河北师范大学软件学院
Software College of Hebei Normal University

Android源生AR应用开发

第二讲 OpenGL ES 基础

智能设备教研室

目录

OpenGL ES 简介

- OpenGL ES（OpenGL for Embedded System），是基于OpenGL三维图形API的子集，主要针对于手机以及PDA等嵌入式设备设计的。
- 随着Android系统版本以及硬件水平的提升，OpenGL ES版本也由原先仅支持固定渲染管线的OpenGL ES 1.X升级为支持自定义渲染管线的OpenGL ES 2.0。

OpenGL ES 简介

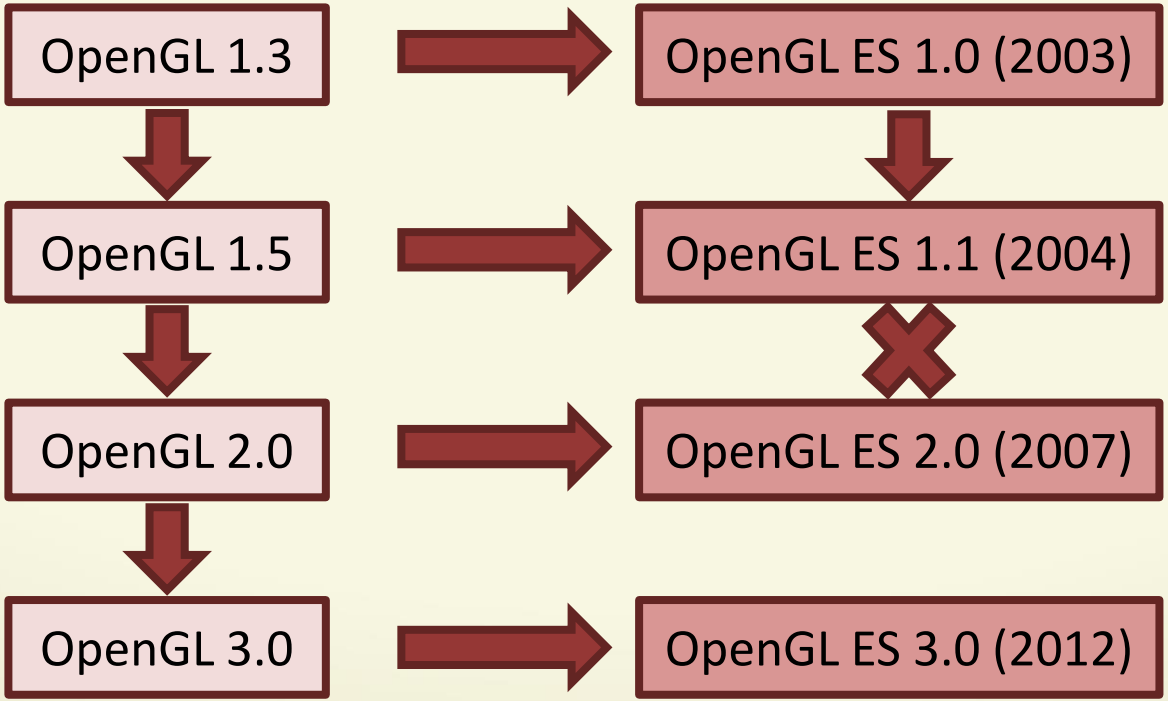
- 使用OpenGL ES 2.0渲染的3D场景更加真实从而能够创造全新的用户体验。
- 基于OpenGL ES 2.0的3D应用不能在模拟器上运行，必须使用配置了GPU（GPU要求支持OpenGL ES 2.0）的真机（Android版本要求最低为2.2）才可以。

OpenGL ES 简介

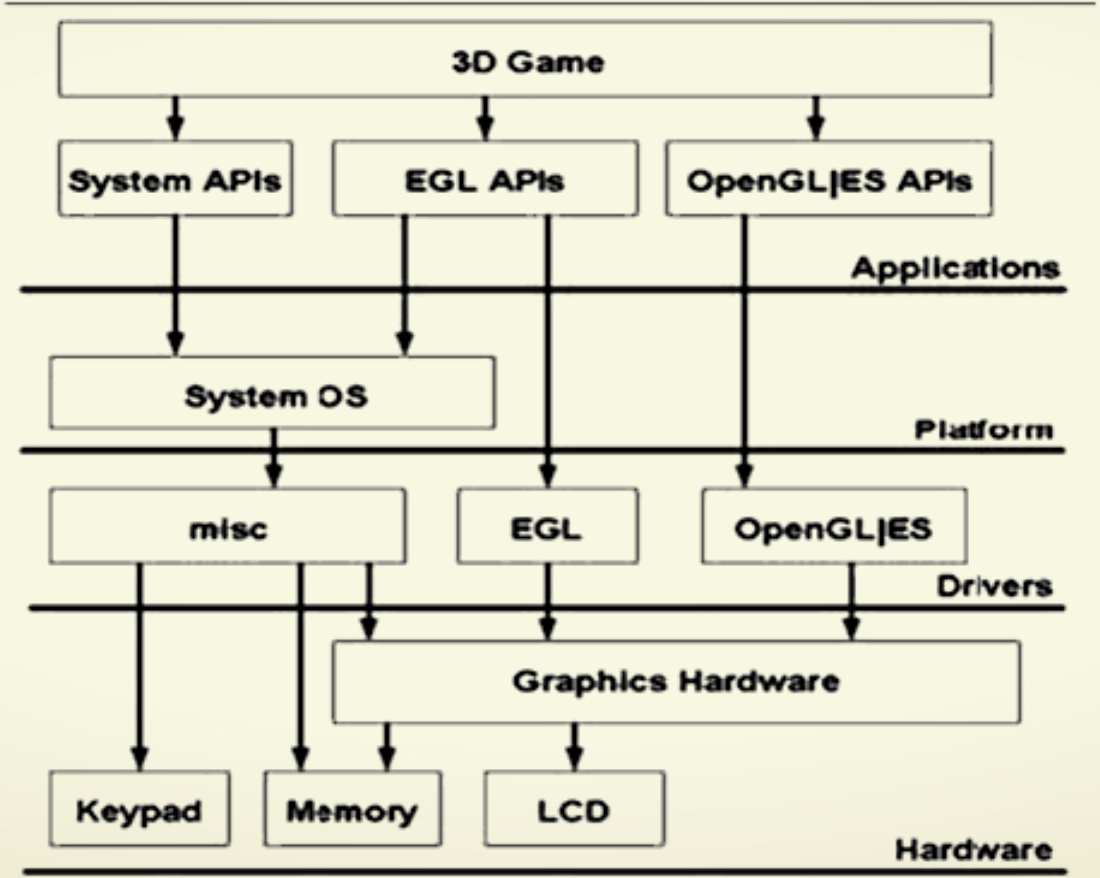
- 现今较为知名的3D图形API有OpenGL、DirectX以及OpenGL ES，它们各自的应用领域如下：
  - **DirectX**：主要应用与Windows下游戏开发。
  - **OpenGL**：应用领域比较广泛，适用于UNIX、Mac OS、Linux以及Microsoft等几乎所有操作系统。
  - **OpenGL ES**：专门针对于嵌入式设备的，其实是OpenGL的剪裁版本，去除了OpenGL中许多不是必须存在的特性。

OpenGL ES 简介

- OpenGL 及 OpenGL ES 历史版本信息：



OpenGL ES 简介


3D Game
System APIs
EGL APIs
OpenGL|ES APIs
Applications
System OS
Platform
misc
EGL
OpenGL|ES
Drivers
Graphics Hardware
Keypad
Memory
LCD
Hardware

OpenGL ES 简介

- Android 3D图形系统也分为Java框架和本地代码两部分:
  - 本地代码主要实现OpenGL接口库
  - 在Java框架层
    - javax.microedition.khronos.opengles是Java标准的OpenGL包
    - android.opengl提供了OpenGL系统和Android GUI系统之间的联系

目录

构造OpenGL ES View

- OpenGL ES API核心包类 – **GLSurfaceView**
  - 起到连接OpenGL ES与Android 的View层次结构之间的桥梁作用。
  - 使得Open GL 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。

构造OpenGL ES View

- 为了能在Android应用中使用OpenGLES绘画，必须创建一个View作为容器。而最直接的方式就是从GLSurfaceView和GLSurfaceView.Renderer分别派生一个类。
- GLSurfaceView作为OpenGL绘制所在的容器，而实际的绘图动作都是在GLSurfaceView.Renderer里面发生的。

构造OpenGL ES View - GLSurfaceView

- 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OpenGL View用到的GLSurfaceView.Renderer。

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

构造OpenGL ES View - GLSurfaceView

- 自定义GLSurfaceView示例：

```
class MyGLSurfaceView extends GLSurfaceView {

    public MyGLSurfaceView(Context context) {
        super(context);

        // 设置Renderer到GLSurfaceView
        setRenderer(new MyRenderer());
    }
}
```

不建议直接使用GLSurfaceView，一般情况下都会需要对其进行功能扩展，例如响应触摸事件。

构造OpenGL ES View - GLSurfaceView

- 当使用OpenGLES 2.0时，你必须在GLSurfaceView构造器中调用另外一个函数，它说明将要使用2.0版的API：

```
// 创建一个OpenGL ES 2.0 Context
setEGLContextClientVersion(2);
```

构造OpenGL ES View - GLSurfaceView

- 另一个可以添加到GLSurfaceView实现的可选的操作是设置Render模式为只在绘制数据发生改变时才绘制View。使用 GLSurfaceView.RENDERMODE_WHEN_DIRTY：

```
// 只有在绘制数据改变时才绘制view
setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
```

此设置会阻止绘制GLSurfaceView的帧，直到你调用了requestRender()，这样会非常高效。

构造OpenGL ES View - 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：

```
public void onSurfaceCreated(GL10 gl, EGLConfig config)
// 在窗口创建的过程中，会调用该方法，因此，可以在该函数中进行
// 一些初始化工作。比如，设置背景色等
public void onDrawFrame(GL10 gl)
// 对当前View进行实际绘图操作
public void onSurfaceChanged(GL10 gl,
                             int width, int height)
// 在窗口发生变化时调用的函数，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 - 纵向切换时，此时可以重新设置绘制的纵
// 横比率
```

构造OpenGL ES View - GLSurfaceView.Renderer

- onSurfaceCreated()
  - 仅调用一次，用于设置View的OpenGLES环境。
- onDrawFrame()
  - 每次View被重绘时被调用。
- onSurfaceChanged()
  - 如果view的几和形状发生变化了就调用，例如当竖屏变为横屏时。

构造OpenGL ES View - GLSurfaceView.Renderer

❖自定义Renderer示例：

```
public class MyRenderer implements GLSurfaceView.Renderer{
  public void onSurfaceCreated(GL10 unused,
                               EGLConfig config)
  {  GLES20.glClearColor(0.5f, 0.5f, 0.5f, 1.0f); }

  public void onDrawFrame(GL10 unused)
  {  GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT); }

  public void onSurfaceChanged(GL10 unused,
                               int width,int height)
  {  GLES20.glViewport(0, 0, width, height); }
}
```

设置背景的颜色

重绘背景色

构造OpenGL ES View - GLSurfaceView.Renderer

- 为什么这些方法们都具有一个 GL10 类型的参数，但是使用的却是OpengGL ES 2.0 API？

  其实这是为了使Android框架能简单的兼容各OpenGLES版本。

目录

OpenGL ES 管线(Pipeline)

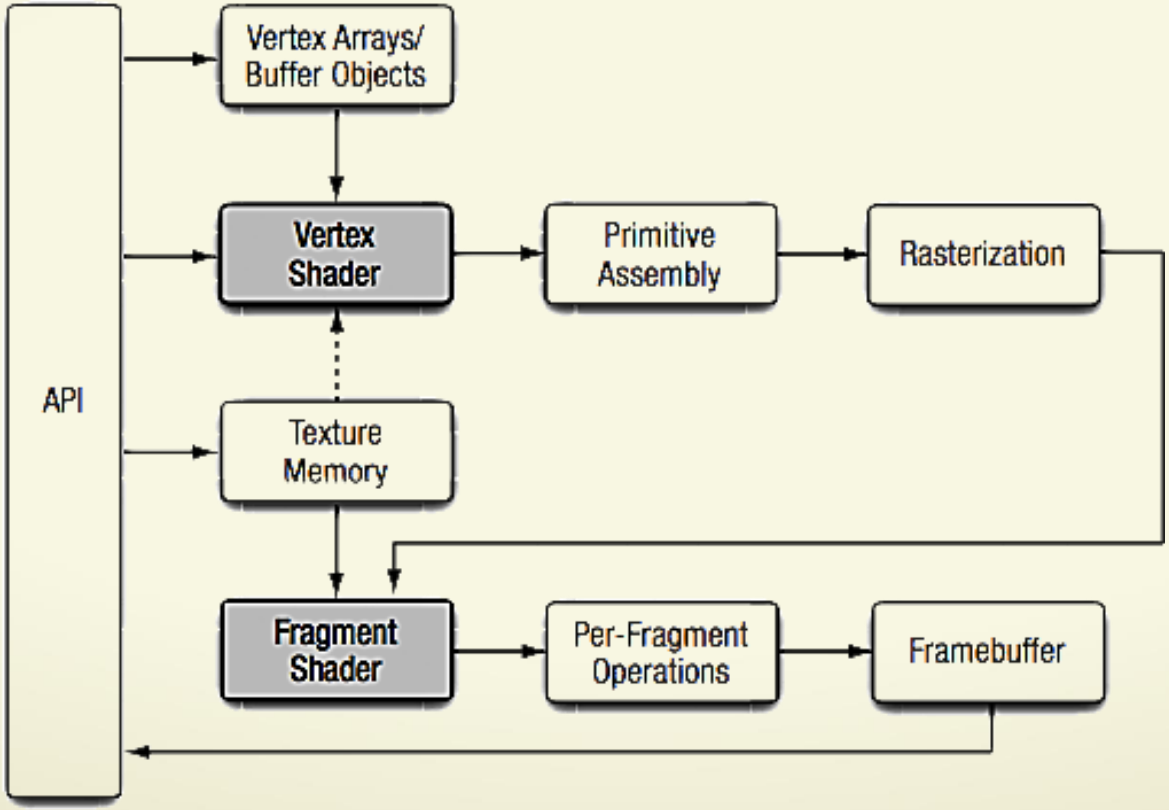
- 大部分图形系统都可以比作工厂中的装配线(Assemble line)或者称为管线(Pipeline)。前一道的输出作为下道工序的输入。主CPU发出一个绘图指令，然后可能由硬件部件完成坐标变换，裁剪，添加颜色或是材质，最后在屏幕上显示出来。

OpenGL ES 管线(Pipeline)

- OpenGL ES 1.x 的工序是固定的，称为Fix-Function Pipeline，可以想象一个带有很多控制开关的机器，尽管加工的工序是固定的，但是可以通过打开或关闭开关来设置参数或者打开关闭某些功能。
- OpenGL ES 2.0 允许提供编程来控制一些重要的工序，一些“繁琐”的工序比如栅格化等仍然是固定的。

OpenGL ES 管线(Pipeline)

❖OpenGL ES 2.0 的渲染管线



OpenGL ES 管线(Pipeline)

- **Vertex Array/Buffer Objects**：顶点数据来源，这时渲染管线的顶点输入，通常使用 Buffer Objects效率更好。

- **Vertex Shader**：顶点着色器通过可编程的方式实现对顶点的操作，如进行坐标空间转换，计算 per-vertex color以及纹理坐标；

OpenGL ES 管线(Pipeline)

- **Primitive Assembly**：图元装配，经过着色器处理之后的顶点在图片装配阶段被装配为基本图元。OpenGL ES 支持三种基本图元：点，线和三角形，它们是可被 OpenGL ES 渲染的。接着对装配好的图元进行裁剪（clip）：保留完全在视锥体中的图元，丢弃完全不在视锥体中的图元，对一半在一半不在的图元进行裁剪；接着再对在视锥体中的图元进行剔除处理（cull）：这个过程可编码来决定是剔除正面，背面还是全部剔除。

OpenGL ES 管线(Pipeline)

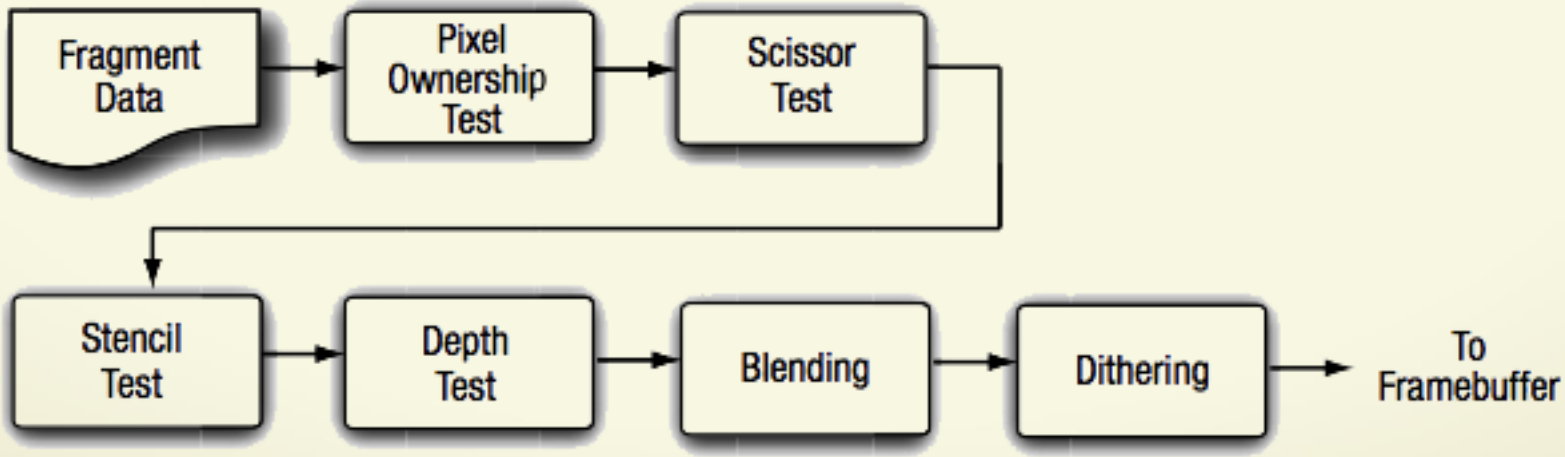
- **Rasterization**：光栅化。在光栅化阶段，基本图元被转换为二维的片元(fragment)，fragment 表示可以被渲染到屏幕上的像素，它包含位置，颜色，纹理坐标等信息，这些值是由图元的顶点信息进行插值计算得到的。这些片元接着被送到片元着色器中处理。这是从顶点数据到可渲染在显示设备上的像素的质变过程。

OpenGL ES 管线(Pipeline)

- **Fragment Shader**：片元着色器通过可编程的方式实现对片元的操作。在这一阶段它接受光栅化处理之后的fragment，color，深度值，模版值作为输入。

OpenGL ES 管线(Pipeline)

- **Per-Fragment Operation**：在这一阶段对片元着色器输出的每一个片元进行一系列测试与处理，从而决定最终用于渲染的像素。这一系列处理过程如下：



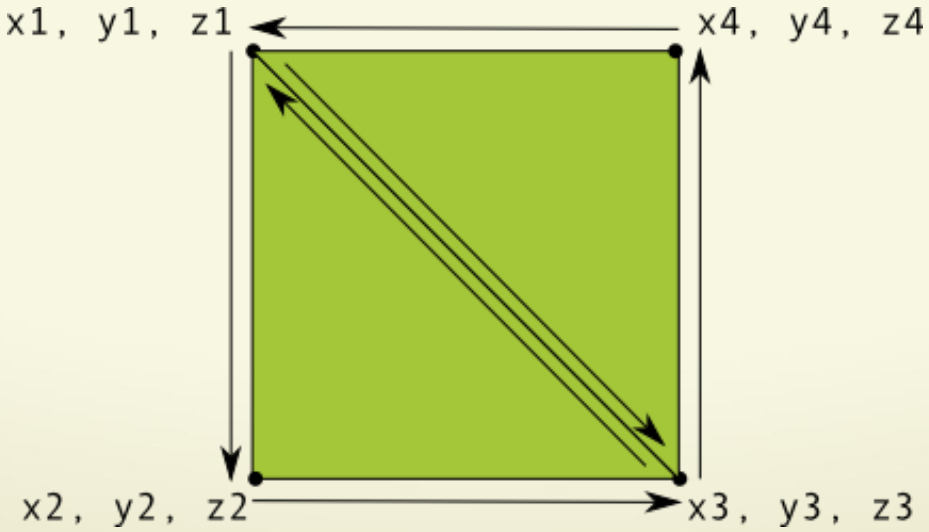
OpenGL ES 管线(Pipeline)

- **Framebuffer**：这是流水线的最后一个阶段，Framebuffer 中存储这可以用于渲染到屏幕或纹理中的像素值，也可以从Framebuffer 中读回像素值，但不能读取其它值（如深度值，模版值等）。

目录

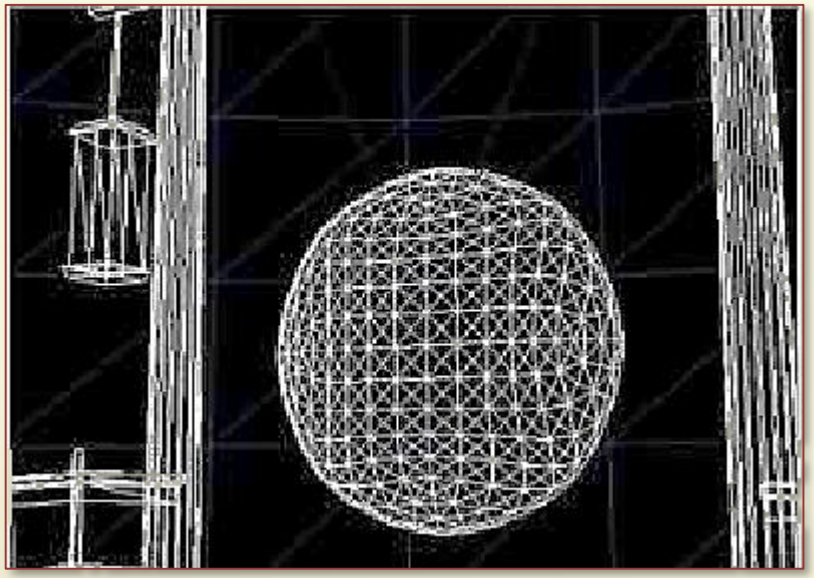
图形图像绘制

- 通常二维图形库可以绘制点，线，多边形，圆弧，路径等等。OpenGL ES 支持绘制的基本几何图形分为三类：点，线段，三角形。也就是说OpenGL ES 只能绘制这三种基本几何图形。任何复杂的2D或是3D图形都是通过这三种几何图形构造而成的。



图形图像绘制

- 比如下图复杂的3D图形，都有将其分割成细小的三角形面而构成的。然后通过上色(Color)，添加材质(Texture)，再添加光照（Lighting)，构造3D效果的图形：

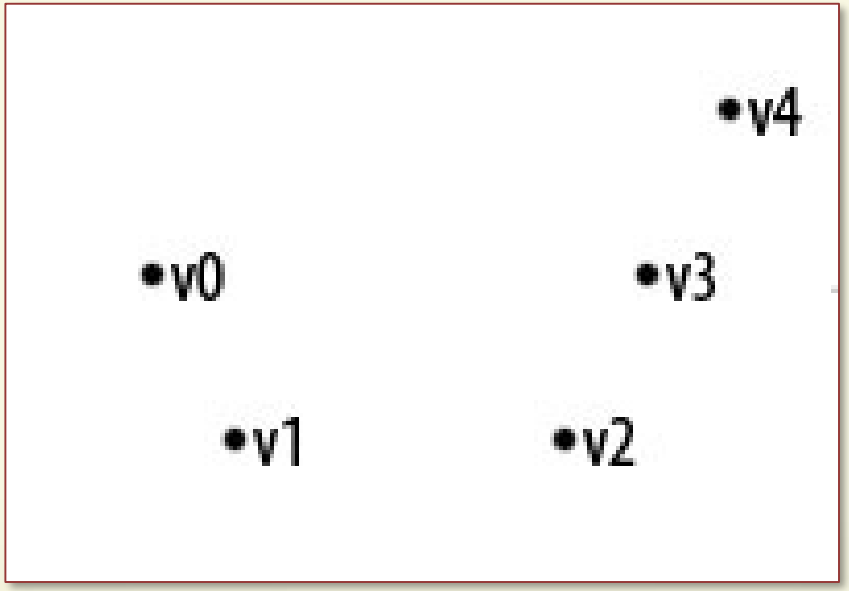


图形图像绘制

- 点，线段，三角形都是通过顶点来定义的，也就是顶点数组来定义。对应平面上的一系列顶点，可以看出一个个孤立的点(Point)，也可以两个两个连接成线段(Line Segment)，也可以三个三个连成三角形(Triangle)。这些对一组顶点的不同解释就定义了Android OpenGL ES可以绘制的基本几何图形，下面定义了OpenGL ES定义的几种模式：
  - GL_POINTS
  - GL_LINE_STRIP
  - GL_LINE_LOOP
  - GL_LINES
  - GL_TRIANGLES
  - GL_TRIANGLE_STRIP
  - GL_TRIANGLE_FAN

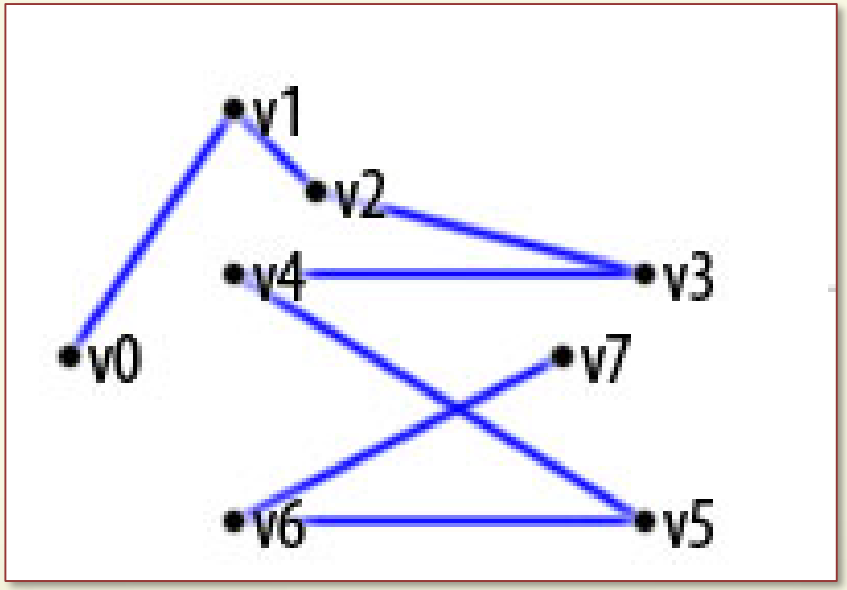
图形图像绘制

- GL_POINTS - 绘制独立的点



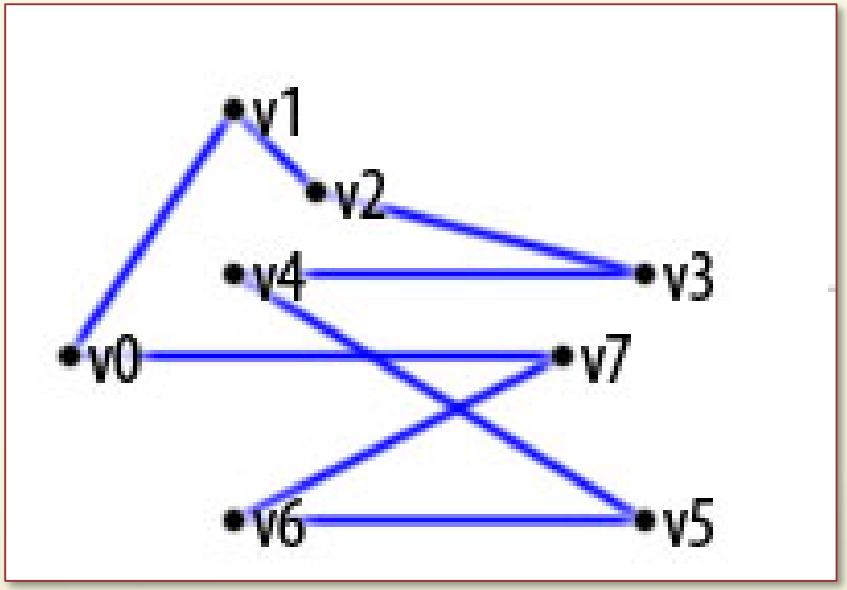
图形图像绘制

- GL_LINE_STRIP - 绘制一系列线段



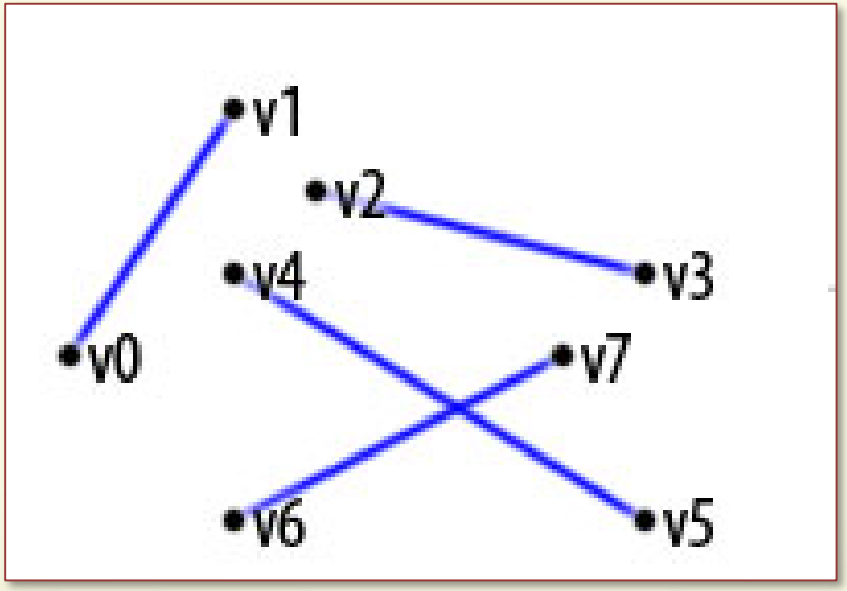
图形图像绘制

- GL_LINE_LOOP
  - 类同上，但是首尾相连，构成一个封闭曲线



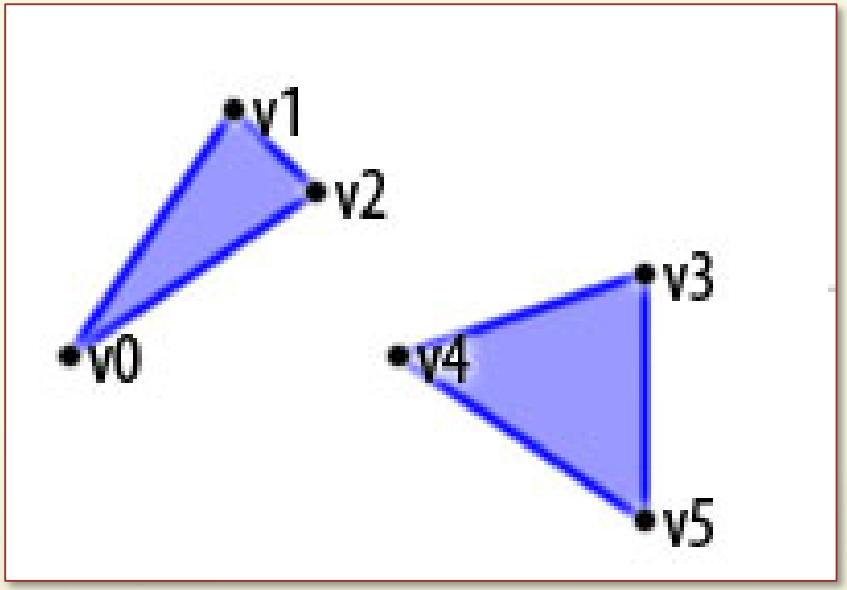
图形图像绘制

- GL_LINES - 顶点两两连接，为多条线段构成



图形图像绘制

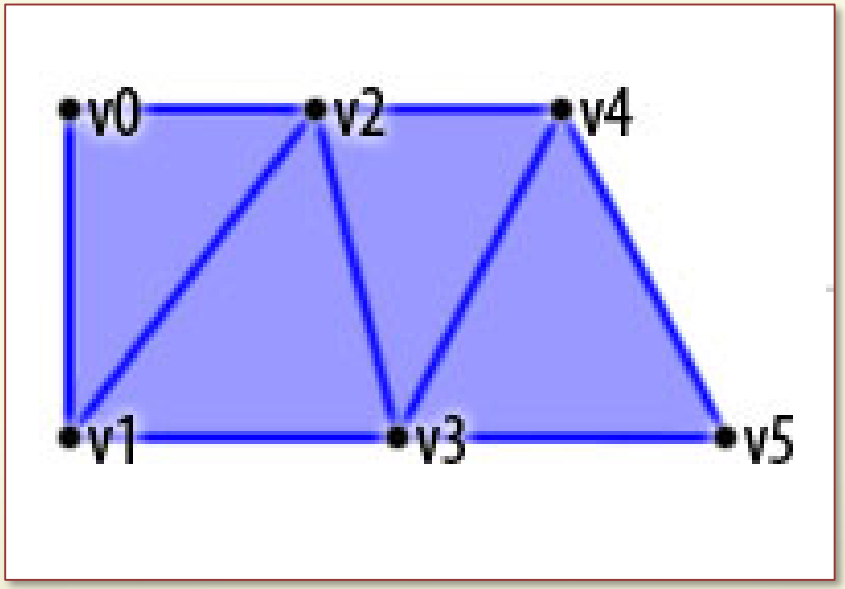
- GL_TRIANGLES
  - 每隔三个顶点构成一个三角形，为多个三角形组成



图形图像绘制

❖GL_TRIANGLE_STRIP

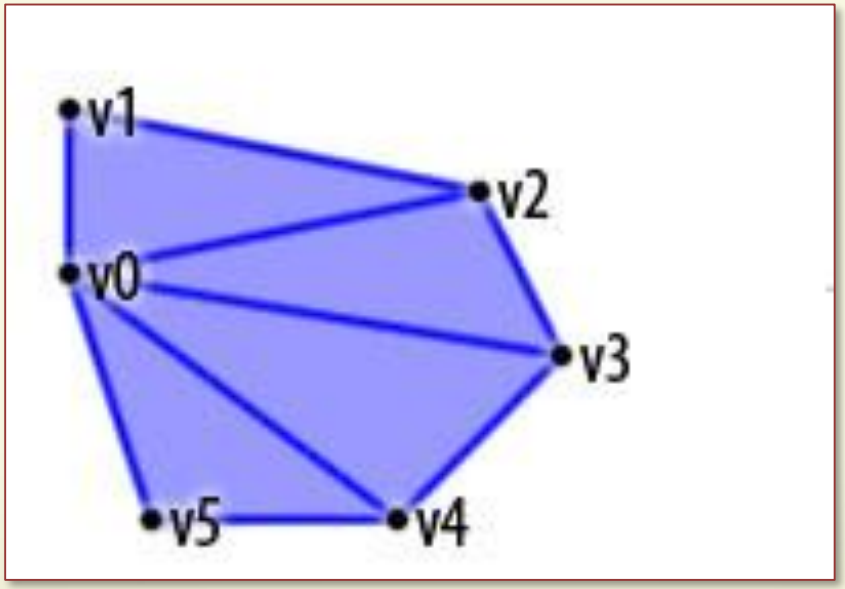
- 每相邻三个顶点组成一个三角形，为一系列相接三角形构成



图形图像绘制

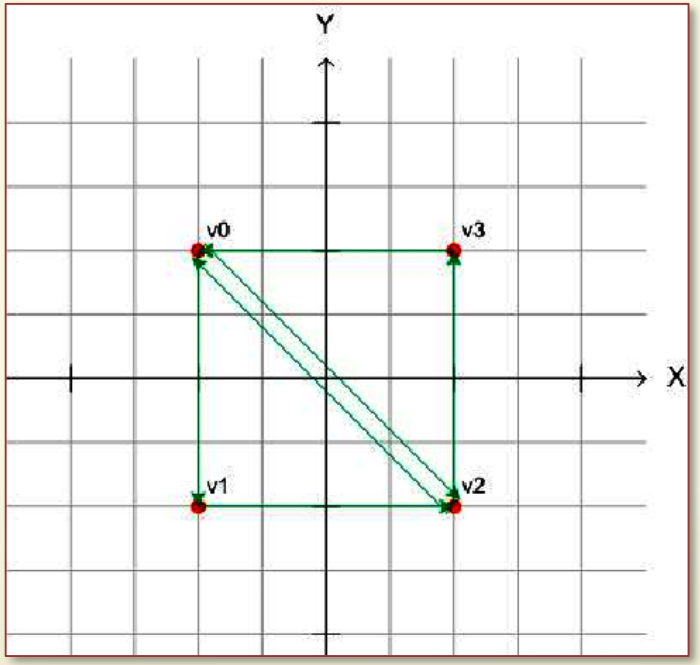
❖GL_TRIANGLE_FAN

- 以一个点为三角形公共顶点，组成一系列相邻的三角形



图形图像绘制

- 如果需要使用三角形来构造复杂图形，可以使用GL_TRIANGLE_STRIP或GL_TRIANGLE_FAN模式，通过定义顶点序列。如下图定义了一个正方形：



图形图像绘制

- OpenGL ES提供了两类方法来绘制一个空间几何图形，其中 mode 为上述解释顶点的模式：

```
// 使用VetexBuffer来绘制，顶点的顺序由vertexBuffer中的顺序指定
public abstract void glDrawArrays(int mode,int first,
                                  int count)

// 可以重新定义顶点的顺序，顶点的顺序由indices Buffer指定
public abstract void glDrawElements(int mode,int count,
                                    int type,Buffer indices)
```

图形图像绘制

- OpenGL ES中顶点一般使用数组来定义，并使用Buffer来存储以提高绘图性能，如下面定义三个顶点坐标，并把它们存放在FloatBuffer 中：

```
float [] vertexArray = {-0.8f, -0.4f*1.732f, 0.0f,
    0.8f, -0.4f*1.732f, 0.0f,  0.0f, 0.4f*1.732f, 0.0f,
};
ByteBuffer vbb =
          ByteBuffer.allocateDirect(vertexArray.length*4);
vbb.order(ByteOrder.nativeOrder());
FloatBuffer vertex = vbb.asFloatBuffer();
vertex.put(vertexArray);
vertex.position(0);
```

图形图像绘制

- 接下来是通过 glVertexAttribPointer() 设置渲染时使用到的数组数据。

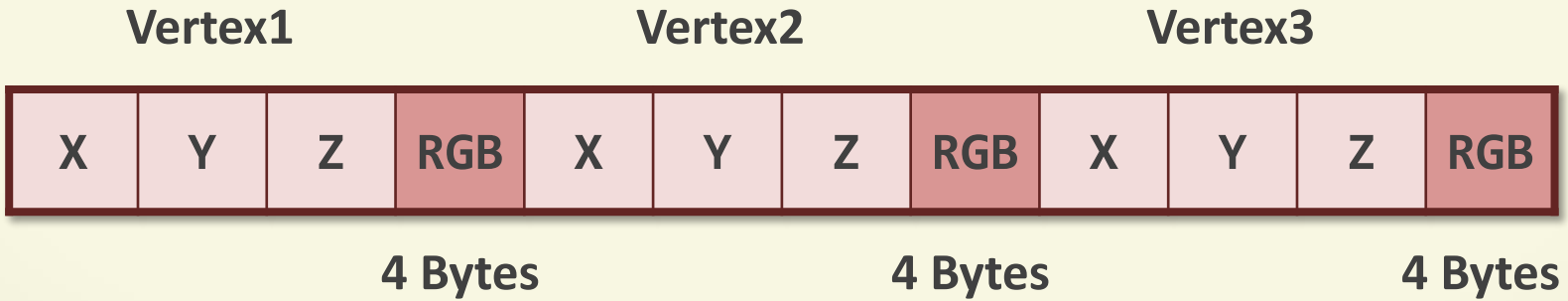
```
void glVertexAttribPointer(int index,
                           int size,
                           int type,
                           boolean normalized,
                           int stride,
                           Buffer pointer);
```

图形图像绘制

- index ： 指定要修改的顶点属性的索引值。
- size ： 每个顶点坐标维数，可以为2，3，4。
- type ： 顶点的数据类型，可以为GL_BYTE, GL_SHORT, GL_FIXED,或 GL_FLOAT，缺省为浮点类型GL_FLOAT。
- normalized ： 指定当被访问时，固定点数据值是否应该被归一化（GL_TRUE）或者直接转换为固定点值（GL_FALSE）。
- stride ： 每个相邻顶点之间在数组中的间隔（字节数），缺省为0，表示顶点存储之间无间隔。
- pointer: 存储顶点的数组。

图形图像绘制

- 可以将顶点的颜色值存放在对应顶点后面，如下图，RGB 采用4 字节表示，此时相邻顶点就不是连续存放的，stride 值为4。



Stride = 4

图形图像绘制

- 有了顶点的数据，就可以通过下面方法来控制数据是否可用：

```
void glEnableVertexAttribArray(int index);

...

void glDisableVertexAttribArray(int index);
```


THANK YOU