EGL is the interface between OpenGL ES and the underlying native display platform. It is used to create & manage rendering surfaces & graphics contexts.

eglMakeCurrent将context分别和当前渲染线程、drawSurface、readSurface绑定。drawSurface被用于所有gl操作，除了对取自readSurface帧buffer的任意像素数据的读回

(glReadPixels,glCopyTexImage2D,glCopyTexImage2D)。

如果调用线程已经绑定了一个当前渲染context，这个context会被flushed并被标记不再是当前context。

context第一次被绑定为当前context时，viewport和scissor的尺寸设置成drawSurface的

尺寸。viewPort和scissor不会被修改当后面又有eglMakeCurrent发生时。

通过将drawSurface和readSurface都设置成EGL\_NO\_SURFACE并设置context为

EGL\_NO\_CONTEXT，然后调用eglMakeCurrent，可以实现释放当前context而不需要分配新的context。

使用eglGetCurrentContext,eglGetCurrentDisplay和eglGetCurrentSurface来查询当前渲染

context，和关联的显示连接以及surfaces。

eglGetDisplay

eglInitialize:重复调用这个方法没有什么卵用。

eglTerminate:释放和display连接关联的资源。这个方法将所有和eglDisplay关联的资源标记为删除。如果和eglDisplay关联的context或者surface是任意线程的当前资源，那么这些相关资源将不会被释放直到他们不再是任意线程的当前资源。

eglChooseConfig(EGLDisplay display,

EGLint const \* attrib\_list,

EGLConfig \* configs,

EGLint config\_size,

EGLint \* num\_config)

返回值在configs中，所有满足attrib\_list 的egl frame buffer配置的一个列表。返回的eglConfig

能在任何需要egl frame buffer配置的egl函数中被使用。

如果configs不为空，取决于config\_size的configs会被放到configs中。而实际返回的configs

数目会写回到num\_config中。

如果configs为空，则没有返回configs。反而，匹配attrib\_list的configs的数目会写回到

Num\_config中。这种情况下，config\_size参数被忽略。这种形式的eglChooseConfig被用来决定匹配的frame buffer配置的数目，紧接着分配一个eglConfig的数组，然后参数不变的传递给下一个eglChooseConfig调用。

所有在attrib\_list中的属性，包括布尔属性，紧接着是对应的期望值。attrib\_list以

egl\_none结尾。如果一个属性没有在attrib\_list中指定，则使用该属性的默认值（

这就是通常所说的隐含指定）。

属性以特定属性的方式匹配。一些属性，例如egl\_level，必须准确的匹配指定的值。

其他的，例如egl\_red\_size，必须达到或者超过指定的最小值。如果有超过一个egl

frame buffer配置满足attrib\_list，然后一个以最佳匹配标准排列的配置列表被返回。

每一个属性的配置标准和准确的排序顺序如下所定义。

## openGl es

警告：在一个设备上支持openGl es 3.0，需要实现设备提供商提供的图形处理pipeline。所以一个4.3或者以上版本的设备可能不支持openGl es 3.0。

注意：android framework和J2ME提供的openGl es规范很像，但是不同，请注意区分。

1. 基础：

android通过两种方式支持openGl es：framework，ndk。

* glSurfaceView，你可以在它上面用openGl API来操作图形对象，首先创建它的实例，然后向它添加renderer。通过继承glSurfaceView可以实现对事件的处理。
* glSurfaceView.renderer
  1. openGl es包，见关键词。

1. 声明openGl要求

* openGl es版本要求

<!-- Tell the system this app requires OpenGL ES 3.1. -->  
<uses-feature android:glEsVersion="0x00030001" android:required="true" />

* 纹理压缩要求

如果你的应用使用了纹理压缩格式，你需要使用<supports-gl-texture>。

1. 图形对象的坐标映射

openGl坐标系统是个标准的正方形，而实际设备往往不是一个正方形，openGl映射和相机视图可以帮助解决这个问题，让你的图形对象在任何设备上都有正确的比例。为了使用openGl映射和相机视图，你需要创建一个映射矩阵和一个相机视图矩阵并将它们应用到渲染pipeline。映射矩阵重新计算图形对象的坐标，相机视图矩阵创建从特定视角呈现对象的变换。

* 1. openGl es 1.x中的映射和视角变换
     1. 映射矩阵确保图形的正确比例

public void onSurfaceChanged(GL10 gl, int width, int height) {  
    gl.glViewport(0, 0, width, height);  
    // make adjustments for screen ratio  
    float ratio = (float) width / height;

// set matrix to projection mode  
    gl.glMatrixMode(GL10.GL\_PROJECTION);

// reset the matrix to its default state  
    gl.glLoadIdentity();

// apply the projection matrix

gl.glFrustumf(-ratio, ratio, -1, 1, 3, 7);     
}

* + 1. 确保正确视角

public void onDrawFrame(GL10 gl) {  
    ...  
    // Set GL\_MODELVIEW transformation mode  
    gl.glMatrixMode(GL10.GL\_MODELVIEW);

// reset the matrix to its default state  
    gl.glLoadIdentity();                        
    // When using GL\_MODELVIEW, you must set the camera view  
    GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 0f, 1.0f, 0.0f);  
    ...  
}

* 1. openGl es 2.x和更高版本中的映射和视角变换

在2.x和3.x中，要使用映射和视角，首先你需要向图形对象的vertex shader添加矩阵成员。通过这个矩阵成员，你可以生成以及应用映射和视角矩阵到图形对象。

* + 1. 向vertex shader添加矩阵

创建一个代表映射矩阵的变量，并把它作为shader位置的左乘矩阵。下面是vertex shader的编码示例，包含其中的uMVPMatrix成员允许你使用这个vertex shader向图形对象的坐标应用映射和视角矩阵。

private final String vertexShaderCode =  
    // This matrix member variable provides a hook to manipulate  
    // the coordinates of objects that use this vertex shader.  
    "uniform mat4 uMVPMatrix;   \n" +  
    "attribute vec4 vPosition;  \n" +  
    "void main(){               \n" +  
    // The matrix must be included as part of gl\_Position  
    // Note that the uMVPMatrix factor \*must be first\* in order  
    // for the matrix multiplication product to be correct.  
    " gl\_Position = uMVPMatrix \* vPosition; \n" +  
    "}  \n";

注意：上例中使用了映射和视角矩阵的结合矩阵，你可以根据你自己的需要使用分开的两个矩阵。

* + 1. 访问shader矩阵

当你在vertex shader中创建了钩子以应用映射和视角变换，你可以通过访问这个变量去应用映射和视角矩阵了。

public void onSurfaceCreated(GL10 unused, EGLConfig config) {  
    ...  
    muMVPMatrixHandle = GLES20.glGetUniformLocation(mProgram, "uMVPMatrix");  
    ...  
}

* + 1. 创建映射和视角矩阵

public void onSurfaceCreated(GL10 unused, EGLConfig config) {  
    ...  
    // Create a camera view matrix  
    Matrix.setLookAtM(mVMatrix, 0, 0, 0, -3, 0f, 0f, 0f, 0f, 1.0f, 0.0f);  
}  
public void onSurfaceChanged(GL10 unused, int width, int height) {  
    GLES20.glViewport(0, 0, width, height);  
    float ratio = (float) width / height;  
    // create a projection matrix from device screen geometry  
    Matrix.frustumM(mProjMatrix, 0, -ratio, ratio, -1, 1, 3, 7);  
}

* + 1. 应用映射和视角矩阵

为了应用映射和视角变换，将这些矩阵相乘，然后设置给vertex shader。

public void onDrawFrame(GL10 unused) {  
    ...  
    // Combine the projection and camera view matrices  
    Matrix.multiplyMM(mMVPMatrix, 0, mProjMatrix, 0, mVMatrix, 0);  
    // Apply the combined projection and camera view transformations  
    GLES20.glUniformMatrix4fv(muMVPMatrixHandle, 1, false, mMVPMatrix, 0);  
    // Draw objects  
    ...  
}

1. 面和轨迹

face和winding有关，winding是定义shape的点的方向。openGl中约定逆时针绘画的面是正面，反之是反面。为什么区分face这么重要，这和openGl的一个通用特性有关，face culling。face culling允许openGl渲染pipeline忽略反面的渲染。

// enable face culling feature  
gl.glEnable(GL10.GL\_CULL\_FACE);  
// specify which faces to not draw  
gl.glCullFace(GL10.GL\_BACK);

如果你试图使用face culling特性而不去分清楚face，那么你的图形会看起来很薄或者根本没有。

1. openGl版本和设备兼容
   1. 纹理压缩支持
   2. 确定openGl扩展
      1. aep(android extension pack)

aep确保你的应用在opengl3.1标准描述的核心集上支持前文描述到的openGl扩展的标准化集。

* 1. 检测openGl es版本
     1. 方法一：创建高版本的eglContext，然后检测它。

private static double glVersion = 3.0;  
private static class ContextFactory implements GLSurfaceView.EGLContextFactory {  
private static int EGL\_CONTEXT\_CLIENT\_VERSION = 0x3098;  
public EGLContext createContext(  
          EGL10 egl, EGLDisplay display, EGLConfig eglConfig) {  
      Log.w(TAG, "creating OpenGL ES " + glVersion + " context");  
      int[] attrib\_list = {EGL\_CONTEXT\_CLIENT\_VERSION, (int) glVersion,  
              EGL10.EGL\_NONE };  
      // attempt to create a OpenGL ES 3.0 context  
      EGLContext context = egl.eglCreateContext(  
              display, eglConfig, EGL10.EGL\_NO\_CONTEXT, attrib\_list);  
      return context; // returns null if 3.0 is not supported;  
  }  
}

如果createContext返回null，则回退到2.0版本。

* + 1. 方法二：创建低版本的eglContext，然后检测版本值。

// Create a minimum supported OpenGL ES context, then check:  
String version = javax.microedition.khronos.opengles.GL10.glGetString(  
        GL10.GL\_VERSION);  
Log.w(TAG, "Version: " + version );  
// The version format is displayed as: "OpenGL ES <major>.<minor>"  
// followed by optional content provided by the implementation.

通过这种方法你能发现设备支持的高版本API，你必须先销毁最低版本的eglContext然后再创建新的。

1. 选择openGl API版本

2.x和3.x比较像，3.x代表着2.x的一个超集，添加了一些附加的特性。而1.x和后两个版本的差距就比较大，因此在使用这三大版本API之前，必须考虑以下因素：

* 1. 性能：一般来说，2.x和3.x的性能要比1.x的好。然而性能很大程度依赖openGl所运行的设备，因为不同的设备提供商对openGl es pipeline的实现不同。
  2. 设备兼容性：设备类型、android版本、openGl es版本等。
  3. 代码便捷： 1.x提供2.x和3.x不具备的固定pipeline和方便的函数，这可能会让新手觉得使用API 1.x编程更快。
  4. 图形处理控制：2.x和3.x通过shader提供完全可编程pipeline提供更高等级的控制。通过直接控制图形处理pipeline，开发者可以产生和1.x非常不同的效果。
  5. 纹理的支持：3.x对纹理压缩的支持是最好的，因为它保证对ETC2（支持alpha通道）的完美支持。1.x和2.x只提供对ETC1的支持，这种纹理格式不支持alpha通道，所以你只能使用目标设备支持的格式去提供资源。

以上五大因素会同时影响你的决定，你应该选择一个你认为能给你的用户带去更好体验的版本作为基础。

关键词：openGl es 1.x 2.x 3.x，glSurfaceView，glSurfaceView.renderer，android.opengl，

javax.microedition.khronos，texture compression format，projection modes，camera views，

vertex shader，shape faces，winding，face culling