OpenGL ES 基础入门

本文摘自网络，本人只是进行了一下规整，只作为学习参考使用，内容覆盖面并不全，但对于新手入门来说，有一定的帮助作用。

了解 OpenGL ES 社区

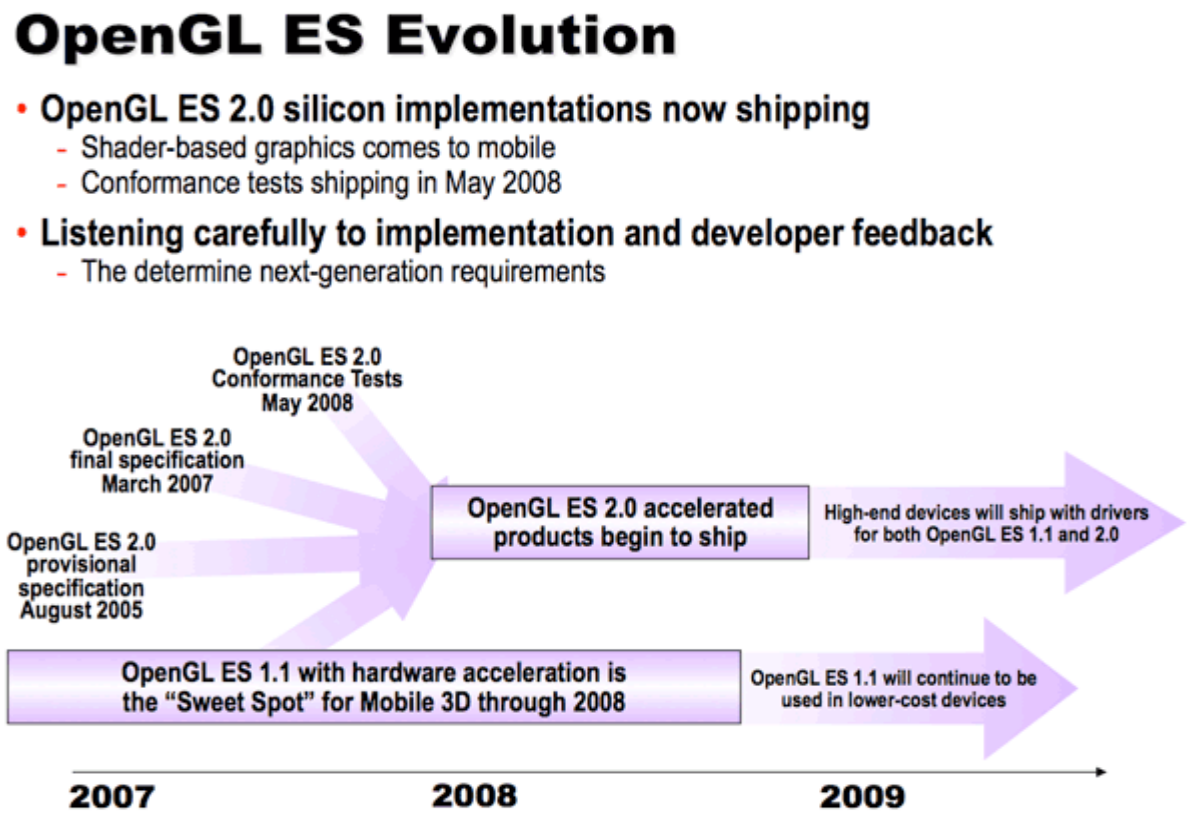
学习任何一种新技术，要先对它有一个全局性的了解，这样才知道用功的方向。而这全局性的了解中，非常重要的一块就是要了解该技术的社区情况。

OpenGL|ES 的官方组织是：http://www.khronos.org/ 该组织关注于手持和移动平台上的动态媒体编著、播放所需的 API，并致力于为这些 API 建立无限权费用的开放标准。(focused on the creation of open standard, royalty-free APIs to enable the authoring and accelerated playback of dynamic media on a wide variety of platforms and devices.)

在 Khronos 的网站上，有大量开发者资源，其中最重要的部份就是规范文档和头文件(EGL, GLES 1.x, GLES 2.x)，同时还列出了很多的例子和教程。

OpenGL|ES 是根据手持及移动平台的特点，对 OpenGL 3D 图形 API 标准进行裁剪定制而形成的，因此大多数 OpenGL 方面的知识都是可以借鉴的，因此保持对 OpenGL 官方组织的关注是非常有益的，OpenGL ARB 网站在

http://www.opengl.org 。Khronos 对两个 API 的关系有非常清楚的定义：



目前，各路厂商对 OpenGL ES 的支持才刚刚起步，在很多平台上都还没有官方的 OpenGL ES 实现，在这种情况，利用一些开源的产品也可以开始相关的研究和开发。其中发展得比较好的是 Vicent Mobile 3D Rendering Library (1.x http://sourceforge.net/projects/ogl-es/ , 2.x http://sf.net/projects/ogles2)，基于 OpenGL ES 的 3D 引擎 http://sourceforge.net/projects/es3d/，在 OpenGL 世界里广受欢迎的 GLUT 也出了 ES 版本 (http://sourceforge.net/projects/glutes/) ，不过个人感觉，有了 EGL 标准，开发者不必再面对 glx/wgl/agl 等一堆平台相关的初始化方法，glut 的作用已经小多了，不过对于从 OpenGL 平台转过来的用户还是非常友好的。

对于象笔者这样的 Dell Axim X50v/X51v 用户来说，intel 2700G 芯片的支持仍可以 intel 官方网站(http://www.intel.com)搜索找到，2700G 是基于 PowerVR NMX IP Core 研制的，也可以直接去 PowerVR 的网站下载 SDK 及演示程序。

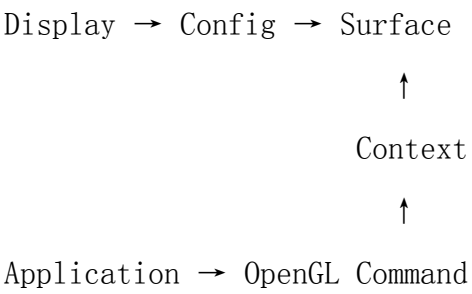
对于 Java 世界的开发者来说，JSR 239 跟 OpenGL ES 几乎是一样的。

已经出版的 OpenGL|ES 方面的书还不多，已知的有 OpenGL ES Game Development，一些传统经典 OpenGL 书籍的最新版中也会有所提及。

初始化 EGL

OpenGL ES 是一个平台中立的图形库，在它能够工作之前，需要与一个实际的窗口系统关联起来，这与 OpenGL 是一样的。但不一样的是，这部份工作有标准，这个标准就是 EGL。而 OpenGL 时代在不同平台上有不同的机制以关联窗口系统，在 Windows 上是 wgl，在 X-Window 上是 xgl，在 Apple OS 上是 agl 等。EGL 的工作方式和部份术语都接近于 xgl。

OpenGL ES 的初始化过程如下图所示意：



1. 获取 Display。
Display 代表显示器，在有些系统上可以有多个显示器，也就会有多个 Display。获得 Display 要调用 EGLboolean eglGetDisplay(NativeDisplay dpy)，参数一般为 EGL_DEFAULT_DISPLAY 。该参数实际的意义是平台实现相关的，在 X-Window 下是 XDisplay ID，在 MS Windows 下是 Window DC。

2. 初始化 egl。
调用 EGLboolean eglInitialize(EGLDisplay dpy, EGLint *major, EGLint *minor)，该函数会进行一些内部初始化工作，并传回 EGL 版本号(major.minor)。

3. 选择 Config。
所为 Config 实际指的是 FrameBuffer 的参数，在 MS Windows 下对应于 PixelFormat，在 X-Window 下对应 Visual。一般用 EGLboolean eglChooseConfig(EGLDisplay dpy, const EGLint * attr_list, EGLConfig * config, EGLint config_size, EGLint *num_config)，其中 attr_list 是以 EGL_NONE 结束的参数数组，通常以 id,value 依次存放，对于个别标识性的属性可以只有 id，没有 value。另一个办法是用 EGLboolean eglGetConfigs(EGLDisplay dpy, EGLConfig * config, EGLint config_size, EGLint *num_config) 来获得所有 config。这两个函数都会返回不多于 config_size 个 Config，结果保存在 config[]中，系统的总 Config 个数保存 在 num_config 中。可以利用 eglGetConfig()中间两个参数为 0 来查询系统支持的 Config 总个数。
Config 有众多的 Attribute，这些 Attribute 决定 FrameBuffer 的格式和能力，通过 eglGetConfigAttrib ()来读取，但不能修改。

4. 构造 Surface。
Surface 实际上就是一个 FrameBuffer，通过 EGLSurface eglCreateWindowSurface(EGLDisplay dpy, EGLConfig confg, NativeWindow win, EGLint *cfg_attr) 来创建一个可实际显示的 Surface。系统通常还支持另外两种 Surface：PixmapSurface 和 PBufferSurface，这两种都不 是

可显示的 Surface，PixmapSurface 是保存在系统内存中的位图，PBuffer 则是保存在显存中的帧。
Surface 也有一些 attribute，基本上都可以故名思意， EGL_HEIGHT EGL_WIDTH EGL_LARGEST_PBUFFER EGL_TEXTURE_FORMAT EGL_TEXTURE_TARGET EGL_MIPMAP_TEXTURE EGL_MIPMAP_LEVEL ， 通过 eglSurfaceAttrib() 设置、eglQuerySurface()读取。

5. 创建 Context。
OpenGL 的 pipeline 从程序的角度看就是一个状态机，有当前的颜色、纹理坐标、变换矩阵、绚染模式等一大堆状态，这些状态作用于程序提交的顶点 坐标等图元从而形成帧缓冲内的像素。在 OpenGL 的编程接口中，Context 就代表这个状态机，程序的主要工作就是向 Context 提供图元、设置状 态，偶尔也从 Context 里获取一些信息。
用 EGLContext eglCreateContext(EGLDisplay dpy, EGLSurface write, EGLSurface read, EGLContext * share_list)来创建一个 Context。

6. 绘制。
应用程序通过 OpenGL API 进行绘制，一帧完成之后，调用 eglSwapBuffers(EGLDisplay dpy, EGLContext ctx)来显示。

初始化 GLES

初始化 OpenGL ES 分四步：调用 eglInitialize()初始化 egl 库，用 eglChooseConfig()选择合适的 framebuffer，调用 eglCreateWindowSurface 创建 EGLSurface，用 eglCreateContext 创建 RenderContext(RC)。 其实跟标准 OpenGL 的初始化过程很接近(选择 PixelFormat 或者 Visual，再创建 Rc），只是多了一个 Window Surface 的概念。OpenGL 标准是不依赖于窗口系统的，这提供了很强的平台无关性，但也使得我们在 Windows 下要用 wgl 初始化，而 X-Window 下就得学会用 xgl，Mac OS X 上则是 agl。而手持及嵌入式市场的平台种类不计其数，单是学习各家手机操作系统的接口就是很大的负担了，更不用说总有一些有志于支持各种尺寸平台的软件开发者，所以 OpenGL ES 提供了 Window Surface 的抽象，使得移植工作可以基本局限在重新实现一下建立窗口的过程。

以下是我的 EGLWrapper 和 EGLConfigWrapper：

用法简单，包含"eglwrapper.h"，创建一个 EGLWrapper 实例，调用 init()，第一个参数是利用系统相关 API 建好的 window id，对 Windows CE 而言就是 HWND 了，程序结束时再 clean()一下就好。

```
1  //  @ Project : OpenGL ES Wrapper
2  //  @ File Name : eglwrapper.h
3  //  @ Date : 2006-7-5
4  //  @ Author : kongfu.yang
5  //  @ Company : http://www.play3d.net
```

```
  6 //  @ Copyright : 2006-7, Example source license.
        By keep this header comment,
          you may use this source file in your project for non-commicial
or commicial purpose.
  7
  8 #ifndef _EGL_WRAPPER_H_
  9 #define _EGL_WRAPPER_H_
 10
 11 #include <gles/gl.h>
 12
 13 #ifdef UNDER_CE
 14 #include <aygshell.h>
 15 #endif
 16 #include "eglConfigWrapper.h"
 17
 18 #include <stdio.h>
 19 #include <math.h>
 20
 21 #pragma warning ( disable : 4244 )
 22
 23 //#define Float2Fixed(fl) (fl)
 24 #define Float2Fixed(fl) ((GLfixed)((fl)*65536.0f))
 25
 26 class EGLWrapper
 27 {
 28 private:
 29     EGLDisplay dpy;
 30     EGLConfig config;
 31     EGLSurface surface;
 32     EGLContext context;
 33     EGLint major;
 34     EGLint minor;
 35
 36     NativeWindowType nativeWindow;
 37
 38     bool isFullScreen;
 39
 40 private:
 41     bool initialized;
 42
 43 public:
 44     EGLWrapper()
 45     {
 46         dpy = EGL_NO_DISPLAY;
```

```
 47          config = EGL_NONE;
 48          surface = EGL_NO_SURFACE;
 49          context = EGL_NO_CONTEXT;
 50          isFullScreen = false;
 51
 52          initialized = false;
 53      }
 54
 55      bool isInitialized(){ return initialized; }
 56
 57      void default3DEnv()
 58      {
 59          glClearColorx(Float2Fixed(0.5), Float2Fixed(0.5),
Float2Fixed(0.5), Float2Fixed(1.0));
 60
 61          glShadeModel(GL_SMOOTH);
 62          glEnable(GL_CULL_FACE);
 63          glCullFace(GL_BACK);
 64          glEnable(GL_DEPTH_TEST);
 65          glEnable(GL_TEXTURE_2D);
 66
 67          glMatrixMode(GL_PROJECTION);
 68          glLoadIdentity();
 69          //glFrustumx( Float2Fixed(-10.0f), Float2Fixed(10.0f),
Float2Fixed(-10.0f),
                                    Float2Fixed(10.0),
Float2Fixed(0.1f), Float2Fixed(100.0f) );
 70
 71          glMatrixMode(GL_MODELVIEW);
 72          glLoadIdentity();
 73      }
 74
 75      void resize(int width, int height)
 76      {
 77          glViewport( 0, 0, width, height );
 78
 79          glMatrixMode(GL_PROJECTION);
 80          glLoadIdentity();
 81
 82          float top = tan(45.0*3.14159/360.0) * 0.01;
 83          float right = (float)width/(float)height * top;
 84          glFrustumx( Float2Fixed(-right), Float2Fixed(right),
Float2Fixed(-top), Float2Fixed(top),
                  Float2Fixed(0.1f), Float2Fixed(1000.0f) );
```

```
 85
 86
 87          glMatrixMode(GL_MODELVIEW);
 88          glLoadIdentity();
 89          glTranslatex( 0, 0, Float2Fixed(-60.0f) );
 90      }
 91
 92      void toggleFullScreen()
 93      {
 94  #ifdef UNDER_CE
 95          extern HWND g_hWndMenuBar;
 96
 97          RECT rect;
 98
 99          isFullScreen = !isFullScreen;
100          if ( isFullScreen )
101          {
102              ShowWindow( g_hWndMenuBar, SW_HIDE );
103              SHFullScreen( (HWND) nativeWindow, SHFS_HIDETASKBAR |
SHFS_HIDESTARTICON | SHFS_HIDESIPBUTTON );
104              SetRect(&rect, 0, 0, GetSystemMetrics(SM_CXSCREEN),
GetSystemMetrics(SM_CYSCREEN));
105              MoveWindow((HWND) nativeWindow, rect.left, rect.top,
rect.right-rect.left, rect.bottom-rect.top, TRUE);
106          }else{
107              SystemParametersInfo(SPI_GETWORKAREA, 0, &rect,
FALSE);
108              MoveWindow((HWND) nativeWindow, rect.left, rect.top,
rect.right-rect.left, rect.bottom-rect.top, TRUE);
109              ShowWindow( g_hWndMenuBar, SW_SHOW );
110              SHFullScreen((HWND) nativeWindow, SHFS_SHOWTASKBAR |
SHFS_SHOWSTARTICON | SHFS_SHOWSIPBUTTON);
111          }
112  #endif
113      }
114
115      void setFullScreen()
116      {
117          if ( ! isFullScreen )
118              toggleFullScreen();
119      }
120
121      bool init(NativeWindowType hwnd, int bpp = 16, bool
fullscreen=false)
```

```
122     {
123         nativeWindow = hwnd;
124 
125         if( fullscreen ) toggleFullScreen();
126 
127         dpy = eglGetDisplay( (NativeDisplayType)
GetDC( (HWND)hwnd ) );
128         if ( ! dpy )
129             return false;
130 
131         if ( eglInitialize(dpy, &major, &minor) == EGL_FALSE )
132         {
133             return false;
134         }
135 
136         // choose config
137         EGLint cfg_attr_list[] = { EGL_BUFFER_SIZE, bpp, EGL_NONE};
138         int num = 0;
139         if ( eglChooseConfig(dpy, cfg_attr_list, &config, 1, &num)
== EGL_FALSE || num == 0 )
140         {
141             return false;
142         }
143 
144         // create surface
145         // EGLint sf_attr_list[] = {};
146         surface = eglCreateWindowSurface(dpy, config, hwnd, 0);
147         if ( surface == EGL_NO_SURFACE )
148         {
149             return false;
150         }
151 
152         // create context
153         // EGLint ctx_attr_list[] = { EGL_NONE };
154         context = eglCreateContext(dpy, config, NULL, NULL);
155 
156         // active context (make current)
157         initialized = makeCurrent();
158 
159         default3DEnv();
160 
161         glClear( GL_COLOR_BUFFER_BIT );
162 
163         return initialized;
```

```
164      }
165
166      bool makeCurrent()
167      {
168          EGLBoolean ret = eglMakeCurrent(dpy, surface, surface,
context);
169          return ret != EGL_FALSE;
170      }
171
172      bool swapBuffers()
173      {
174          return GL_TRUE == eglSwapBuffers(dpy, surface);
175      }
176
177      int getVersionMajor(){ return major; }
178      int getVersionMinor(){ return minor; }
179
180      void clean()
181      {
182          if ( isFullScreen )
183              toggleFullScreen();
184
185          swapBuffers();
186
187          // deactive context
188          // eglMakeCurrent( dpy, NULL, NULL, NULL );
189          eglMakeCurrent(dpy, EGL_NO_SURFACE, EGL_NO_SURFACE,
EGL_NO_CONTEXT) ;
190
191          // destroy context
192          if ( context != EGL_NO_CONTEXT )
193              eglDestroyContext( dpy, context );
194
195          // destroy suerface
196          if ( surface != EGL_NO_SURFACE )
197              eglDestroySurface( dpy, surface );
198
199          // terminate display
200          if ( dpy != EGL_NO_DISPLAY )
201              eglTerminate( dpy );
202      }
203
204      // Return attribute values description of all config in the
parameter configs
```

```
205      void describeConfigs(int num, EGLConfig *configs,
ConfigDescription *configDesc)
206      {
207          EGLConfig * ptr = configs;
208          for ( int i = 0; i < num; ++i,++ptr )
209          {
210              EGLConfig * ptr = configs;
211              for ( int i = 0; i < num; ++i,++ptr )
212              {
213                  configDesc[i].describe( dpy, *ptr );
214              }
215          }
216      }
217
218      // Notes: caller MUST response to delete[] the momery pointing
by configs allocated inside
219      bool getConfigDescriptions(int & num, ConfigDescription **
configDesc)
220      {
221          bool result = false;
222          if ( eglGetConfigs( dpy, 0, 0, &num ) )
223          {
224              // get configs
225              EGLConfig * configs = new EGLConfig[num]();
226              * configDesc = new ConfigDescription[num]();
227              if ( eglGetConfigs( dpy, configs, num, &num) )
228              {
229                  describeConfigs(num, configs, *configDesc);
230                  result = true;
231              }
232              delete [] configs;
233          }
234
235          return result;
236      }
237
238      void dumpConfig()
239      {
240          int num = 0;
241          if ( eglGetConfigs( dpy, 0, 0, &num ) )
242          {
243              // get configs
244              EGLConfig *configs = (EGLConfig *) malloc( num *
sizeof(EGLConfig) );
```

```
245             ConfigDescription * configDesc = new
ConfigDescription[num]();
246             if ( eglGetConfigs( dpy, configs, num, &num) )
247             {
248                 describeConfigs( num, configs, configDesc);
249
250                 // export config details
251                 FILE * fp = fopen( "config_attribute.csv", "wb" );
252
253                 // print titles
254                 fprintf(fp, "attrName");
255                 EGLConfig * ptr = configs;
256                 for ( int i = 0; i < num; ++i, ++ptr )
257                     fprintf(fp, ",config %d", *ptr);
258                 fprintf(fp, ",explain");
259                 fprintf(fp, "\n");
260
261                 char buf[1024];
262                 // print attributes, one attribute per line, name
and value of each config
263                 for ( int j = 0; j < MAX_ATTRIBUTES_OF_CONFIG &&
configDesc[0].attributes[j].id != 0; ++j )
264                 {
265                     memset( buf, 0,1024);
266                     WideCharToMultiByte( CP_ACP, 0,
configDesc[0].attributes[j].name,

wcslen(configDesc[0].attributes[j].name), buf, 1024, NULL, NULL );
267                     fprintf(fp, "\"%s\"", buf);
268                     for ( int i = 0; i < num; ++ i )
269                     {
270                         fprintf( fp, ", 0x%x",
configDesc[i].attributes[j].value );
271                     }
272                     memset( buf, 0,1024);
273                     WideCharToMultiByte( CP_ACP, 0,
configDesc[0].attributes[j].explain,

wcslen(configDesc[0].attributes[j].explain), buf, 1024, NULL, NULL );
274                     fprintf( fp, ",\"%s\"", buf );
275                     fprintf(fp, "\n");
276                 }
277
278                 fclose(fp);
```

```
279             }
280             delete [] configDesc;
281             free( configs );
282         }
283     }
284
285     struct eglErrorString
286     {
287         GLint code;
288         const TCHAR * message;
289     };
290
291     const TCHAR * getErrorString(GLint errorCode)
292     {
293         static eglErrorString errors[] = {
294             { EGL_SUCCESS,
295             TEXT("Function succeeded.") },
296
297             { EGL_NOT_INITIALIZED,
298             TEXT("EGL is not initialized, or could not be
initialized, for the specified display.") },
299
300             { EGL_BAD_ACCESS,
301             TEXT("EGL cannot access a requested resource (for
example, a context is bound in another thread).") },
302
303             { EGL_BAD_ALLOC,
304             TEXT("EGL failed to allocate resources for the
requested operation.") },
305
306             { EGL_BAD_ATTRIBUTE,
307             TEXT("An unrecognized attribute or attribute value was
passed in an attribute list.") },
308
309             { EGL_BAD_CONTEXT,
310             TEXT("An EGLContext argument does not name a valid
EGLContext.") },
311
312             { EGL_BAD_CONFIG,
313             TEXT("An EGLConfig argument does not name a valid
EGLConfig.") },
314
315             { EGL_BAD_CURRENT_SURFACE,
```

```
316             TEXT("The current surface of the calling thread is a
window, pbuffer, or pixmap that is no longer valid.") },
317
318             { EGL_BAD_DISPLAY,
319             TEXT("An EGLDisplay argument does not name a valid
EGLDisplay; or,
                                        EGL is not initialized on the
specified EGLDisplay.") },
320
321             { EGL_BAD_SURFACE,
322             TEXT("An EGLSurface argument does not name a valid
surface (window, pbuffer, or pixmap)
                                        configured for OpenGL ES
rendering.") },
323
324             { EGL_BAD_MATCH,
325             TEXT("Arguments are inconsistent; for example, an
otherwise valid context requires buffers
                                        (e.g. depth or stencil) not
allocated by an otherwise valid surface.") },
326
327             { EGL_BAD_PARAMETER,
328             TEXT("One or more argument values are invalid.") },
329
330             { EGL_BAD_NATIVE_PIXMAP,
331             TEXT("A NativePixmapType argument does not refer to a
valid native pixmap.") },
332
333             { EGL_BAD_NATIVE_WINDOW,
334             TEXT("A NativeWindowType argument does not refer to a
valid native window.") },
335
336             { EGL_CONTEXT_LOST,
337             TEXT("A power management event has occurred. The
application must destroy all contexts
                        and reinitialise OpenGL ES state and objects to
continue rendering, as described in section 2.6.") }
338         };
339
340         for ( int i = 0; i < sizeof(errors); ++i )
341         {
342             if ( errors[i].code == errorCode )
343                 return errors[i].message;
344         }
```

```
345
346             return NULL;
347         }
348     };
349
350     #endif // _EGL_WRAPPER_H_
  1     //  @ Project : OpenGL ES Wrapper
  2     //  @ File Name : eglConfigWrapper.h
  3     //  @ Date : 2006-7-5
  4     //  @ Author : kongfu.yang
  5     //  @ Company : http://www.play3d.net
  6     //  @ Copyright : 2006-7, Example source license.
            By keep this header comment,
              you may use this source file in your project for non-commicial
    or commicial purpose.
  7
  8
  9     #ifndef _EGL_CONFIG_WRAPPER_H_
 10     #define _EGL_CONFIG_WRAPPER_H_
 11
 12     #include <gles/egl.h>
 13
 14     #define MAX_ATTRIBUTES_OF_CONFIG 0x30
 15
 16     struct ConfigAttribute
 17     {
 18             EGLint id;
 19             EGLint value;
 20             TCHAR * name;
 21             TCHAR * explain;
 22     };
 23
 24     class ConfigDescription
 25     {
 26     public:
 27       ConfigAttribute attributes[MAX_ATTRIBUTES_OF_CONFIG];
 28
 29       ConfigDescription()
 30       {
 31           init();
 32       }
 33
 34       void init()
 35       {
```

```
 36        int i = 0;
 37
 38        attributes[i].id = EGL_BUFFER_SIZE;
 39        attributes[i].name = TEXT("EGL_BUFFER_SIZE");
 40        attributes[i++].explain = TEXT("color buffer bits, r+g+b+
alpha");
 41
 42        attributes[i].id = EGL_ALPHA_SIZE;
 43        attributes[i].name = TEXT("EGL_ALPHA_SIZE");
 44        attributes[i++].explain = TEXT("bits for alpha");
 45
 46        attributes[i].id = EGL_BLUE_SIZE;
 47        attributes[i].name = TEXT("EGL_BLUE_SIZE");
 48        attributes[i++].explain = TEXT("blue color bits");
 49
 50        attributes[i].id = EGL_GREEN_SIZE;
 51        attributes[i].name = TEXT("EGL_GREEN_SIZE");
 52        attributes[i++].explain = TEXT("green color bits");
 53
 54        attributes[i].id = EGL_RED_SIZE;
 55        attributes[i].name = TEXT("EGL_RED_SIZE");
 56        attributes[i++].explain = TEXT("red color bits");
 57
 58        attributes[i].id = EGL_DEPTH_SIZE;
 59        attributes[i].name = TEXT("EGL_DEPTH_SIZE");
 60        attributes[i++].explain = TEXT("z buffer depth");
 61
 62        attributes[i].id = EGL_STENCIL_SIZE;
 63        attributes[i].name = TEXT("EGL_STENCIL_SIZE");
 64        attributes[i++].explain = TEXT("stancil bits per pixel");
 65
 66        attributes[i].id = EGL_CONFIG_CAVEAT;
 67        attributes[i].name = TEXT("EGL_CONFIG_CAVEAT");
 68        attributes[i++].explain = TEXT("side effect of the config,
EGL_NONE(0x3038), EGL_SLOW_CONFIG(0x3050),
                        EGL_NON_COMFORMANT_CONFIG(0x3051,native
optimized, but failed to EGL standard comformant test)");
 69
 70        attributes[i].id = EGL_CONFIG_ID;
 71        attributes[i].name = TEXT("EGL_CONFIG_ID");
 72        attributes[i++].explain = TEXT("given config ID");
 73
 74        attributes[i].id = EGL_LEVEL;
 75        attributes[i].name = TEXT("EGL_LEVEL");
```

```
 76        attributes[i++].explain = TEXT("0 is defalt, <0 underlay, >0
overlay");
 77
 78        attributes[i].id = EGL_MAX_PBUFFER_PIXELS;
 79        attributes[i].name = TEXT("EGL_MAX_PBUFFER_PIXELS");
 80        attributes[i++].explain = TEXT("maximum pixels in a pbuffer,
maybe not max_width * max_height");
 81
 82        attributes[i].id = EGL_MAX_PBUFFER_HEIGHT;
 83        attributes[i].name = TEXT("EGL_MAX_PBUFFER_HEIGHT");
 84        attributes[i++].explain = TEXT("maximum pixels in y
direction");
 85
 86        attributes[i].id = EGL_MAX_PBUFFER_WIDTH;
 87        attributes[i].name = TEXT("EGL_MAX_PBUFFER_WIDTH");
 88        attributes[i++].explain = TEXT("maximum pixels in x
direction");
 89
 90        attributes[i].id = EGL_NATIVE_RENDERABLE;
 91        attributes[i].name = TEXT("EGL_NATIVE_RENDERABLE");
 92        attributes[i++].explain = TEXT("native API (GDI) can draw on
EGL surface");
 93
 94        attributes[i].id = EGL_NATIVE_VISUAL_ID;
 95        attributes[i].name = TEXT("EGL_NATIVE_VISUAL_ID");
 96        attributes[i++].explain = TEXT("native visual (pixel format)
ID");
 97
 98        attributes[i].id = EGL_NATIVE_VISUAL_TYPE;
 99        attributes[i].name = TEXT("EGL_NATIVE_VISUAL_TYPE");
100        attributes[i++].explain = TEXT("native visual type
(PIXELFORMAT) ?");
101
102        attributes[i].id = EGL_SAMPLES;
103        attributes[i].name = TEXT("EGL_SAMPLES");
104        attributes[i++].explain = TEXT("sample count required by a
multisampling buffer. 0 if none sample buffer");
105
106        attributes[i].id = EGL_SAMPLE_BUFFERS;
107        attributes[i].name = TEXT("EGL_SAMPLE_BUFFERS");
108        attributes[i++].explain = TEXT("number of multisampling
buffers");
109
110        attributes[i].id = EGL_SURFACE_TYPE;
```

```
111         attributes[i].name = TEXT("EGL_SURFACE_TYPE");
112         attributes[i++].explain = TEXT("supported surface type in
config,

EGL_WINDOW_BIT(1)|EGL_PIXMAP_BIT(2)|EGL_PBUFFER_BIT(4)");
113
114         attributes[i].id = EGL_TRANSPARENT_TYPE;
115         attributes[i].name = TEXT("EGL_TRANSPARENT_TYPE");
116         attributes[i++].explain = TEXT("support a special color as
transparent. EGL_NONE(0x3038),

EGL_TRANSPARENT_RGB(0x3052)");
117
118         attributes[i].id = EGL_TRANSPARENT_BLUE_VALUE;
119         attributes[i].name = TEXT("EGL_TRANSPARENT_BLUE_VALUE");
120         attributes[i++].explain = TEXT("blue component of
transparent color");
121
122         attributes[i].id = EGL_TRANSPARENT_GREEN_VALUE;
123         attributes[i].name = TEXT("EGL_TRANSPARENT_GREEN_VALUE");
124         attributes[i++].explain = TEXT("green component of
transparent color");
125
126         attributes[i].id = EGL_TRANSPARENT_RED_VALUE;
127         attributes[i].name = TEXT("EGL_TRANSPARENT_RED_VALUE");
128         attributes[i++].explain = TEXT("red component of transparent
color");
129
130         // EGL 1.1
131         attributes[i].id = EGL_BIND_TO_TEXTURE_RGB;
132         attributes[i].name = TEXT("EGL_BIND_TO_TEXTURE_RGB");
133         attributes[i++].explain = TEXT("true if bindalble to RGB
texture");
134
135         attributes[i].id = EGL_BIND_TO_TEXTURE_RGBA;
136         attributes[i].name = TEXT("EGL_BIND_TO_TEXTURE_RGBA");
137         attributes[i++].explain = TEXT("true if bindalbe to RGBA
texture");
138
139         attributes[i].id = EGL_MIN_SWAP_INTERVAL;
140         attributes[i].name = TEXT("EGL_MIN_SWAP_INTERVAL");
141         attributes[i++].explain = TEXT("minimum swap interval");
142
143         attributes[i].id = EGL_MAX_SWAP_INTERVAL;
```

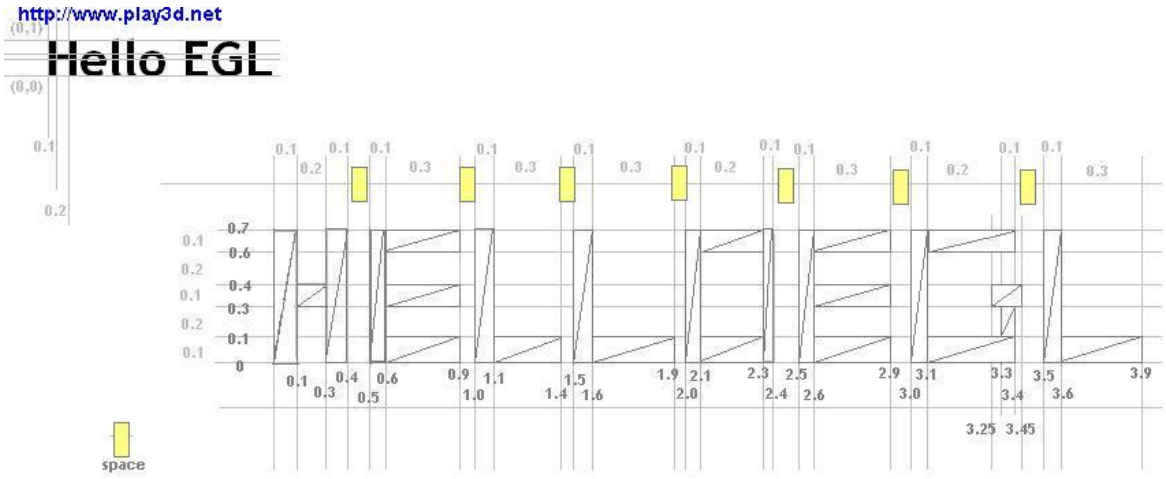
```
144         attributes[i].name = TEXT("EGL_MAX_SWAP_INTERVAL");
145         attributes[i++].explain = TEXT("maximum swap interval");
146 
147         attributes[i].id = 0;
148         attributes[i].value = 0;
149         attributes[i].name = 0;
150         attributes[i++].explain = 0;
151     }
152 
153       bool describe(EGLDisplay dpy, EGLConfig config)
154       {
155           for ( int i = 0; i < MAX_ATTRIBUTES_OF_CONFIG &&
attributes[i].id != 0; ++i )
156           {
157               eglGetConfigAttrib( dpy, config, attributes[i].id,
&attributes[i].value);
158           }
159 
160           return true;
161       }
162   };
163 
164   #endif //_EGL_CONFIG_WRAPPER_H_
```

Hello,EGL

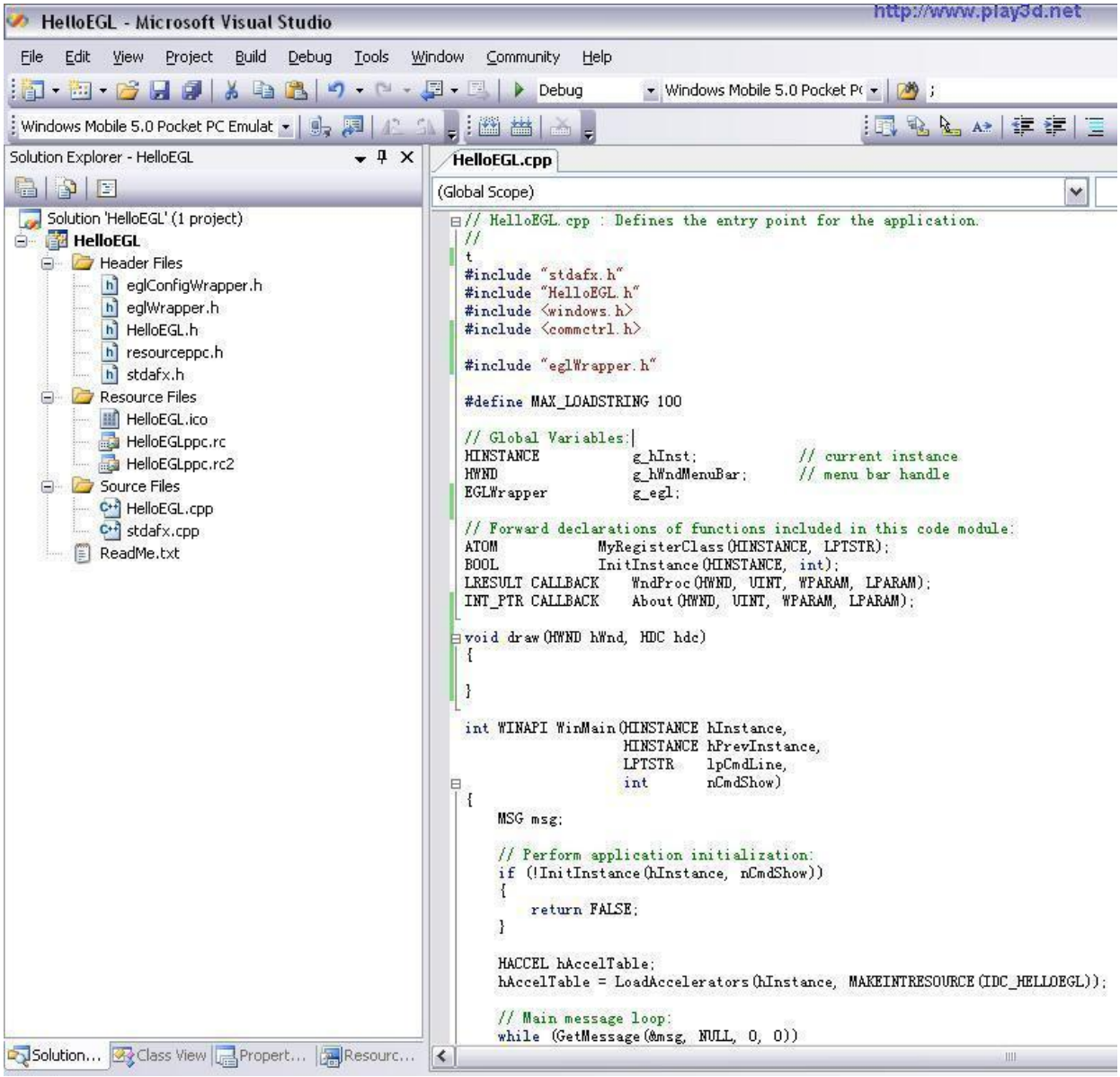
注：本文中例子针对 Windows CE 平台编写。

OpenGL ES 简化了模型描述，取消了通过在 glBegin/glEnd 之间使用大量 glVertex 之类的调用来逐点描述模型，统一到使用 VertexArray。

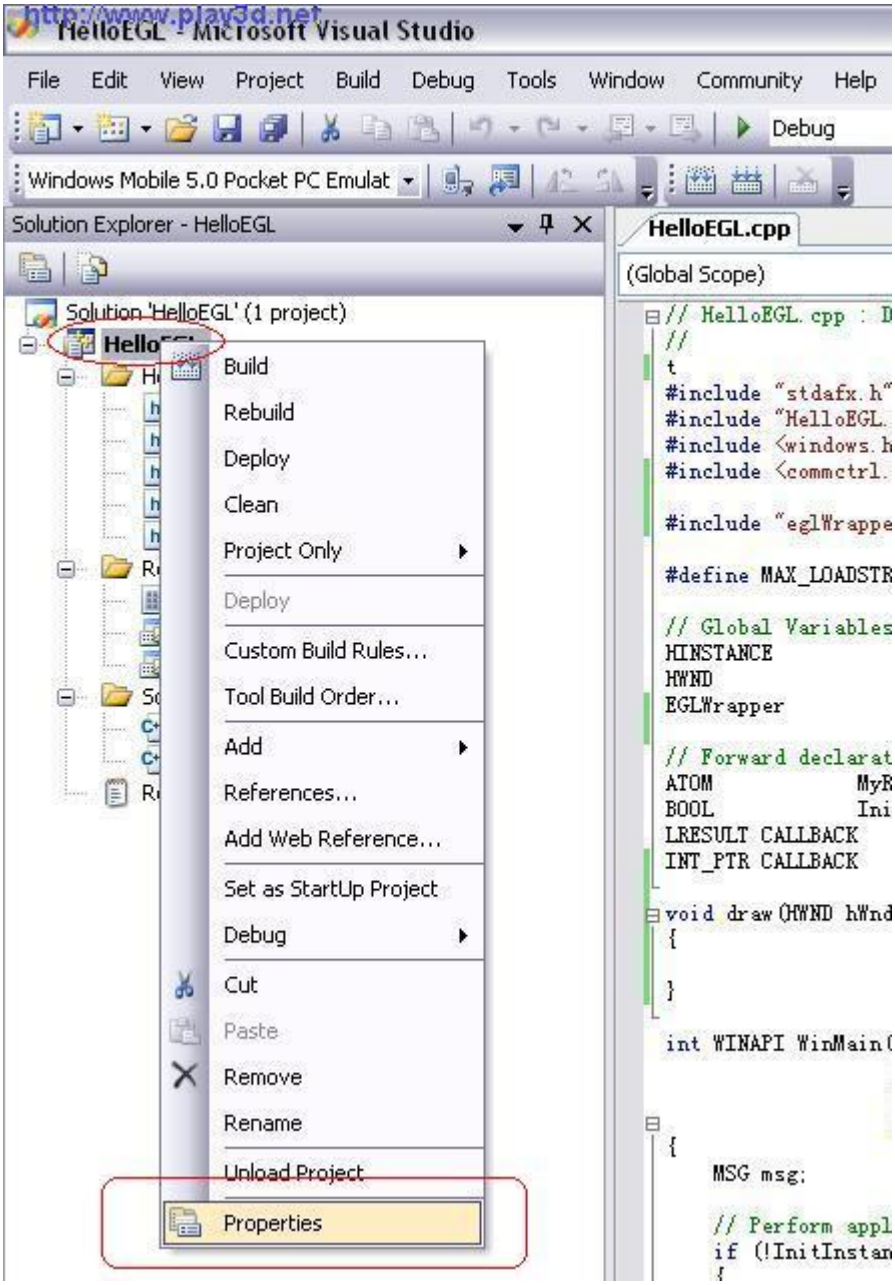
下面我们就来画一个简单的物体：用三角形拼出的 HELLO EGL。该物体的示意图如下：



照例，先 New App：

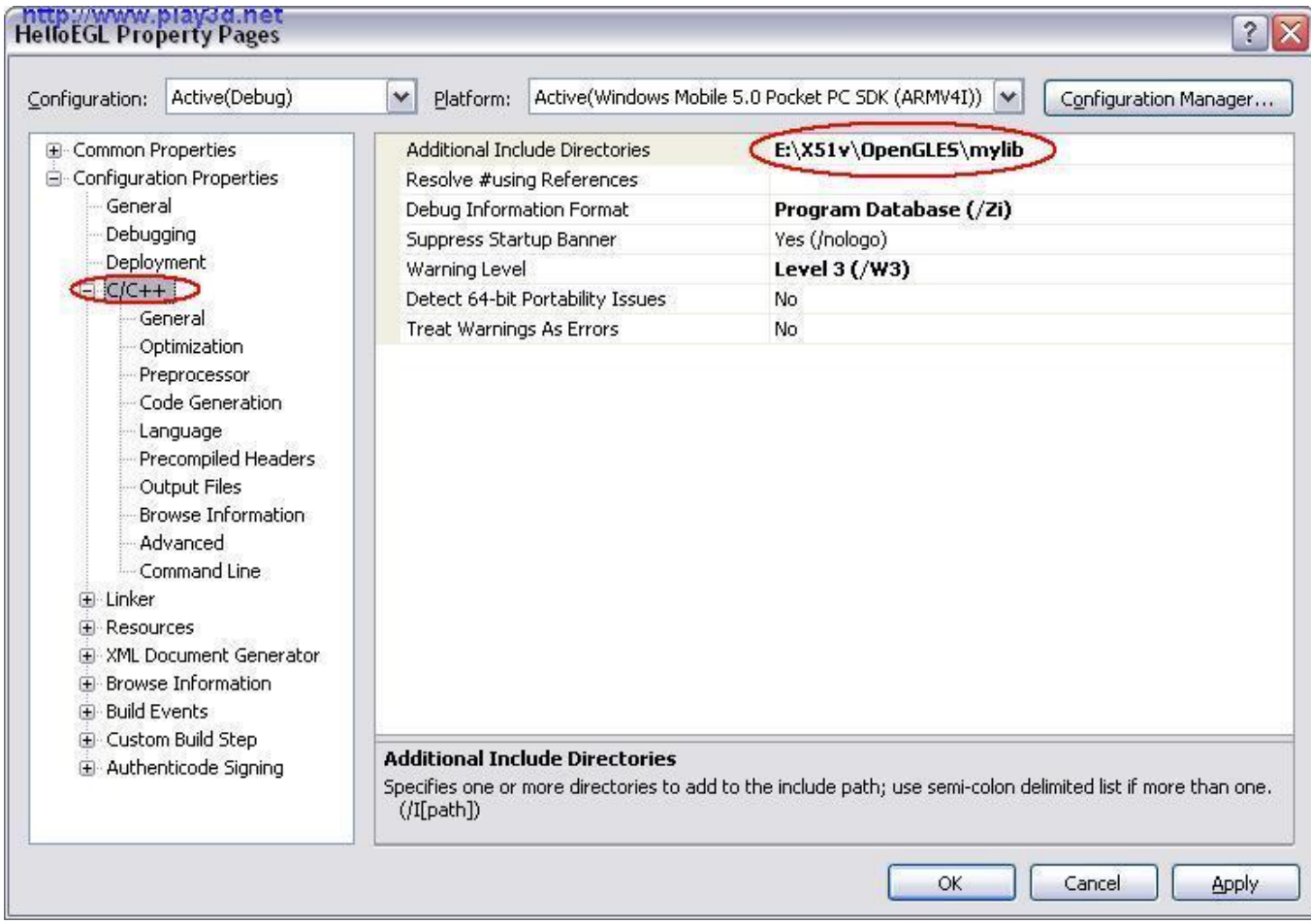

HelloEGL - Microsoft Visual Studio
http://www.play3d.net
File Edit View Project Build Debug Tools Window Community Help
Debug
Windows Mobile 5.0 Pocket PC Emulat
Solution Explorer - HelloEGL
Solution 'HelloEGL' (1 project)
HelloEGL
Header Files
eglConfigWrapper.h
eglWrapper.h
HelloEGL.h
resourceppc.h
stdafx.h
Resource Files
HelloEGL.ico
HelloEGLppc.rc
HelloEGLppc.rc2
Source Files
HelloEGL.cpp
stdafx.cpp
ReadMe.txt
Solution...
Class View
Propert...
Resourc...
HelloEGL.cpp
(Global Scope)
// HelloEGL.cpp : Defines the entry point for the application.
//
t
#include "stdafx.h"
#include "HelloEGL.h"
#include <windows.h>
#include <commctrl.h>
#include "eglWrapper.h"
#define MAX_LOADSTRING 100
// Global Variables:
HINSTANCE g_hInst; // current instance
HWND g_hWndMenuBar; // menu bar handle
EGLWrapper g_egl;
// Forward declarations of functions included in this code module:
ATOM MyRegisterClass(HINSTANCE, LPTSTR);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);
void draw(HWND hWnd, HDC hdc)
{
}
int WINAPI WinMain(HINSTANCE hInstance,
HINSTANCE hPrevInstance,
LPTSTR lpCmdLine,
int nCmdShow)
{
MSG msg;
// Perform application initialization:
if (!InitInstance(hInstance, nCmdShow))
{
return FALSE;
}
HACCEL hAccelTable;
hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_HELLOEGL));
// Main message loop:
while (GetMessage(&msg, NULL, 0, 0))

然后设置一下项目属性：

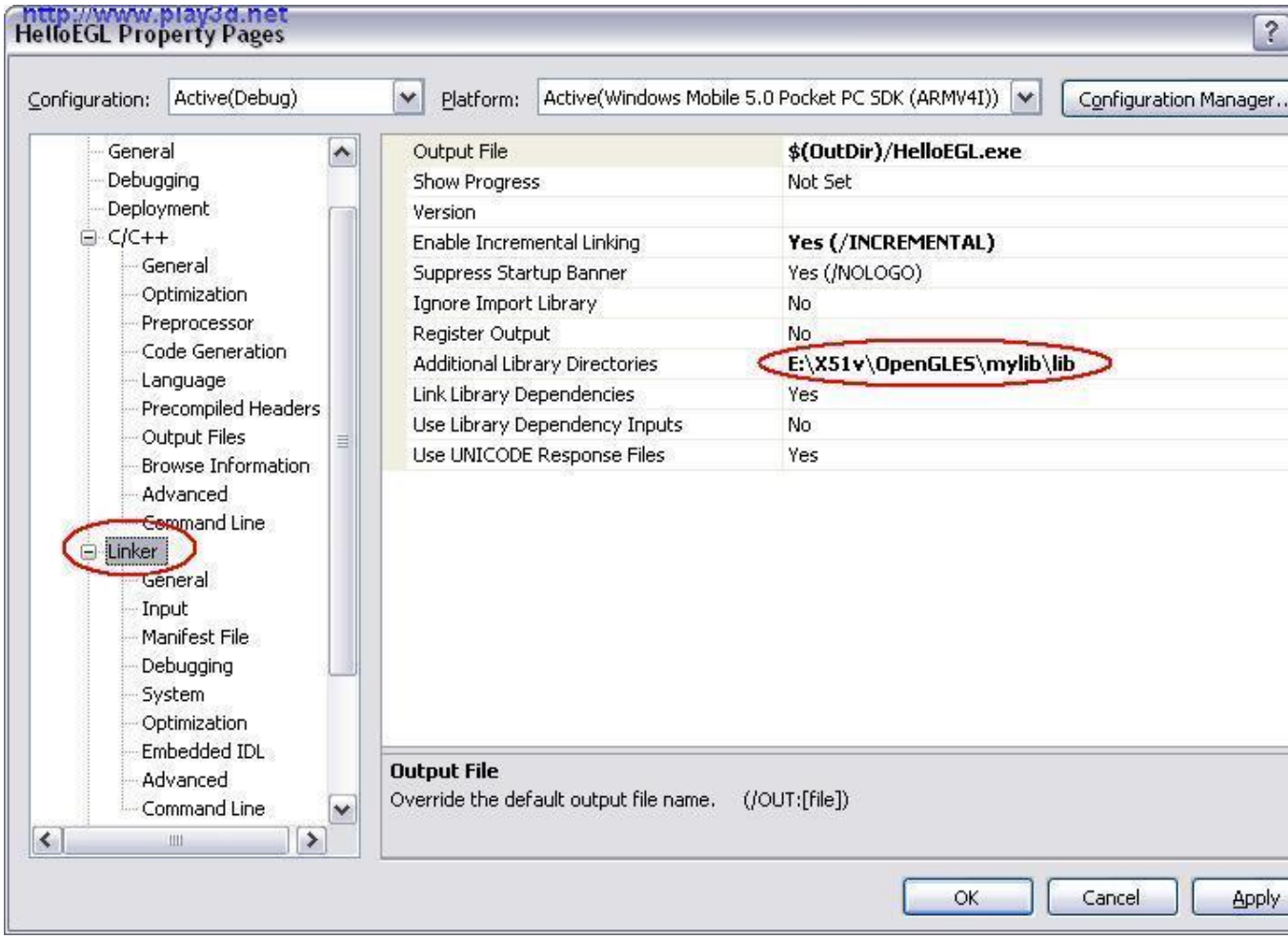


共有三处要改的，使得该项目可以利用上如下图所示的库和头文件：

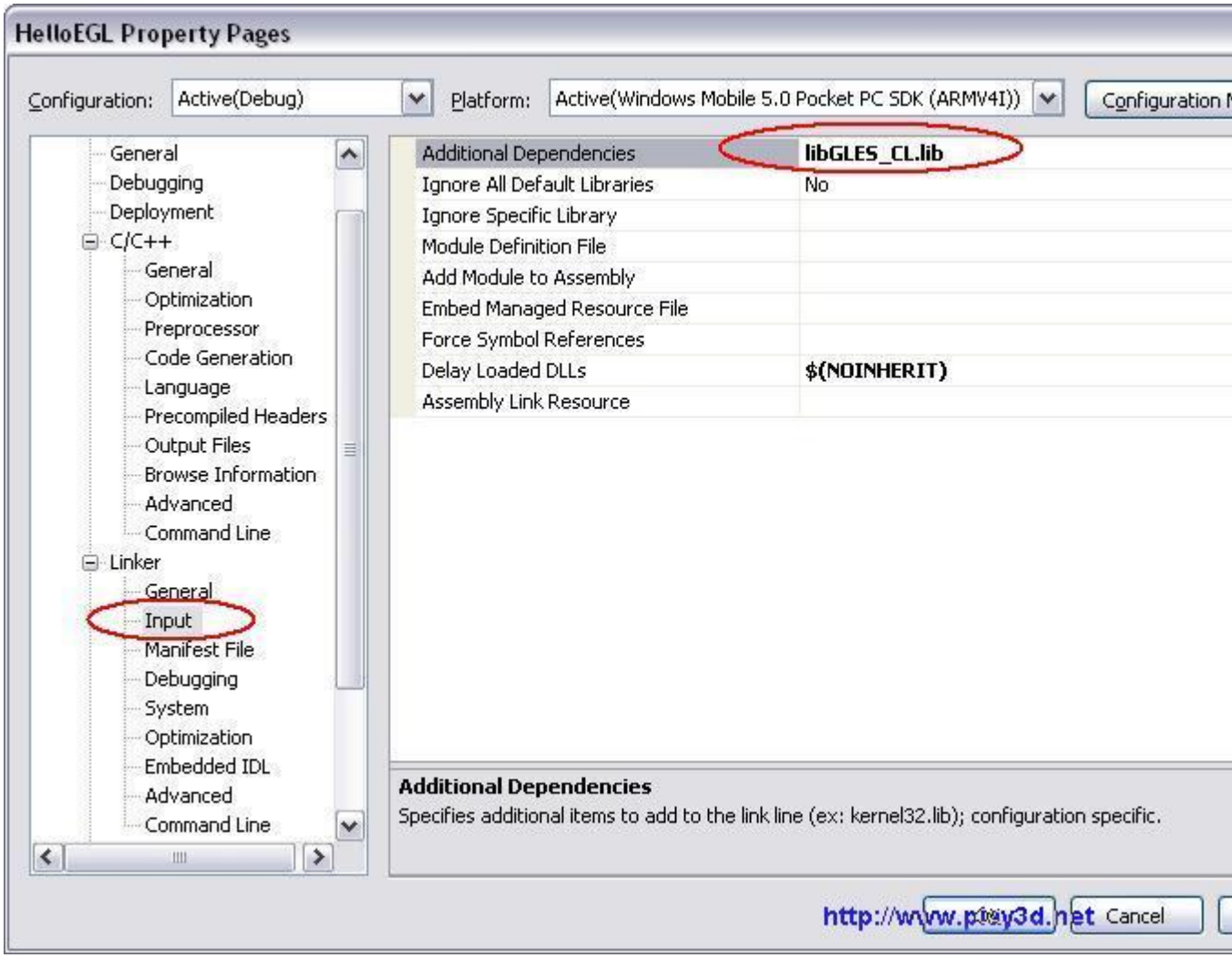
首先是头文件包含目录：


http://www.play3d.net
HelloEGL Property Pages
Configuration: Active(Debug)
Platform: Active(Windows Mobile 5.0 Pocket PC SDK (ARMV4I))
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
Deployment
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
Resources
XML Document Generator
Browse Information
Build Events
Custom Build Step
Authenticode Signing
Additional Include Directories
E:\X51v\OpenGLES\mylib
Resolve #using References
Debug Information Format
Program Database (/Zi)
Suppress Startup Banner
Yes (/nologo)
Warning Level
Level 3 (/W3)
Detect 64-bit Portability Issues
No
Treat Warnings As Errors
No
Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])
OK
Cancel
Apply

然后是库文件目录：


http://www.play3d.net
HelloEGL Property Pages
Configuration: Active(Debug)
Platform: Active(Windows Mobile 5.0 Pocket PC SDK (ARMV4I))
Configuration Manager...
General
Debugging
Deployment
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Advanced
Command Line
Output File $(OutDir)/HelloEGL.exe
Show Progress Not Set
Version
Enable Incremental Linking Yes (/INCREMENTAL)
Suppress Startup Banner Yes (/NOLOGO)
Ignore Import Library No
Register Output No
Additional Library Directories E:\X51v\OpenGLES\mylib\lib
Link Library Dependencies Yes
Use Library Dependency Inputs No
Use UNICODE Response Files Yes
Output File
Override the default output file name. (/OUT:[file])
OK
Cancel
Apply

最后是 OpenGL ES 库：



OK，环境准备好了，可以写代码了。先把我们准备好的 OpenGL ES 初始化代码加上：

HelloEGL.cpp

(Global Scope)

```
// HelloEGL.cpp : Defines the entry point for the application.
//
t
#include "stdafx.h"
#include "HelloEGL.h"
#include <windows.h>
#include <commctrl.h>

#include "eglWrapper.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE           g_hInst;            // current instance
HWND                g_hWndMenuBar;      // menu bar handle
EGLWrapper          g_egl;
```

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;
    TCHAR szTitle[MAX_LOADSTRING];      // title bar text
    TCHAR szWindowClass[MAX_LOADSTRING];    // main window class name

    ...
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    g_egl.init(hWnd, 32, true);
```

初始化

```
    return TRUE;
}
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;

    static SHACTIVATEINFO s_sai;

    switch (message)
    {
        case WM_LBUTTONDOWN:
                g_egl.toggleFullScreen(hWnd);
                break;
```

点击切换全屏状态

```
        case WM_PAINT:
            hdc = BeginPaint(hWnd, &ps);

            // TODO: Add any drawing code here...
            draw(hWnd, hdc);
```

绘制

```
            EndPaint(hWnd, &ps);
            break;
```

最后标记出来的那个 draw()函数就是实际绘制 HELLO EGL 的代码了：

```
35 void draw(HWND hWnd, HDC hdc)
36 {
37     if ( ! g_egl.isInitialized() ) return;
38
39     if ( g_egl.makeCurrent() )
40     {
41         glClear(GL_COLOR_BUFFER_BIT);
42
43         glEnableClientState( GL_VERTEX_ARRAY );
44         glEnableClientState( GL_COLOR_ARRAY );
45
46         glVertexPointer( 2, GL_FIXED, 0, helloVertex ); // 2 means (x,y) for one vertex, coord is in float number, stride is 0
47         glColorPointer( 4,  GL_FIXED, 0, helloColor); // 4 means (R,G,B, A) for color of one vertex, color is in float number
48
49         int count = sizeof( helloVertex ) / sizeof(GLfixed) / 2;
50         glDrawArrays(GL_TRIANGLES, 0, count);
51
52         g_egl.swapBuffers();
53     }
54
55     reportError();
56 }
57
```


正如前面提过的，我们只能通过 VERTEX_ARRAY 批量的将模型顶点坐标发送给 OpenGL ES，为此需要先 glEnableClientState()以打开相关支持。为了让世界看上去更美好一点，我还用 ColorArray 给每个顶点指定了颜色。然后通过 glVertexPointer/glColorPointer 将数据发送出去，再通过 glDrawArrays(GL_TRIANGLES)进行绘制。值得一提的是，目前 GLES 不支持 GL_QUADS。 除了调用 glDrawArrays()，OpenGL ES 还提供了更为灵活的 glDrawElements()方法，值得看一下规范里的说明。

剩下的奥密就全在数组 helloVertex 里了。其实它的内容已经完全表示的本文开始的那张示意图里了，建议您自己推算一下，以熟悉 TRIANGLES 建模。这个小小的例子也可以利用 GL_TRIANG_STRIP 得到很大的优化，感兴趣的可以练习一下。

代码如下：

hellomesh.h

```
#ifndef _HELLO_MESH_
#define _HELLO_MESH_

GLfixed helloVertex[] = {       // HELLO EGL
        // H
        Float2Fixed(0.0f), Float2Fixed(0.0f),  Float2Fixed(0.1f), Float2Fixed(0.(
        Float2Fixed(0.0f), Float2Fixed(0.0f),  Float2Fixed(0.1f), Float2Fixed(0.

        Float2Fixed(0.1f), Float2Fixed(0.3f),  Float2Fixed(0.3f), Float2Fixed(0.
        Float2Fixed(0.1f), Float2Fixed(0.3f),  Float2Fixed(0.3f), Float2Fixed(0.

        Float2Fixed(0.3f), Float2Fixed(0.0f),  Float2Fixed(0.4f), Float2Fixed(0.(
        Float2Fixed(0.3f), Float2Fixed(0.0f),  Float2Fixed(0.4f), Float2Fixed(0.

        // E
        Float2Fixed(0.5f), Float2Fixed(0.0f),  Float2Fixed(0.6f), Float2Fixed(0.(
        Float2Fixed(0.5f), Float2Fixed(0.0f),  Float2Fixed(0.6f), Float2Fixed(0.

        Float2Fixed(0.6f), Float2Fixed(0.0f),  Float2Fixed(0.9f), Float2Fixed(0.(
        Float2Fixed(0.6f), Float2Fixed(0.0f),  Float2Fixed(0.9f), Float2Fixed(0.

        Float2Fixed(0.6f), Float2Fixed(0.3f),  Float2Fixed(0.9f), Float2Fixed(0.
        Float2Fixed(0.6f), Float2Fixed(0.3f),  Float2Fixed(0.9f), Float2Fixed(0.

        Float2Fixed(0.6f), Float2Fixed(0.6f),  Float2Fixed(0.9f), Float2Fixed(0.(
        Float2Fixed(0.6f), Float2Fixed(0.6f),  Float2Fixed(0.9f), Float2Fixed(0.

        // L
        Float2Fixed(1.0f), Float2Fixed(0.0f),  Float2Fixed(1.1f), Float2Fixed(0.(
```

可能注意到里面大量调用了宏 Float2Fixed，这是因为 OpenGL ES 1.0 Common Lite Profile 只支持定点数，不能直接使用浮点坐标。所谓定点数这里指的是 16 整数部份+16 位小数部份的 32 位数，也就是 OpenGL ES 预定义的 GLfixed 类型。 将单精度浮点数转成定点数很简单，直接乘个 65536 再强制一下类型就行了，这也就是 Float2Fixed 所做的。

加载模型

如何导出模型，请参考 Maya 部份。现在我已经得到了一个非常简单易用的模型，下面就写一点代码把它加载到内存里，由于模型文件非常简单，所以加载非常简单，而且不浪费内存。 如果你是 3DS Max 用户，那么在 Power VR SDK 里已经带有相当强大的场景导出插件了，请参考相关文档及源码。

以下是头文件

```
 1  //
 2  //  @ Project : MayaMELObjectExport
 3  //  @ File Name : play3dObj.h
 4  //  @ Date : 2007-2-23
 5  //  @ Author : kongfu.yang
 6  //  @ Company : http://www.play3d.net
 7  //  @ Copyright : 2007-2, Example source license.
        By keep this header comment,
        you may use this source file in your project for non-commicial
or commicial purpose.
 8
 9  #ifndef _GL_ES_OBJ_H__
10  #define _GL_ES_OBJ_H__
11
12  #include "gles/gl.h"
13
14  struct play3dObjFixed
15  {
16      int faceCount;
17      int vertexCount;
18      GLfixed *vertexs;
19      GLfixed *normals;
20      GLfixed *uvs;
21      unsigned short *faces;
22
23      void loadFromFile(const char * fname);
24      play3dObjFixed() { vertexs = normals = uvs = 0; faces = 0; }
25      ~play3dObjFixed() { clean(); }
26      void clean(void);
```

```
27  };
28
29
30
31  #endif // _GL_ES_OBJ_H__
```

以下是实现：

```
 1  //
 2  //  @ Project : MayaMELObjectExport
 3  //  @ File Name : play3dObj.h
 4  //  @ Date : 2007-2-23
 5  //  @ Author : kongfu.yang
 6  //  @ Company : http://www.play3d.net
 7  //  @ Copyright : 2007-2, Example source license.
          By keep this header comment,
          you may use this source file in your project for non-commicial
or commicial purpose.
 8
 9  #include "play3dObj.h"
10  #include <stdio.h>
11  #include <stdlib.h>
12
13  void play3dObjFixed::loadFromFile(const char * fname)
14  {
15      FILE * fp = fopen(fname, "rb");
16      fread( &faceCount, sizeof(int), 1, fp );
17      fread( &vertexCount, sizeof(int), 1, fp );
18
19      vertexs = (GLfixed*) malloc( sizeof(GLfixed) * vertexCount * 3 );
20      normals = (GLfixed*) malloc( sizeof(GLfixed) * vertexCount * 3 );
21      uvs = (GLfixed *) malloc( sizeof(GLfixed) * vertexCount * 2 );
22      faces = (unsigned short *) malloc( sizeof(unsigned short) *
faceCount * 3 );
23
24      fread( vertexs, sizeof(GLfixed), vertexCount * 3, fp );
25      fread( normals, sizeof(GLfixed), vertexCount * 3, fp );
26      fread( uvs,     sizeof(GLfixed), vertexCount * 2, fp );
27
28      fread( faces,   sizeof(unsigned short), faceCount * 3, fp );
29
30      fclose(fp);
31  }
32
```

```
33  void play3dObjFixed::clean(void)
34  {
35      if ( vertexs != 0 ) free(vertexs);
36      if ( normals != 0 ) free(normals);
37      if ( uvs != 0 ) free(uvs);
38      if ( faces != 0 ) free(faces);
39      vertexs = normals = uvs = 0; faces = 0;
40  }
```

在适当的位置创建 play3dObjFixed g_model 并调用其 loadFromFile()方法就不罗嗦了。

以下是绘制：

万分抱歉，2007-3-1 之前这部份代码的 53 行少了"* 3"，如果你直接拷过去使用的话，会发现只能显示模型的 1/3 的面。这是因为我先后在家里的两台电脑上改过这个程序，编写本文的台式机上的是比较老的版本，今天在做另一个例子时才发现。看来，再简单的程序，也需要用 CVS 来管理呀。

```
37  void draw(HWND hWnd, HDC hdc)
38  {
39      if ( ! g_egl.isInitialized() ) return;
40
41      if ( g_egl.makeCurrent() )
42      {
43          glClear(GL_COLOR_BUFFER_BIT);
44
45          glEnableClientState( GL_VERTEX_ARRAY );
46          glEnableClientState( GL_NORMAL_ARRAY );
47          // glEnableClientState( GL_UV_ARRAY );
48          // 3 means (x,y,z) for one vertex, coord is in fixed number,
stride is 0
49          glVertexPointer( 3, GL_FIXED, 0, g_model.vertexs );
50          glNormalPointer( GL_FIXED, 0, g_model.normals );
51          // glTexCoordPointer( 3, GL_FIXED, 0, g_model.uvs );
52
53          glDrawElements( GL_TRIANGLES, g_model.faceCount * 3,
GL_UNSIGNED_SHORT, g_model.faces );
54
55          g_egl.swapBuffers();
56      }
57
58      reportError();
59  }
```

这里有三组重要的 API：glEnableClientState，glVertexPointer，glDrawElements。

glEnableClientState 是指示 OpenGL 状态机打开对 Vertex/Normal/UV Array 的支持，使得批量发送模型数据成为可能。

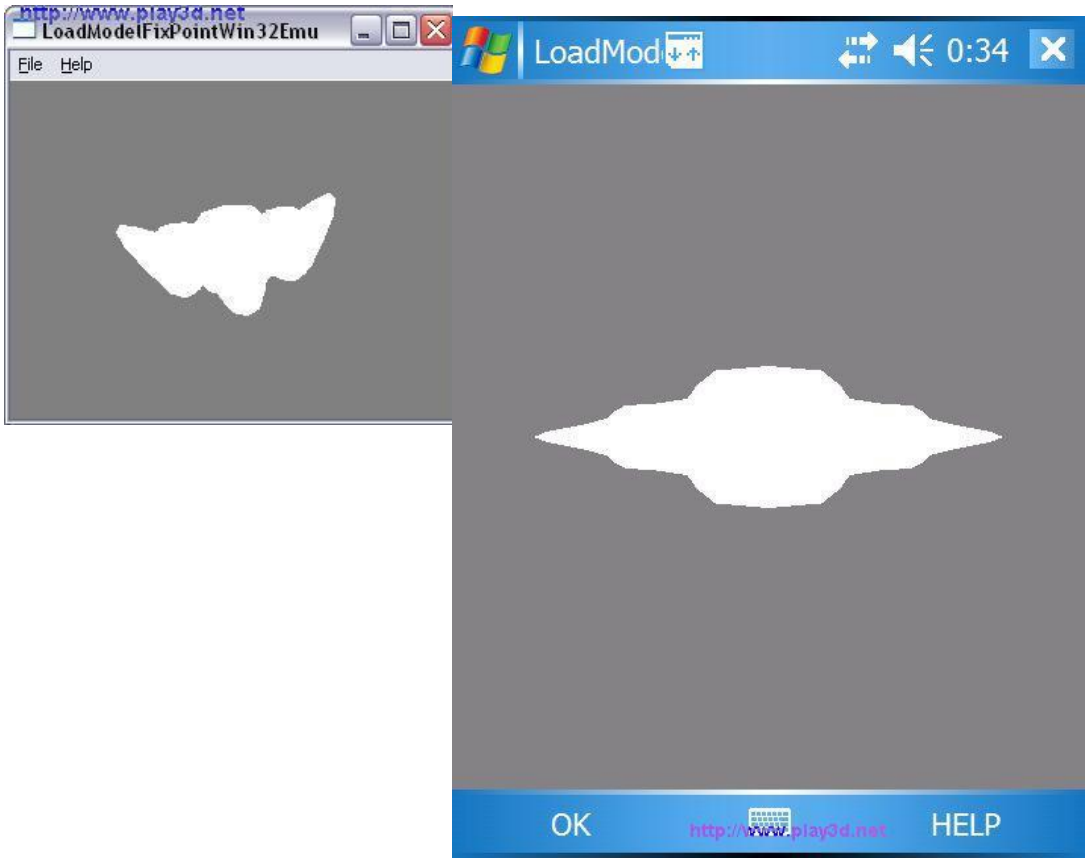
glVertexPointer,glNormalPointer, glTexCoordPoint 等 Pointer 则是实际用来发送模型数据的。

glDrawElements, glDrawArray 则是用利用已发送数据来绘制实际图形的，前者通过额外的顶点索引数组来定义面，后者直接利用已经发送的模型数据数组绘制(在 HelloEGL 例子里用的就是它)，前者显然更加灵活。glDrawElements 第一个参数决定绘制的图元，不过 OpenGL ES 1.x 最多画三角形，四边形和多边形是不支持滴；第二个参数是索引元素的个数，不是面的个数；第三个参数是顶点索引的数据类型，注意 OpenGL ES 1.x 不支持 int/unsigned int；最后是实际的数据。

当前 OpenGL ES 的书籍并不多，好在 OpenGL ES 对 OpenGL 删得多改得少，完全可以把 OpenGL 的书籍拿来参考。以上 API 除了使用定点数之外，其它与 OpenGL 规范中的同名 API 用法完全相同，搞一本 OpenGL 参考手册(Blue Book)到手边是非常有必要的。

绘制效果：

PC Emulator　　Dell Axim X51v Device



材质纹理

材质纹理是增加物体表面细节的有效手段。前面我们已经可以加载任意复杂的三维模型了，但是白乎乎的团，看着一点也不酷，现在是时候让它变漂亮一些了。

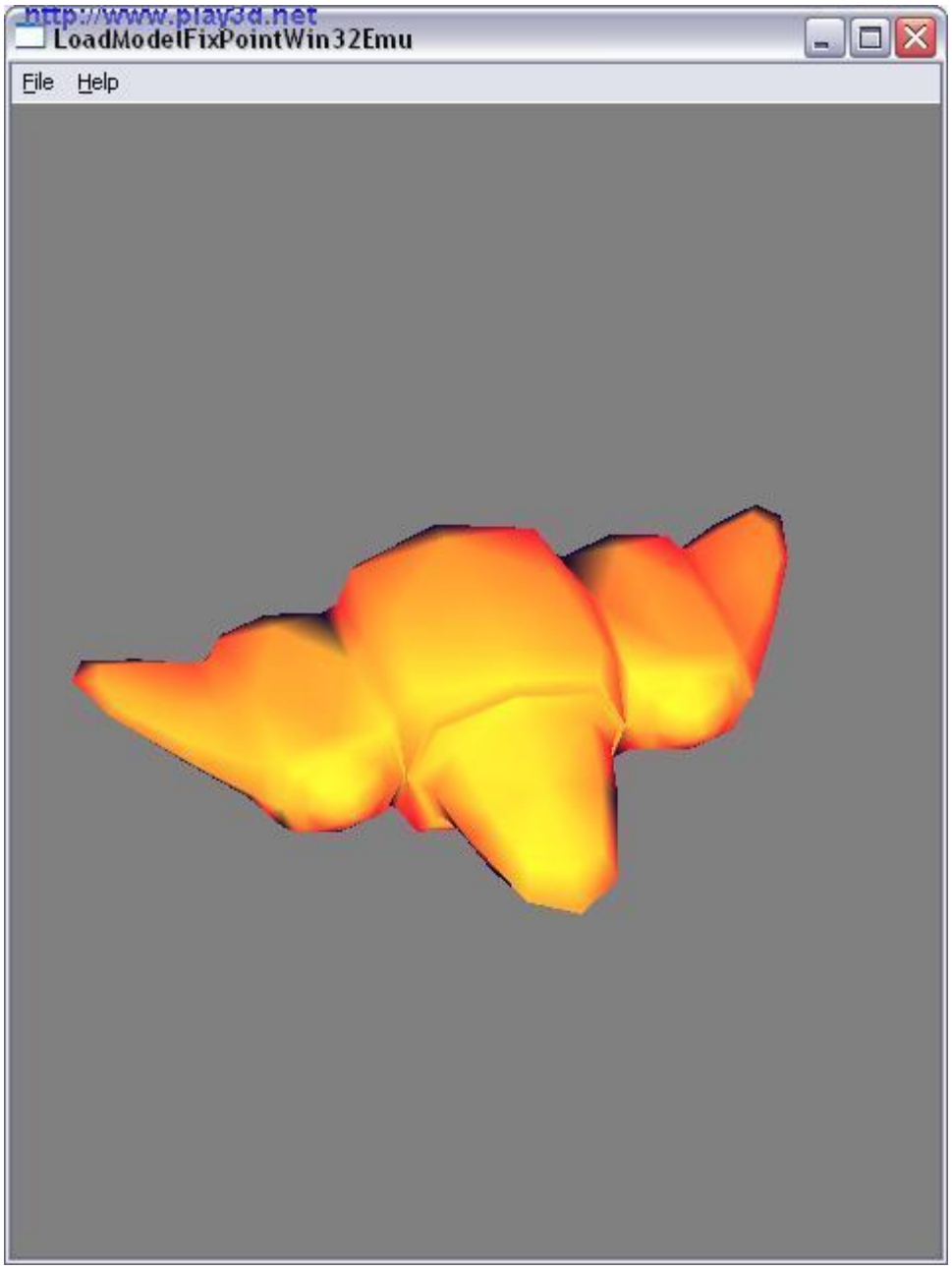
第一步我们给它上点颜色。首先我们需要对光照模型有点概念，物体看上去有颜色，是它被光线照射的结果，如果光是白色的，那么呈现的就是物体本身的颜色，否则会是光色和表面本色综合的结果。物体表面被点光源照射后，会呈现三个区域：高光区、过渡色区和环境色区。高光区是镜面反射的结果，亮度特别高，光源越强越明显；过渡色区是漫反射的结果，通常面积较大；环境色区则是不受光照部份的颜色，通常较暗。

OpenGL ES 提供了 API 供我们为物体表面指定这三个区域的颜色：**void glMaterialx(GLenum face, GLenum pname, GLfixed * param);**

```
GLfixed specmat[4] = { 1<<16, 0, 0, 0 };
GLfixed diffmat[4] = { 0, 1<<16, 0, 0 };
GLfixed ambmat [4] = { 0, 0, 1<<16, 0 };

glMaterialxv( GL_FRONT, GL_SPECULAR, specmat );
```

```
glMaterialxv( GL_FRONT, GL_DIFFUSE, diffmat );
glMaterialxv( GL_FRONT, GL_AMBIENT, ambmat );
... draw code ...
```

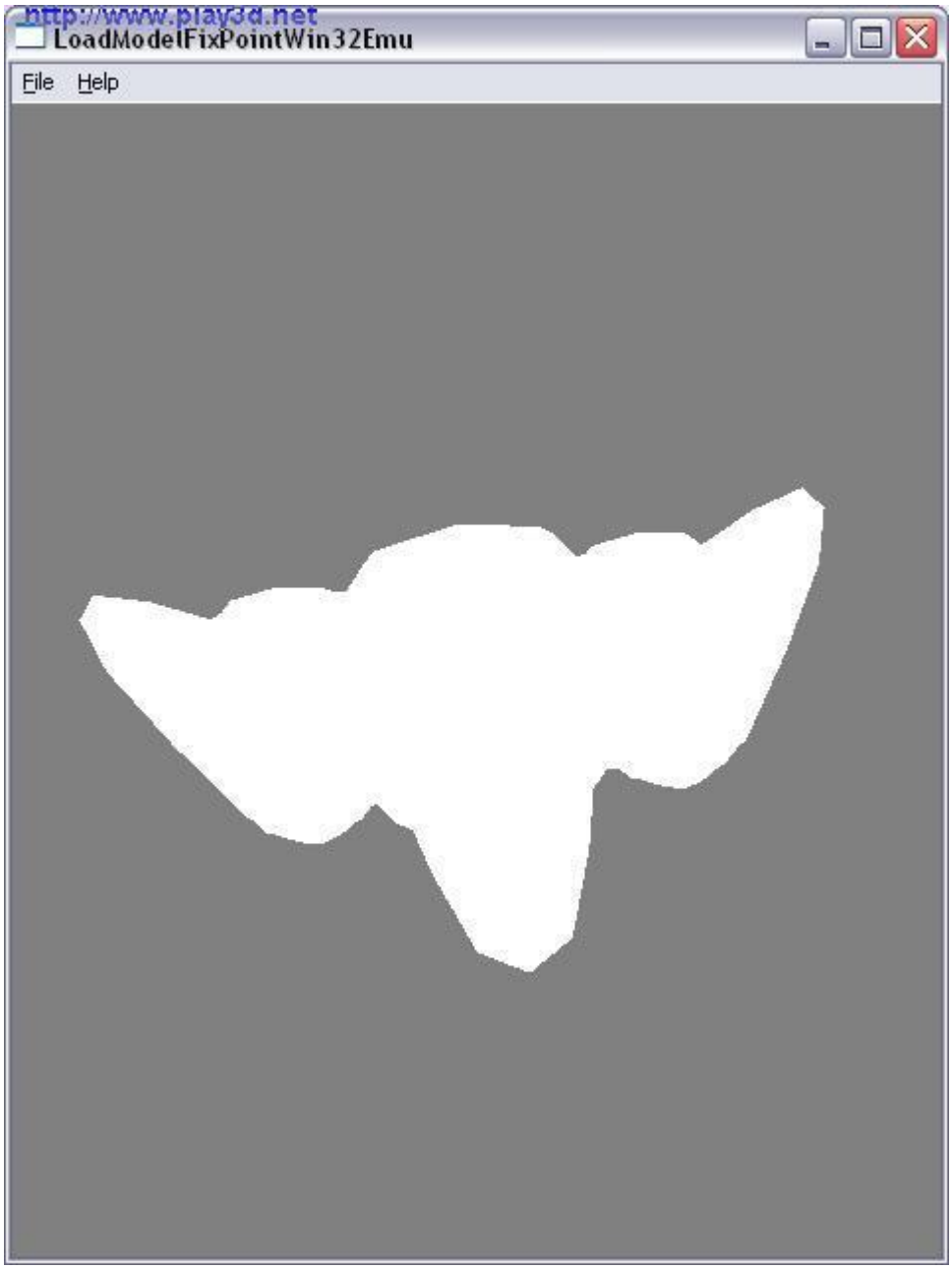
在之前的加载模型示例程序中使用以上材质后的效果：



注意，既然颜色与光照有如此密切的关系，如果不调用以下两行代码：

```
glEnable( GL_LIGHTING );
glEnable( GL_LIGHT0 );
```

以启用光照计算，绘制结果将会是：



如果你看过 Hello EGL 例子，你会发现在不启用光照计算时，也可以直接用 COLOR_ARRAY 来给表面上色，但这并不是一般意义上的材质。

给模型表面上色之后，看上去漂亮多了，不过这样还是不足以表达复杂的表面细节，如果物体表面不是大面积的色块而是复杂的花纹，那就需要通过纹理贴图来表现了。实际上，纹理贴图是游戏类图形程序最重要的工作，针对娱乐市场开发的显示卡其优化重点也是增加显卡总线带宽以便更快速度的将纹图贴图从系统内存传递到显卡处理流水线中以及并行根据纹理计算每个像素点最终颜色。而所谓专业图形卡则更重视于线条等基本图元的发生处理能力。

在 OpenGL ES 中给一个面指定纹理有四步：在内存里准备好纹理数据，调用 glTexImage2D 将纹理数据上传给显卡并设置当前纹理，设置贴图参数，在绘制面时提供纹理坐标(uv)。

OpenGL ES 限制纹理图片的长和宽都必须是 2 的次方，长和宽可以不一样。以下这段代码在生成一个 128x128 的图片并保存到数组 embText 里：

```
1       unsigned short embTex[128*128];
2       unsigned short color[4]={ 0xF800, 0x7E0, 0x1F, 0xffff };
3       for ( int i = 0; i < 128; ++i )
4       {
5           for ( int j = 0; j < 128; ++j )
6           {
7               embTex[i*128+j] = color[(i/32 + j/32)%4];
8           }
9       }
10
```

在实际应用中，纹理图片通常需要从特写格式的文件中读取，像 tga 之类的文件格式比较简单，而 jpg/gif 则相当复杂但压缩率高，多数游戏会采用自定义格式的文件，这样不仅可以按需要对格式进行优化，也可以提供简单的资源保护。

纹理数据放进内存之后，就可以调用 glTexImage2D 将其上传到显存里并通知流水线将其作为“当前”纹理：

```
11 glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, 128, 128 , 0, GL_RGB,
GL_UNSIGNED_SHORT_5_6_5, embTex);
```

参数的意义是相当直接的：

1. target: GL_TEXTURE_2D 表示定义一个二维纹理，实际上这是目前 OpenGL ES 支持的唯一一个值；
2. level: LOD 级数，每一级 LOD 纹理是上级纹理长宽各缩小一半的图片。如果使用这个纹理的面由于距离或角度，实际可见面积很小，流水线会选用适当级别的纹理来绘制它，这样可以避免实时缩小图片而需要的大量计算。
3. internalFormat：内部格式，指定纹理中的颜色组分数量，用象 GL_RGB 这样的符号常量。可以想象 OpenGL ES 只支持几种最常用的格式。简单而言之，这个参数定义纹理在**显存**中的格式。后面还有一个 format，那是说明**内存**中的格式的。
4. width, height：无边界时必须是 2^n，有边界时必须是 2^n+2
5. border：边界宽度，必须是 0 或者 1。使用带边界的纹理对解决拼接时的缝隙是很有帮助的。不过 OpenGL ES 1.0 不支持

    1. OpenGL 支持的纹理大小是非常有限的，你很可能要用多个纹理拼成一个很大图案，由于 Linear 之类的贴图算法需要将拼缝上的像素与相邻像素平均，将被切到下一个贴图上的像素放到边界里，可以保证获得的效果和整张图时是一样的。否则，可能会出现一条明显的拼缝。
6. format：待上传的像素数据格式，GL_RGB 等都可以故名思义的。同样，OpenGL ES 1.0 只支持最常用的 RGB,RGBA,LUMINANCE_ALPHA,ALPHA。
7. type：待上传像素的数据类型。GL_UNSIGNED_BYTE 表示纹理是一组 8bit 字节序列； GL_UNSIGNED_SHORT_5_6_5 表示是 16bit 字序列，其中头 5 位记录 Red 分量，中 6 位记录 Green 分量，末 5 位记录 Blue 分量。
8. pixel：实际的纹理数据。

这里有几点值得关注一下：

1. embTex 在这个命令执行完之后就没用了，如果是动态分配的内存，可以 free/delete 之。数据已经上传到显存里了。
2. 流水线只会有一个“**当前**”纹理，如果在绘制之前调用多个 glTexImage3D，只有最后一个起作用。
3. glTexImage 如果一个纹理反复被用到，那么可以调用 glGenTextures(count, idArray)/glBindTexture(GL_TEXTURE_2D, id)为其指定一个 ID，需要时 glActiveTexture(id)就可再次将其设为“**当前**”纹理。这样可以省掉重复上传纹理纹理数据的开销。

控制贴图过程的参数主要有：

1. glEnable(GL_TEXTURE_2D)：如果 disable，那么三维图形流水线会完全忽略纹理贴图，默认是 disable 的。
2. glTexParameterx(GL_TEXTURE_2D, param, value)：当面生成的光栅与纹理像素点不能一一对应时如何处理。
    - GL_TEXTURE_WRAP_T/S：在纵横方向上大小不一致时，如何调整纹理坐标(uv)。
        - 默认的 GL_REPEAT 为重复使用纹理，
        - GL_CLAMP 为将 uv 坐标收缩到[0,1.0]之内，
        - GL_CLAMP_TO_EDGE 为将 uv 坐标收缩到[1/(2*size), 1-1/(2*size)]之间。
        - 只有 GL_CLAMP 时边界上的纹理才可以访问。
    - GL_TEXTURE_MIN_FILTER：纹理缩小算法。当面所生成的像素在纹理化过程中，映射到纹理上的大于一个像素的区域时，该参数选择如何合并多个纹理像素产生最终应用到面上的颜色。GL_NEAREST 表示用离得最近的那个像素，GL_LINEAR 表示周围最近四个纹理像素加权平均，GL_*A*_MIPMAP_*B*表示根据*B*选择适当的 MIPMAP 级，然后再用*A*选择最近的或计算。
    - GL_TEXTURE_MAG_FILTER：纹理放大算法。与 MIN 相反的情况。只有 GL_NEAREST/GL_LINEAR 两个选择。
3. glTexEnvx：这个比较复杂，见手册。

最常见的做法是这样的：

```
        glEnable(GL_TEXTURE_2D);
        glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
        glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
        glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
        glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```

现在万事俱备，就差画画了。由于 OpenGL ES 只支持 vetex array 方式发送模型数据，这一步只有一件事要做，给模型加上 uv 坐标：

```
glEnableClientState( GL_TEXTURE_COORD_ARRAY );
glTexCoordPointer( components, GL_FIXED, stride, uvPointer );
```

注意：纹理坐标(uv)与顶点坐标系的习惯略有不同，其原点在图片的左下角，左至右为正 u 方向，底至顶为正 v 方向。

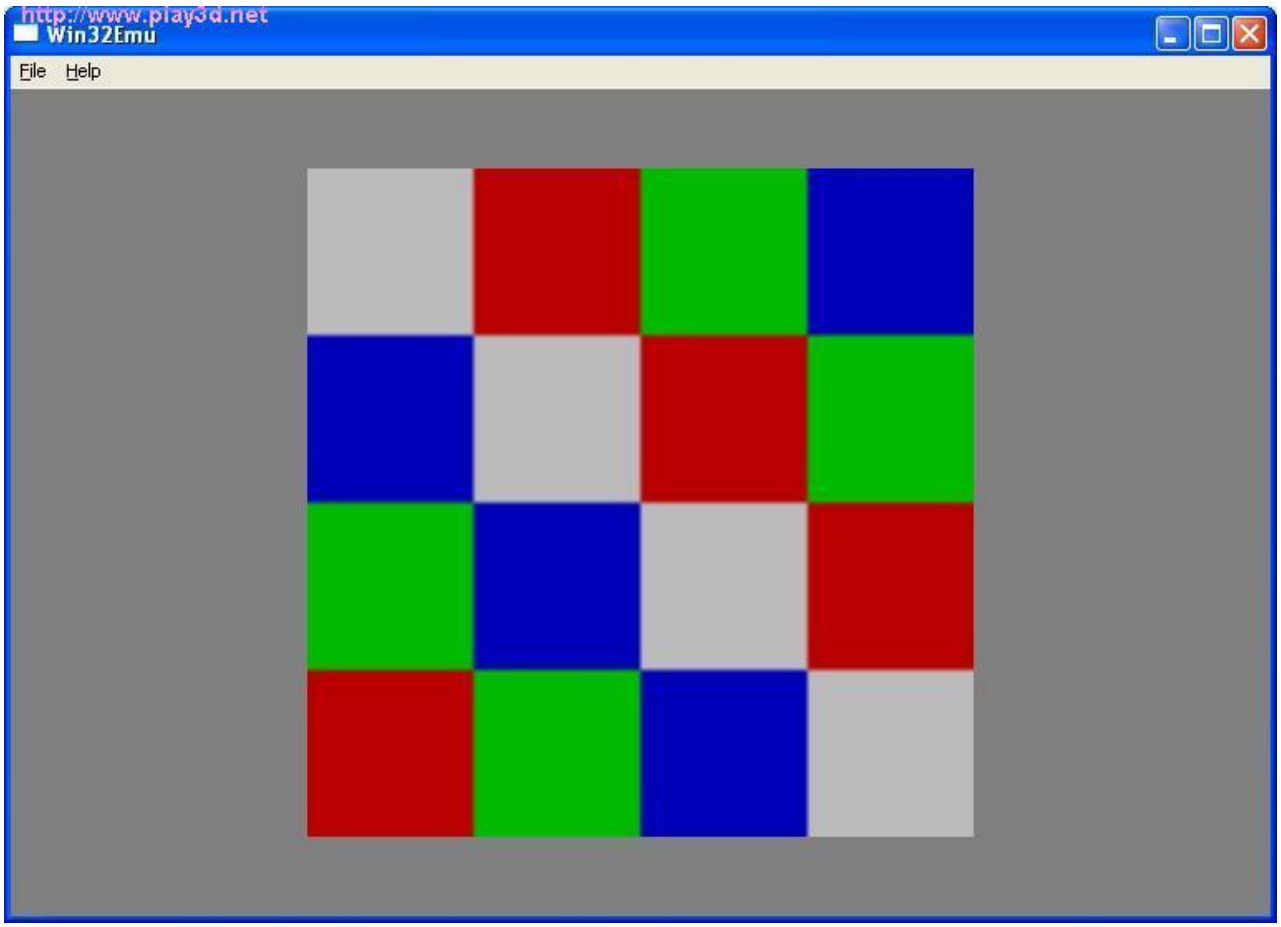
完整形式的纹理坐标可以表示为(s,t,r,q)，其中(s,t)对应一般三维建模软件中的 uv 也就是平面纹理图片的(x,y)；r 在使用三维纹理时使用，OpenGL ES 目前明确不支持三维纹理，应设为 0；q 为齐次坐标，通常为 1，在坐标变换计算过程中可能使用非 1 的值。

以下是一个完整的例子：

```
1  void testTexture()
2  {
3      GLfixed rect[] = { Float2Fixed(-1.0f), Float2Fixed(-1.0),0,
4                     Float2Fixed(1.0f), Float2Fixed(-1.0), 0,
5                     Float2Fixed(1.0f), Float2Fixed(1.0), 0,
6
7                     Float2Fixed(-1.0f), Float2Fixed(-1.0),0,
8                     Float2Fixed(1.0f), Float2Fixed(1.0), 0,
9                     Float2Fixed(-1.0f), Float2Fixed(1.0f) , 0
10                   };
11     GLfixed rnormal[] = {
12                       0, Float2Fixed(1.0f), 0,
13                       0, Float2Fixed(1.0f), 0,
14                       0, Float2Fixed(1.0f), 0,
15
16                       0, Float2Fixed(1.0f), 0,
17                       0, Float2Fixed(1.0f), 0,
18                       0, Float2Fixed(1.0f), 0 };
19     GLfixed rectuv[] = { Float2Fixed(0.0f), Float2Fixed(0.0),
20                        Float2Fixed(1.0f), Float2Fixed(0.0),
```

```
                        Float2Fixed(1.0f), Float2Fixed(1.0),

                        Float2Fixed(0.0f), Float2Fixed(0.0),
                        Float2Fixed(1.0f), Float2Fixed(1.0),
                        Float2Fixed(0.0f), Float2Fixed(1.0f) };

    unsigned short color[4]={ 0xF800, 0x7E0, 0x1F, 0xffff };

    for ( int i = 0; i < 128; ++i )
    {
        for ( int j = 0; j < 128; ++j )
        {
            embTex[i*128+j] = color[(i/32 + j/32)%4];
        }
    }

    glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
    glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);

    glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB,  128, 128 , 0, GL_RGB,
GL_UNSIGNED_SHORT_5_6_5, embTex);

    glEnableClientState( GL_VERTEX_ARRAY );
    glEnableClientState( GL_NORMAL_ARRAY );
    glEnableClientState( GL_TEXTURE_COORD_ARRAY );
    glVertexPointer( 3, GL_FIXED, 0, rect );
    glNormalPointer( GL_FIXED, 0, rnormal);
    glTexCoordPointer( 2, GL_FIXED, 0, rectuv );

    glDrawArrays( GL_TRIANGLES, 0, 6);
}
```

运行效果如下图所示：



从上图中不难看出，左上第一格是白色而不是纹理图片左上第一格的红色，如果将纹理坐标中 v 的 0.0,1.0 互换一下，就完全一样了。这是因为 OpenGL ES 规定二维纹理图片数据是从下到上逐行存放的，而示例程序中生成的数据是自上而下逐行存放的。

这里介绍的只是表面材质纹理效果的基础知识，目前材质与贴图是实时应用中获得高真实感效果的主要手段，这部份还有大量更深入的技术和运用方式值得研究，例如多重纹理、多遍纹理、材质效果合并、BUMP-MAPPING、环境贴图等等。

光照

前面已经提到过没有光就看不见东西，无论是顶点颜色、表面材质还是纹理贴图，只有打开了光照才能看得见。

还记得下面这两行代码吗？

```
glEnable( GL_LIGHTING );
glEnable( GL_LIGHT0 );
```

第一行是在说“要有光”，于是世界就有光了。第二行是在说打开 0 号光源，于是世界就被在默认光源(0, 0, 1.0, 0)——照向-z 轴方向的平行光——白色光照亮了。

在 CG 世界里，我们把光照总结成几种常用模型：环境光、平行光、点光源、聚光灯。环境光是无所不在的漫射光，是场景的背景亮度。

在 OpenGL ES 里，没法直接定义一个无确定方向和位置的环境光源，所以忘掉在某些 3D 制作软件或开发工具里放置一个环境光源的经历吧。OpenGL ES 的环境光是光源本身的一个属性。

平行光是指有方向但没有具体位置的光源，比如实际生活中的日光。这种光源也被称为方向光或平面光。在 OpenGL ES 中可以将光源位置(x,y,z,w)中的 w 置为 0 来定义平行光源。

点光源是指有具体位置，向所有方向均匀发射光线的光源，就象实际生活中的灯泡。在 OpenGL ES 中的光源默认就是点光源。

聚光灯是受限的点光源，只在照射方向某个角度范围之内投射光线，并且光照的强度可能与夹角有关系。在 OpenGL ES 中可以通过 GL_SOPT_DIRECTION, GL_SPOT_CUTOFF, GL_SPOT_EXPONENT 等参数来定义一个聚光灯。

现在再来看一下 OpenGL ES 里定义一个光源都有哪些参数可用：

光源颜色：GL_AMBIENT 环境光 RGBA 强度, GL_DIFFUSE 散射光 RGBA 强度, GL_SPECULAR 高光(镜面反射区)RGBA 强度
光源位置：GL_POSITION (x,y,z,w)齐次坐标，w 为 0 时为平行光，此时光照计算仅依赖于光源方向而与其位置无关，并且光线不进行衰减处理。
聚光性：GL_SPOT_DIRECTION 聚光灯光照方向，GL_SPOT_CUTOFF 最大发散角 [0,90] 与方向夹角超过此范围的区域不会被光照到。

光照计算:
OpenGL ES 里光照的效果最终表现为物体表面各点的颜色，它必然要和物体表面的材质纹理合并起来。而在同一个场景里可能有多个光源，这些光源的照明效果也需要合并。

- 合并光照效果：
  - C = 材质的自发光($emissive_{mat}$) + 环境光强($ambient_{mat}$ * $ambient_{scene}$) + Σ 反射光
  - 各光源的反射光强度计算：
    - 传播衰减系数 * 聚光衰减系数 * (散射光 + 漫射光 + 高光)。
      - 传播衰减系数表示光源与物体间距 d 对光强的影响，OpenGL ES 实际上可指定常量衰减因子 k0, 线性衰减因子 k1, 平方衰减因子 k2，三者合起来($k0 + k1*d + k2*d^2$)才是总的传播衰减系数。
      - 聚光衰减因子表达的是聚光灯的聚光特性——光锥的中心最亮，越往边缘越暗，与光线偏离中心线的角度有关。

        - 散射光 = $ambient_{mat} * ambient_{light}$
        - 漫射光 = $diffuse_{mat} * diffuse_{light} * (normal \langle dot \rangle Vector_{vertex\_to\_positionlight}))$
        - 高光 = $(normal \langle dot \rangle halfVector)^{specExp_{mat}} * specular_{mat} * specular_{light}$
            - $* halfVector = Vector_{light\_to\_vertex} + Vector_{vertex\_to\_eye}$
- * 高光部分的计算公式表明，OpenGL ES 使用的是 Blinn-Phong 光照模型。如果需要使用其它模型，可以通过编程实现，不过在没有 Shader 支持的情况下，就利用不上硬件加速。OpenGL ES 2.0 定义了可编程管线，可以编写 Shader，不过我还没有见过实际支持的硬件，只有某些软件库支持。

在这个光照计算公式下，各种光源的控制如下：

- 平行光/方向光：
    - 这种光源位置的 w=0，表示从无限远处照过来，这种光是不衰减的，即物体顶点与光源之间的距离差别忽略不计。这种光源的反射光计算非常简单，光源的照射方向记为 $halfVector_{light}$，则光照效果为：
      $ambient \mathrel{+}= ambient_{light}$
      $diffuse \mathrel{+}= diffuse_{light} * (normal \langle dot \rangle position_{light})$
      $specular \mathrel{+}= specular_{light} * (normal \langle dot \rangle halfVector_{light})^{shiness_{material\_front}}$
    - 建立一个这样光源 LIGHTi 的命令序列为：
      glEnable(GL_LIGHTi)
      glLight
- 点光源
- 聚光灯

Light Lab 这是一个挺不错的光照试验程序，可以在 PC 上用来帮助理解 OpenGL 光照模型。作者只以源码的形式发行这个工具，在 windows 上可能需要借助 cygwin 编译，或者有耐心的自己拿源码做个 VC 项目亦可。

压缩纹理

纹理图片数据量巨大，一个 256x256 RGBA16 的原始象素就有 128KB 之多，而一个实际应用中会用到巨量的纹理图片。这些数据都需要从硬盘或存储卡上读进内存，然后再上传到显存里。通过有损压缩，图片的文件尺寸可以急剧减小，512x512 的真彩 BMP 有 768KB，转成 100%质量的 JPG 就只有 235KB。

不过象 JPG 这种常见图象压缩格式对于多数应用的内存占用和显示总线带宽占用并没有直接的好处，因为还得解压缩成原始象素再传给显卡，而且还有加载时的解码计算负担。这是因为显卡的纹理解码硬件不理解 JPG 格式。所以，在没有显卡硬件支持的情况下，用压缩格式保存纹理没什么意义，特别是对于手持移动设备来说，解码象 JPG 这种复杂格式是很浪费电的。

考虑到现代游戏对纹理图片的严重依赖，及相应的对视频总线的巨大压力，硬件实时解压缩获得了广泛的支持，不过这个还没有一种格式获得多个厂家的支持。在 OpenGL ES 里已经为此定义了一个标准接口 glCompressedTexImage2D(..., format, ..., data)，不过纹理数据的格式则没有标准，要参考厂商的 SDK 或文档获得 format 值。这也就意味着，使用了压缩纹理之后就不能跨平台了。对于 OpenGL ES 应用来说可能问题不大，可以用不压缩纹理做为起点，然后针对不同硬件平台转换相应的优化版本。而对于标准 OpenGL 应用来说问题可能也不太大，因为真正值得关注的也就 ATI vs. NVIDIA 吧。

纹理压缩算法除了要求高压缩比、低失真之外，可随机读取任意一点及便于低成本硬件实现是最大的挑战。2700G 支持的 PowerVR Texture Compression，就是一种按照这种需求开发的高效率压缩算法。对于 PVRTC 算法及 Texture Compression 算法的演化，可以参考 Simon Fenney 题为"Texture Compression using Low-Frequency Signal Modulation"的论文，可以在 PowerVR 的 GLES PCE SDK 里找到。

PVRTC 有两种压缩算法，512x512 RGB 的原始象素 768KB，PVRTC 4BPP 压缩后为 128KB，PVRTC 2BPP 压缩后为 64K。不过 PVRTC 2BPP 效果看上去明显比较差。

值得一提的是，Intel 2700G SDK 里带有一个 PVRTextureTool.exe 可将 BMP 压缩成 PVRTC 格式，这个 tool 版本比较低是 1.x 的。在 PowerVR SDK 里有更新版本(3.0)，新版本明显好用得多，可以处理 alpha channel，支持 OpenGL 纹理格式等等。这个 tool 有命令行版本，也有 GUI 版本，而且还有一个库封装了算法和文件格式处理代码可用来制作自己的工具。PVRTC 的纹理存储顺序是从上到下逐行保存的，而 OpenGL ES 定义纹理数据是从下到上逐行排列的，所以使用 3.0 版本并且在编码时勾上 OpenGL 选项是非常必要的。

根据实测，PVR PCE 不支持带 MipMap 的 PVRTC 数据。可能设计者认为对于现有的 240x320 或 480x640 的屏幕来说，MipMap 实在是意义不大吧。终于知道为什么 PCE 上跑得好好的程序在 X51v 上却显示不出纹理来了，原来 WinCE 是没有当前目录这个说法的，fopen ("t.pvr")会去"My Devices"目录里找这个文件，显然是找不到，把相应的.pvr 文件放进去就好了。

2700G 或 PVR PCE SDK 里都带用 glext.h，里面定义了所需要的常量，主要有用的是 GL_COMPRESSED_RGB_PVRTC_4BPPV1_IMG 等传给 glCompressedTexImage2D()的 format 常量。PVRTextureTool 生成的文件是.pvr 格式的，其格式在 SDK 文档里有详细的描述，甚至连 struct 都定义好了。

总的说来利用上压缩纹理还是相当简单而直接的，所以闲话少说，看代码吧：主程序部份，提示一下，可以用't'轮换纹理图片，用'm'轮换贴图公式，其它也就是画了个带纹理的矩形而矣。主要的代码在 PVRTexture.upload()里。

```
1 //
2 //  @ Project : PVRTC Example
```

```
  3 //  @ File Name : texturepvr.cpp
  4 //  @ Date : 2007-8-30
  5 //  @ Author : kongfu.yang
  6 //  @ Company : http://www.play3d.net
  7 //  @ Copyright : 2007-8, Example source license. By keep this header
comment,

you may use this source file in your project for non-commicial or commicial
purpose.
  8
  9 #include "main.h"
 10 #include "PVRTexture.h"
 11
 12 EGLWrapper      * egl;
 13 PVRTexture  * g_tex = 0;
 14
 15 bool g_wireframe;
 16 GLfixed g_rx, g_ry;
 17 GLfixed zoom = Float2Fixed(1.0);
 18
 19 #define TEXTURE_COUNT 9
 20 char * fileNames[TEXTURE_COUNT] = {
 21     "yuquan4.pvr", "yuquan4mip.pvr", "yuquan4a.pvr",
"yuquan4amip.pvr",
 22     "yuquan4gl.pvr", "yuquan4mipgl.pvr", "yuquan4agl.pvr",
"yuquan4amipgl.pvr",
 23     "yuquan4mip_intel.pvr"
 24 };
 25
 26 #define MAX_TEX_MODE 5
 27 int modeIdx = 0;
 28 GLfixed texModes[MAX_TEX_MODE] = {
 29     GL_REPLACE, GL_MODULATE, GL_DECAL, GL_BLEND, GL_ADD
 30 };
 31 LPTSTR texModeNames[MAX_TEX_MODE] = {
 32     L"GL_REPLACE", L"GL_MODULATE", L"GL_DECAL", L"GL_BLEND",
L"GL_ADD"
 33 };
 34
 35 void nextMode(void)
 36 {
 37     if ( ++modeIdx  >= MAX_TEX_MODE ) modeIdx = 0;
 38     glTexEnvx(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
texModes[modeIdx]);
```

```
39 }
40
41 void nextTexture(void)
42 {
43     static int idx = 3;
44     if ( g_tex != 0 ) delete g_tex;
45     g_tex = new PVRTexture(fileNames[idx++]);
46     g_tex->upload();
47
48     if ( idx >= TEXTURE_COUNT ) idx = 0;
49
50     glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
51     glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
52     glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
53     glTexParameterx(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);
54
55     glTexEnvx(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
texModes[modeIdx]);
56 }
57
58 void init(HWND hWnd, EGLWrapper *g_egl)
59 {
60     egl = g_egl;
61     egl->init(hWnd, 16, true);
62
63     glDisable(GL_LIGHTING);
64
65     glEnable(GL_TEXTURE_2D);
66     glEnable(GL_BLEND);
67
68     nextTexture();
69
70     glClearColorx(0, 0, Float2Fixed(1.0f), Float2Fixed(0.5f));
71     glColor4x(Float2Fixed(1.0f), 0, 0, Float2Fixed(0.5f));
72 }
73
74 GLfixed rect[] = { Float2Fixed(-1.0f), Float2Fixed(-1.0f),0,
75                    Float2Fixed(1.0f), Float2Fixed(-1.0f), 0,
76                    Float2Fixed(1.0f), Float2Fixed(1.0f), 0,
77
```

```
78                    Float2Fixed(-1.0f), Float2Fixed(-1.0f),0,
79                    Float2Fixed(1.0f), Float2Fixed(1.0f), 0,
80                    Float2Fixed(-1.0f), Float2Fixed(1.0f) , 0
81                   };
82 GLfixed rnormal[] = {
83                      0, Float2Fixed(1.0f), 0,
84                      0, Float2Fixed(1.0f), 0,
85                      0, Float2Fixed(1.0f), 0,
86
87                      0, Float2Fixed(1.0f), 0,
88                      0, Float2Fixed(1.0f), 0,
89                      0, Float2Fixed(1.0f), 0 };
90 GLfixed rectuv[] = {
91                       Float2Fixed(0.0f), Float2Fixed(0.0f),
92                       Float2Fixed(1.0f), Float2Fixed(0.0f),
93                       Float2Fixed(1.0f), Float2Fixed(1.0f),
94
95                       Float2Fixed(0.0f), Float2Fixed(0.0f),
96                       Float2Fixed(1.0f), Float2Fixed(1.0f),
97                       Float2Fixed(0.0f), Float2Fixed(1.0f)
98                     };
99
100 void testPVRTC()
101 {
102     //glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB,  128, 128 , 0, GL_RGB,
GL_UNSIGNED_SHORT_5_6_5, embTex);
103         glEnableClientState( GL_VERTEX_ARRAY );
104         //glEnableClientState( GL_NORMAL_ARRAY );
105         glEnableClientState( GL_TEXTURE_COORD_ARRAY );
106         glVertexPointer( 3, GL_FIXED, 0, rect );
107         //glNormalPointer( GL_FIXED, 0, rnormal);
108         glTexCoordPointer( 2, GL_FIXED, 0, rectuv );
109
110         glActiveTexture( g_tex->id );
111         glDrawArrays( GL_TRIANGLES, 0, 6);
112 }
113
114 void draw(HWND hWnd, HDC hdc)
115 {
116     if ( ! egl->isInitialized() ) return;
117
118     if ( egl->makeCurrent() )
119     {
120     glEnable(GL_DEPTH_TEST);
```

```
121     glDepthFunc(GL_LEQUAL);
122 
123     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
124         glPushMatrix();
125 
126         //glRotatex( g_rx, Float2Fixed(1.0f), 0, 0 );
127         //glRotatex( g_ry, 0, Float2Fixed(1.0f), 0 );
128 
129         glScalex( zoom, zoom, Float2Fixed(1.0f) );
130 
131         // draw here
132         testPVRTC();
133 
134         glPopMatrix();
135         egl->swapBuffers();
136     }
137 
138     reportError();
139 }
140 
141 bool msgHandler(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
142 {
143     TCHAR wbuf[1024];
144 
145     switch (message)
146     {
147     case WM_CHAR:
148         switch(wParam){
149         case 'f':
150         case 'F':
151             g_wireframe = ! g_wireframe;
152             InvalidateRect(hWnd, 0, false);
153             break;
154         case 't':
155             nextTexture();
156             ZeroMemory( wbuf, 1024 );
157             MultiByteToWideChar(CP_ACP, 0, g_tex->fileName,
strlen(g_tex->fileName),  wbuf, 1024);
158             SetWindowText(hWnd, wbuf);
159             InvalidateRect(hWnd, 0, false);
160             break;
161         case 'm':
162             nextMode();
163             SetWindowText(hWnd, texModeNames[modeIdx]);
```

```
164             InvalidateRect(hWnd, 0, false);
165             break;
166         case '+':
167             // zoom in to exam mipmap
168             zoom += Float2Fixed(0.1);
169             InvalidateRect(hWnd, 0, false);
170             break;
171         case '-':
172             // zoom out to exam mipmap
173             zoom -= Float2Fixed(0.1);
174             InvalidateRect(hWnd, 0, false);
175             break;
176         default:
177             ;
178         }
179         break;
180     case WM_KEYDOWN:
181         switch(wParam)
182         {
183         case VK_UP:
184             g_rx = Float2Fixed(Fixed2Float(g_rx) + 10.0f);
185             InvalidateRect( hWnd, 0, false );
186             break;
187         case VK_DOWN:
188             g_rx = Float2Fixed(Fixed2Float(g_rx) - 10.0f);
189             InvalidateRect( hWnd, 0, false );
190             break;
191         case VK_LEFT:
192             g_ry = Float2Fixed(Fixed2Float(g_ry) + 10.0f);
193             InvalidateRect( hWnd, 0, false );
194             break;
195         case VK_RIGHT:
196             g_ry = Float2Fixed(Fixed2Float(g_ry) - 10.0f);
197             InvalidateRect( hWnd, 0, false );
198             break;
199         default:
200             ;
201         }
202         break;
203     default:
204         ;
205     }
206
```

```
207     return false;
208 }
```

我已经写了一个非常简单的类来封装 PVRTC，包括读取文件，分析格式，定义成 OpenGL ES 纹理等。如下：

```
 1 //
 2 //  @ Project : PVRTC Wrapper
 3 //  @ File Name : PVRTexture.h
 4 //  @ Date : 2007-8-30
 5 //  @ Author : kongfu.yang
 6 //  @ Company : http://www.play3d.net
 7 //  @ Copyright : 2007-8, Example source license.

By keep this header comment, you may use this source file in your project
for non-commicial or commicial purpose.
 8
 9 #ifndef PVR_TEXTURE_H
10 #define PVR_TEXTURE_H
11
12 #include "gles/gl.h"
13 #include "gles/glext.h"
14 #include "string.h"
15 #include "PVRTCFile.h"
16
17 extern void reportError(void);
18
19 class PVRTexture
20 {
21     public:
22         PVRTexture(char  * fname){ fileName = _strdup(fname); id=0;
data=0; size=0; }
23         ~PVRTexture(){ free(fileName); if ( data != 0 ) free(data); }
24
25     public:
26         char * fileName;
27         unsigned int id;
28
29         void upload(void)
30         {
31             load();
32             glGenTextures(1, &id);
33             glBindTexture(GL_TEXTURE_2D, id);
34             glCompressedTexImage2D(GL_TEXTURE_2D, levels, format,
width, height, 0, size, data);
35             reportError();
```

```
36          }
37
38      protected:
39          int format; // .pvr format map to gles format
40          int width;
41          int height;
42          int levels; // mipmap levels
43
44          long size;
45          unsigned char * data;
46
47          void load(void)
48          {
49              FILE *fp = fopen(fileName, "rb");
50              PVR_Header h;
```

.pvr 文件头部是可变长度的，其头部长度由第一个 DWORD 记录，这个长度已经将该 DWORD 自己计算在内了：

```
51              fread(& h.dwHeaderSize, sizeof(DWORD), 1, fp);
```

读取头部的剩余部分：

```
52              fread(& h.dwHeight, h.dwHeaderSize - sizeof(DWORD), 1,
```

fp);

```
53
54              // format info
55              width = h.dwWidth;
56              height = h.dwHeight;
57              levels = h.dwMipMapCount;
58              size = h.dwTextureDataSize;
59
```

Flags 由两部份组成，0x100 以上是位掩码，记录制作时的选项，低位部分是格式代号：

```
60              h.dwpfFlags &= ~PVRTC_MIPMap; // redundant with levels?
61              h.dwpfFlags &= ~PVRTC_Twiddle; // todo : record?
62              h.dwpfFlags &= ~PVRTC_NormalMap; // todo : handle?
63
```

头部有一个 dwAlphaBitMask 字段，用来记录图片每个单元里有几个位是用来记录 alpha 信息的。

PVRTextureTool 1.x 生成的纹理似乎总是不含 alpha 通道的，而 3.x 则可以控制是否生成。

目前如果有 alpha 通道，这个 alpha bit 总是 1。

```
64              bool hasAlpha = h.dwAlphaBitMask != 0;
65              switch( h.dwpfFlags )
66              {
```

PVRTC4 是用 PVRTC 4BPP 算法压缩的：

```
67                  case PVRTC_PVRTC4:
```

如果纹理数据中有 alpha 通道，那就映射到 GL_RGBA 格式，否则映射到 GL_RGB 格式。

```
68                      format = hasAlpha?
GL_COMPRESSED_RGBA_PVRTC_4BPPV1_IMG :
GL_COMPRESSED_RGB_PVRTC_4BPPV1_IMG;
69                      //if the texture contains alpha information, then
use it.
70                      break;
```

PVRTC2 是用 PVRTC 2BPP 算法压缩的，压缩比更高，失真也更厉害：

```
71                  case PVRTC_PVRTC2:
```

如果纹理数据中有 alpha 通道，那就映射到 GL_RGBA 格式，否则映射到 GL_RGB 格式。

```
72                      format = hasAlpha?
GL_COMPRESSED_RGBA_PVRTC_2BPPV1_IMG :
GL_COMPRESSED_RGB_PVRTC_2BPPV1_IMG;
73                      //if the texture contains alpha information, then
use it.
74                      break;
75                  // _GL format: the order of scan lines are reverted.
76                  case PVRTC_PVRTC4_GL:
77                      format = GL_COMPRESSED_RGBA_PVRTC_4BPPV1_IMG;
78                      break;
79                  case PVRTC_PVRTC2_GL:
80                      format = GL_COMPRESSED_RGBA_PVRTC_2BPPV1_IMG;
81                      break;
82                  default:
83                      // if not special compressed, why use PVRTC format?
84                      // todo : mapping more PVR possible content to
GLES internal format
```

唔，虽然写了个 TODO，但是不知道啥时候才 DO，所以如果需要的话自己先写吧，写好了如果不介意公开的话， 发到论坛上或 email 我，我会补充到这个页面上的。

```
85                      format = -1;
86                      ;
87              }
88
89              data = (unsigned char *) malloc( h.dwTextureDataSize );
90              fread( data, h.dwTextureDataSize, 1, fp );
91              fclose(fp);
92          }
93 };
94
95 #endif // PVR_TEXTURE_H
```

文件格式相关的结构和常量在一个单独的文件里：
内容基本上是直接抄的 SDK PVR 文件格式说明文档的，所以应该不需要再解释什么了。

```
//
//  @ Project : PVRTC Wrapper
//  @ File Name : PVRTCFile.h
//  @ Date : 2007-8-30
//  @ Author : kongfu.yang
//  @ Company : http://www.play3d.net
//  @ Copyright : 2007-8, Example source license.

By keep this header comment, you may use this source file in your project
for non-commicial or commicial purpose.

#ifndef PVR_TC_FILE_H
#define PVR_TC_FILE_H

typedef struct PVR_Header_TAG
{
    DWORD dwHeaderSize;         /* size of the structure */
    DWORD dwHeight;             /* height of surface to be created */
    DWORD dwWidth;              /* width of input surface */
    DWORD dwMipMapCount;        /* Number of additional mipmap levels
*/
    DWORD dwpfFlags;            /* pixel format flags */
    DWORD dwTextureDataSize;    /* Total size in bytes */
    DWORD dwBitCount;           /* number of bits per pixel  */
    DWORD dwRBitMask;           /* mask for red bit */
    DWORD dwGBitMask;           /* mask for green bits */
    DWORD dwBBitMask;           /* mask for blue bits */
    DWORD dwAlphaBitMask;       /* mask for alpha channel */
} PVR_Header;

#define PVRTC_ARGB_4444     0x00000000
#define PVRTC_ARGB_1555     0x00000001
#define PVRTC_RGB_565       0x00000002
#define PVRTC_RGB_555       0x00000003
#define PVRTC_RGB_888       0x00000004
#define PVRTC_ARGB_8888     0x00000005
#define PVRTC_ARGB_8332     0x00000006
#define PVRTC_I_8           0x00000007
#define PVRTC_AI_88         0x00000008
#define PVRTC_1_BPP         0x00000009
#define PVRTC_VY1UY0        0x0000000A
```

```
38 #define PVRTC_Y1VY0U          0x0000000B
39 #define PVRTC_PVRTC2          0x0000000C
40 #define PVRTC_PVRTC4          0x0000000D
41 #define PVRTC_RGBA_4444       0x00000010
42 #define PVRTC_RGBA_5551       0x00000011
43 #define PVRTC_RGBA_8888       0x00000012
44 #define PVRTC_PVRTC2_GL       0x00000018
45 #define PVRTC_PVRTC4_GL       0x00000019
46 #define PVRTC_MIPMap          0x00000100
47 #define PVRTC_Twiddle         0x00000200
48 #define PVRTC_NormalMap       0x00000400
49
50
51 #endif // PVR_TC_FILE_H
52
```

全屏抗锯齿 FSAA

FSAA 是 Full Screen AntiAlias 的缩写，这是一种可由硬件实现的低成本抗锯齿方法，由 pipeline 在光栅化阶段做一下 Super Sampling，效果不错，速度也快。OpenGL ES 给厂商实现 AntiAlias 提供了非常大的灵活性，只有这种 MultiSample 有标准的接口：glEnable (GL_MULTISAMPLE)。
要启用 FSAA 是有前提的，即硬件支持，并且选择了合适的 EGLConfig。所谓合适的 Config 就是有 SAMPLE_BUFFERS(通常 1 个 Buffer)、指定了 SAMPLE 复杂度(2 或 4)：

```
// choose config
EGLint cfg_attr_list[] = {
  EGL_BUFFER_SIZE, bpp,
  EGL_SAMPLE_BUFFERS, sampleBuffers,
  EGL_SAMPLES, samples,
  EGL_NONE
};
```

然后在绘制时 enable 之即可。

奇怪的是，在 PVR PCE 里，这个特性无法正常使用，可以取得 Config，但是 CreateContext 会失败。

在 X51v WM5.0 下运行得倒还好

由于 M8 支持 OpenGL ES2.0，所以文章中的有些内容可能并不相符合，还是那句话，帮助入门，本人从事 Windows 平台软件开发，刚买 M8 没多久，买小 8 就是看中了它的可开发性，希望可以开发点东西，自娱自乐。但是使用后，对 M8 当前的软件整体素质还是感到

比较失望，M8 的 SDK 也并不完善，不过我始终相信 M8 的未来，事在人为，相信我们可以让 M8 更好。

呵呵，本人主业还是 PC 开发，手机开发只是兴趣，大家随便聊聊，有问题想共同解决欢迎联系本人 QQ 15415590 进行交流。

抛砖引玉……