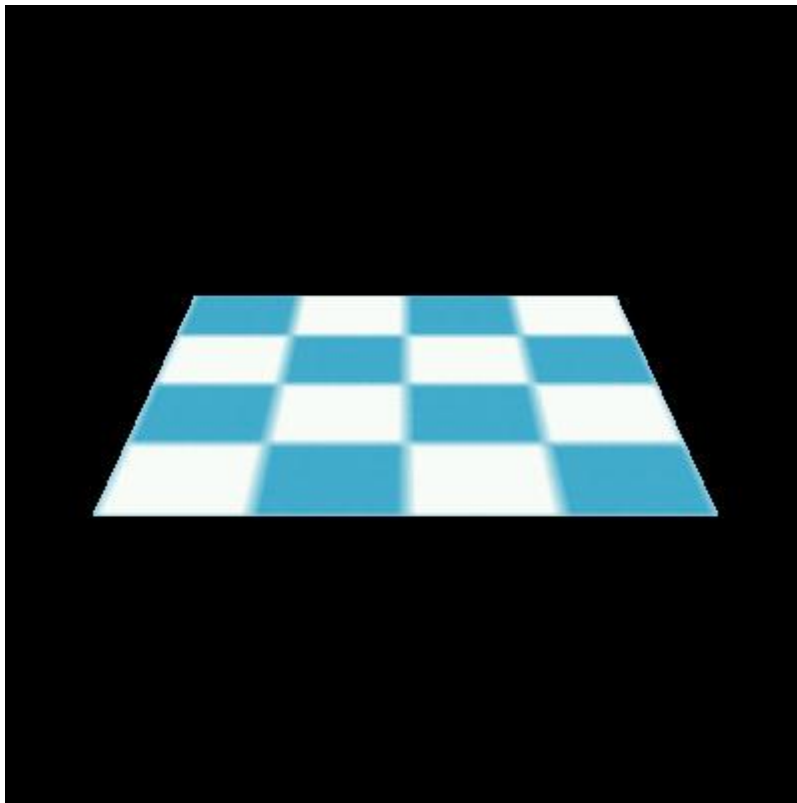
常用的两种模式为 GL10.GL\_LINEAR 和 GL10.GL\_NEAREST。

需要比较清晰的图像使用 GL10.GL\_NEAREST：



而使用

GL10.GL\_LINEAR 则会得到一个较模糊的图像：



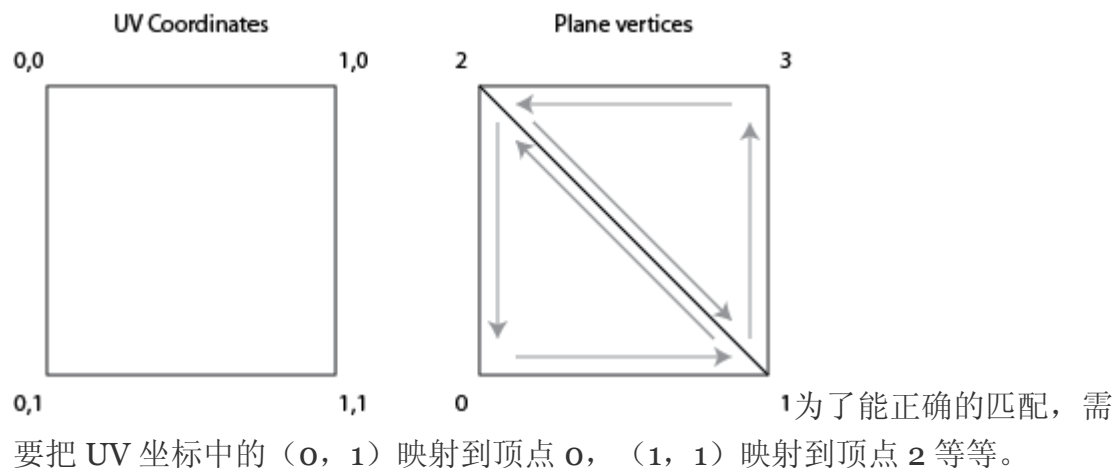
UV Mapping



下一步要告知 OpenGL 库如何将 Bitmap 的像素映射到 Mesh 上。这可以分为两步来完成：

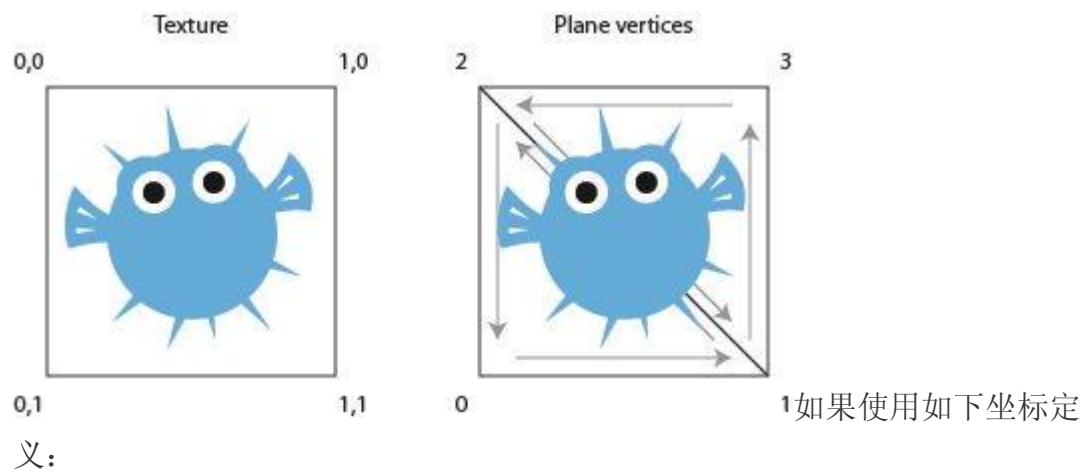
## 定义 UV 坐标

UV Mapping 指将 Bitmap 的像素映射到 Mesh 上的顶点。UV 坐标定义为左上角 (0, 0)，右下角 (1, 1) (因为使用的 2D Texture)，下图坐标显示了 UV 坐标，右边为我们需要染色的平面的顶点顺序：



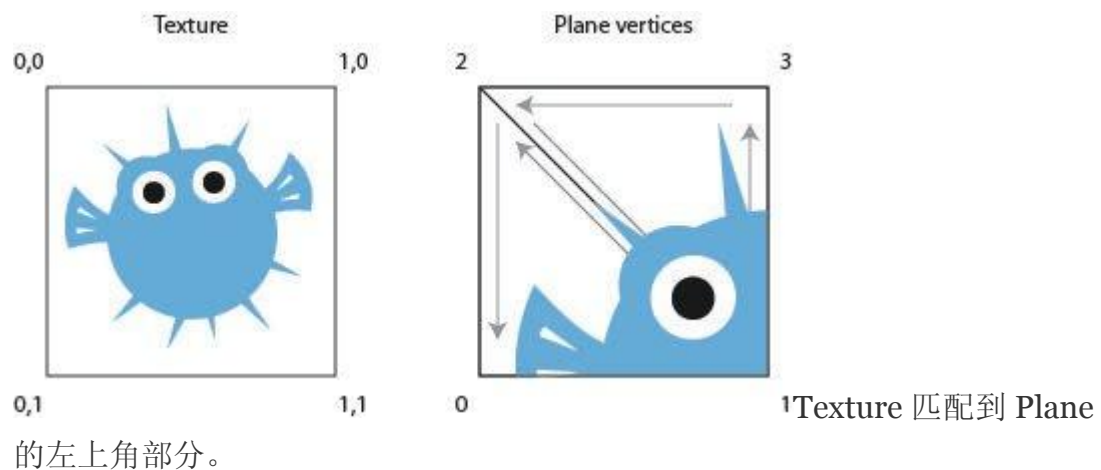
## [帮助](#)

```
1 float textureCoordinates[] = {0.0f, 1.0f,  
2 1.0f, 1.0f,  
3 0.0f, 0.0f,  
4 1.0f, 0.0f };
```



[帮助](#)

```
1 float textureCoordinates[] = {0.0f, 0.5f,
2     0.5f, 0.5f,
3     0.0f, 0.0f,
4     0.5f, 0.0f };
```



而

[帮助](#)

```
1 float textureCoordinates[] = {0.0f, 2.0f,
2     2.0f, 2.0f,
```

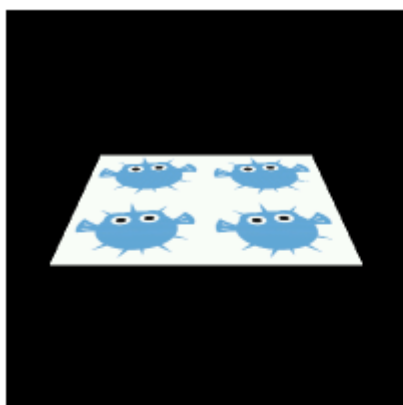
```
3          0.0f, 0.0f,  
4          2.0f, 0.0f };
```

将使用一些不存在的 Texture 去渲染平面(UV 坐标为 0, 0-1, 1 而 (0,0)-(2,2) 定义超过 UV 定义的大小)，这时需要告诉 OpenGL 库如何去渲染这些不存在的 Texture 部分。

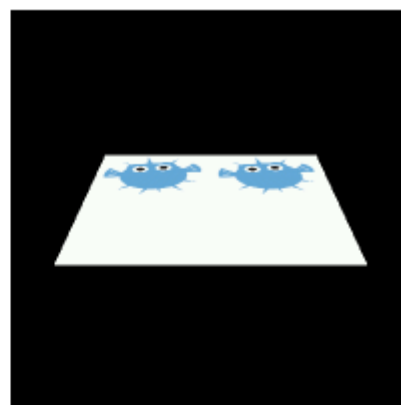
有两种设置

- **GL\_REPEAT** 重复 Texture。
- **GL\_CLAMP\_TO\_EDGE** 只靠边线绘制一次。

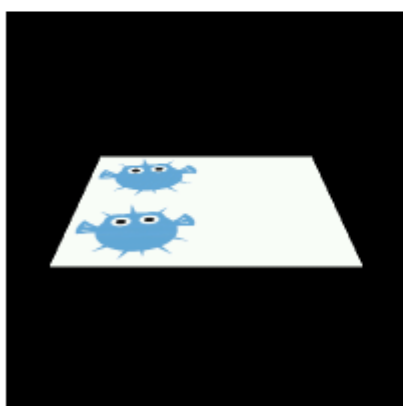
下面有四种不同组合：



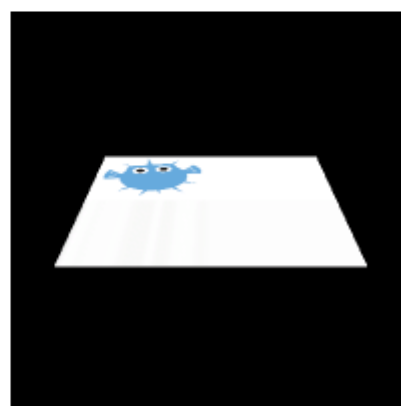
WRAP\_S: GL\_REPEAT  
WRAP\_T: GL\_REPEAT



WRAP\_S: GL\_REPEAT  
WRAP\_T: GL\_CLAMP\_TO\_EDGE



WRAP\_S: GL\_CLAMP\_TO\_EDGE  
WRAP\_T: GL\_REPEAT



WRAP\_S: GL\_CLAMP\_TO\_EDGE  
WRAP\_T: GL\_CLAMP\_TO\_EDGE

本例使用如下配置：

[帮助](#)

```
1      gl.glTexParameterf(GL10.GL_TEXTURE_2D,  
  
2          GL10.GL_TEXTURE_WRAP_S,  
  
3          GL10.GL_REPEAT);  
  
4      gl.glTexParameterf(GL10.GL_TEXTURE_2D,  
  
5          GL10.GL_TEXTURE_WRAP_T,  
  
6          GL10.GL_REPEAT);
```

然后将 **Bitmap** 资源和 **Texture** 绑定起来：

[帮助](#)

```
1      GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
```

## 使用 **Texture**

为了能够使用上面定义的 **Texture**，需要创建一 **Buffer** 来存储 UV 坐标：

[帮助](#)

```
1      FloatBuffer byteBuf = ByteBuffer.allocateDirect(texture.length * 4);  
  
2      byteBuf.order(ByteOrder.nativeOrder());  
  
3      textureBuffer = byteBuf.asFloatBuffer();  
  
4      textureBuffer.put(textureCoordinates);  
  
5      textureBuffer.position(0);
```

## 渲染

[帮助](#)

```

1      // Telling OpenGL to enable textures.

2      gl.glEnable(GL10.GL_TEXTURE_2D);

3      // Tell OpenGL where our texture is located.

4      gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);

5      // Tell OpenGL to enable the use of UV coordinates.

6      gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

7      // Telling OpenGL where our UV coordinates are.

8      gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, textureBuffer);

9

10     // ... here goes the rendering of the mesh ...

11

12     // Disable the use of UV coordinates.

13     gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

14     // Disable the use of textures.

15     gl.glDisable(GL10.GL_TEXTURE_2D);

```

本例代码是在一个平面上（**SimplePlane**）下使用 **Texture** 来渲染，首先是修改 **Mesh** 基类，使它能够支持定义 UV 坐标：

[帮助](#)

```

1      // Our UV texture buffer.

2      private FloatBuffer mTextureBuffer;

```

```

3

4      /**

5          * Set the texture coordinates.

6          *

7          * @param textureCoords

8          */

9      protected void setTextureCoordinates(float[] textureCoords) {

10         // float is 4 bytes, therefore we multiply the number if

11         // vertices with 4.

12         ByteBuffer byteBuf = ByteBuffer.allocateDirect(

13             textureCoords.length * 4);

14         byteBuf.order(ByteOrder.nativeOrder());

15         mTextureBuffer = byteBuf.asFloatBuffer();

16         mTextureBuffer.put(textureCoords);

17         mTextureBuffer.position(0);

18     }

```

并添加设置 **Bitmap** 和创建 **Texture** 的方法：

[帮助](#)

```

1      // Our texture id.

2      private int mTextureId = -1;

```

```
3

4    // The bitmap we want to load as a texture.

5    private Bitmap mBitmap;

6

7    /**

8     * Set the bitmap to load into a texture.

9     *

10    * @param bitmap

11    */

12    public void loadBitmap(Bitmap bitmap) {

13        this.mBitmap = bitmap;

14        mShouldLoadTexture = true;

15    }

16

17    /**

18     * Loads the texture.

19     *

20     * @param gl

21     */

22    private void loadGLTexture(GL10 gl) {
```

```
23     // Generate one texture pointer...

24     int[] textures = new int[1];

25     gl.glGenTextures(1, textures, 0);

26     mTextureId = textures[0];

27

28     // ...and bind it to our array

29     gl.glBindTexture(GL10.GL_TEXTURE_2D, mTextureId);

30

31     // Create Nearest Filtered Texture

32     gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER,

33     GL10.GL_LINEAR);

34     gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER,

35     GL10.GL_LINEAR);

36

37     // Different possible texture parameters, e.g. GL10.GL_CLAMP_TO_EDGE

38     gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,

39     GL10.GL_CLAMP_TO_EDGE);

40     gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,

41     GL10.GL_REPEAT);

42
```



```
43         // Use the Android GLUtils to specify a two-dimensional texture image

44         // from our bitmap

45         GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, mBitmap, 0);

46     }
```

最后修改 **draw** 方法来渲染材质：

[帮助](#)

```
1         // Indicates if we need to load the texture.

2         private boolean mShouldLoadTexture = false;

3

4         /**

5          * Render the mesh.

6          *

7          * @param gl

8          *         the OpenGL context to render to.

9          */

10        public void draw(GL10 gl) {

11            ...

12

13            // Smooth color

14            if (mColorBuffer != null) {
```

```
15         // Enable the color array buffer to be used during rendering.

16         gl.glEnableClientState(GL10.GL_COLOR_ARRAY);

17         gl.glColorPointer(4, GL10.GL_FLOAT, 0, mColorBuffer);

18     }

19

20     if (mShouldLoadTexture) {

21         loadGLTexture(gl);

22         mShouldLoadTexture = false;

23     }

24     if (mTextureId != -1 && mTextureBuffer != null) {

25         gl.glEnable(GL10.GL_TEXTURE_2D);

26         // Enable the texture state

27         gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

28

29         // Point to our buffers

30         gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mTextureBuffer);

31         gl.glBindTexture(GL10.GL_TEXTURE_2D, mTextureId);

32     }

33

34     gl.glTranslatef(x, y, z);
```

```

35
36     ...
37
38     // Point out the where the color buffer is.
39     gl.glDrawElements(GL10.GL_TRIANGLES, mNumOfIndices,
40     GL10.GL_UNSIGNED_SHORT, mIndicesBuffer);
41
42     ...
43
44     if (mTextureId != -1 && mTextureBuffer != null) {
45         gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
46     }
47
48     ...
49
50     }

```

本例使用的 **SimplePlane** 定义如下：

[帮助](#)

```

1     package se.jayway.opengl.tutorial.mesh;
2

```

```
3      /**
4
5      * SimplePlane is a setup class for Mesh that creates a plane mesh.
6
7      *
8      * @author Per-Erik Bergman (per-erik.bergman@jayway.com)
9
10     */
11
12     public class SimplePlane extends Mesh {
13
14         /**
15
16         * Create a plane with a default width and height of 1 unit.
17
18         */
19
20         public SimplePlane() {
21
22         this(1, 1);
23
24         }
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
23      *          the height of the plane.

24      */

25      public SimplePlane(float width, float height) {

26          // Mapping coordinates for the vertices

27          float textureCoordinates[] = { 0.0f, 2.0f, //

28          2.0f, 2.0f, //

29          0.0f, 0.0f, //

30          2.0f, 0.0f, //

31      };

32

33      short[] indices = new short[] { 0, 1, 2, 1, 3, 2 };

34

35      float[] vertices = new float[] { -0.5f, -0.5f, 0.0f,

36      0.5f, -0.5f, 0.0f,

37      -0.5f, 0.5f, 0.0f,

38      0.5f, 0.5f, 0.0f };

39

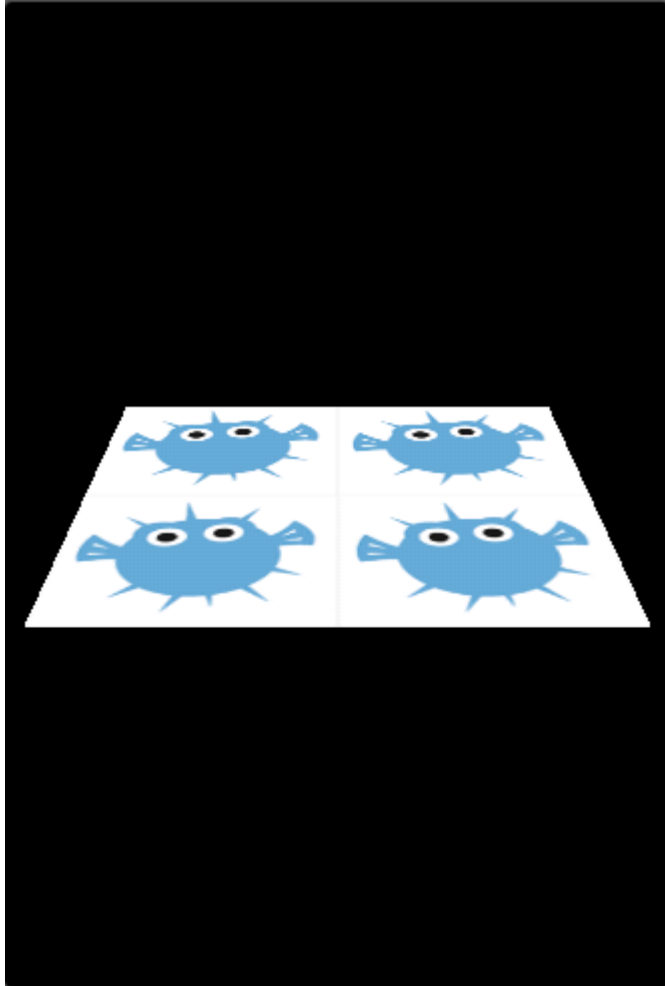
40      setIndices(indices);

41      setVertices(vertices);

42      setTextureCoordinates(textureCoordinates);
```

```
43     }
```

```
44 }
```



本例示例代码[下载](#)，到本篇为止介绍了 OpenGL ES 开发的基本方法，更详细的教程将在以后发布，后面先回到 Android ApiDemos 中 OpenGL ES 的示例。

。