http://blog.csdn.net/myarrow/article/details/7743338

**一、Vertex Attributes简介**

       Vertex属性即顶点数据，它指定了每个顶点的各种属性数据。在OpenGL ES1.1中，顶点属性有四个预定义的名字：position（位置）, normal（法线）, color（颜色）, 和 texture coordinates（纹理坐标）。在OpenGL ES2.0中，用户必须定义“顶点属性的名字”。

**二、常量顶点属性（Constant Vertex Attribute）**

     常量顶点属性对所有顶点都是一样的。因此只需要指定一个值就可以应用于所有顶点。一般很少使用。其设置函数有：

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/7743338) [copy](http://blog.csdn.net/myarrow/article/details/7743338)

[在CODE上查看代码片](https://code.csdn.net/snippets/119336)

1. **void** glVertexAttrib1f(GLuint index, GLfloat x);
2. **void** glVertexAttrib2f(GLuint index, GLfloat x, GLfloat y);
3. **void** glVertexAttrib3f(GLuint index, GLfloat x, GLfloat y, GLfloat z);
4. **void** glVertexAttrib4f(GLuint index, GLfloat x, GLfloat y, GLfloat z,GLfloat w);
5. **void** glVertexAttrib1fv(GLuint index, **const** GLfloat \*values);
6. **void** glVertexAttrib2fv(GLuint index, **const** GLfloat \*values);
7. **void** glVertexAttrib3fv(GLuint index, **const** GLfloat \*values);
8. **void** glVertexAttrib4fv(GLuint index, **const** GLfloat \*values);

**三、 如何装载顶点数据？（Vertex Arrays）**

       Vertex Array（顶点数组）：是一个存储在应用程序空间（Client）中的内存buffer，它存储了每个顶点的属性数据。

       如何把顶点数据组的数据传递给GPU呢？

        void glVertexAttribPointer(GLuint index,

                                                GLint size,     //每个属性元素个数有效值1-4（x,y,z,w）  
                                                GLenum type,  
                                                GLboolean normalized,  
                                                GLsizei stride, //如果数据连续存放，则为0  
                                                const void \*ptr)  //顶点数组指针

       举例如下：

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/7743338) [copy](http://blog.csdn.net/myarrow/article/details/7743338)

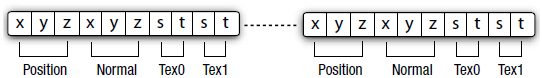
[在CODE上查看代码片](https://code.csdn.net/snippets/119336)

1. GLfloat vVertices[] = {  0.0f,  0.5f, 0.0f,
2. -0.5f, -0.5f, 0.0f,
3. 0.5f, -0.5f, 0.0f };
5. // Set the viewport
6. glViewport ( 0, 0, esContext->width, esContext->height );
8. // Clear the color buffer
9. glClear ( GL\_COLOR\_BUFFER\_BIT );
11. // Use the program object
12. glUseProgram (programObject );
14. // Load the vertex data
15. glVertexAttribPointer ( 0, 3, GL\_FLOAT, GL\_FALSE, 0, vVertices );

**四、顶点属性数据存储方式**

**1.一个顶点的所有属性存储在一起（Array of Structures）**

       如下图所示，顶点的位置(x,y,z)、法线(x,y,z)和两个纹理坐标(s,t)存储在一起，如下图所示：



例子代码如下（当然，此代码在CPU上运动）：

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/7743338) [copy](http://blog.csdn.net/myarrow/article/details/7743338)

[在CODE上查看代码片](https://code.csdn.net/snippets/119336)

1. #define VERTEX\_POS\_SIZE         3 // x, y and z
2. #define VERTEX\_NORMAL\_SIZE      3 // x, y and z
3. #define VERTEX\_TEXCOORD0\_SIZE   2 // s and t
4. #define VERTEX\_TEXCOORD1\_SIZE   2 // s and t
6. #define VERTEX\_POS\_INDX         0
7. #define VERTEX\_NORMAL\_INDX      1
8. #define VERTEX\_TEXCOORD0\_INDX   2
9. #define VERTEX\_TEXCOORD1\_INDX   3
11. // the following 4 defines are used to determine location of various
12. // attributes if vertex data is are stored as an array of structures
13. #define VERTEX\_POS\_OFFSET       0
14. #define VERTEX\_NORMAL\_OFFSET    3
15. #define VERTEX\_TEXCOORD0\_OFFSET 6
16. #define VERTEX\_TEXCOORD1\_OFFSET 8
18. #define VERTEX\_ATTRIB\_SIZE  VERTEX\_POS\_SIZE + \
19. VERTEX\_NORMAL\_SIZE + \
20. VERTEX\_TEXCOORD0\_SIZE + \
21. VERTEX\_TEXCOORD1\_SIZE
23. **float** \*p = malloc(numVertices \* VERTEX\_ATTRIB\_SIZE
24. \* **sizeof**(**float**));
26. // position is vertex attribute 0
27. glVertexAttribPointer(VERTEX\_POS\_INDX, VERTEX\_POS\_SIZE,
28. GL\_FLOAT, GL\_FALSE,
29. VERTEX\_ATTRIB\_SIZE \* **sizeof**(**float**),
30. p+VERTEX\_POS\_OFFSET);
32. // normal is vertex attribute 1
33. glVertexAttribPointer(VERTEX\_NORMAL\_INDX, VERTEX\_NORMAL\_SIZE,
34. GL\_FLOAT, GL\_FALSE,
35. VERTEX\_ATTRIB\_SIZE \* **sizeof**(**float**),
36. (p + VERTEX\_NORMAL\_OFFSET));
38. // texture coordinate 0 is vertex attribute 2
39. glVertexAttribPointer(VERTEX\_TEXCOORD0\_INDX, VERTEX\_TEXCOORD0\_SIZE,
40. GL\_FLOAT, GL\_FALSE,
41. VERTEX\_ATTRIB\_SIZE \* **sizeof**(**float**),
42. (p + VERTEX\_TEXCOORD0\_OFFSET));
44. // texture coordinate 1 is vertex attribute 3
45. glVertexAttribPointer(VERTEX\_TEXCOORD1\_INDX, VERTEX\_TEXCOORD1\_SIZE,
46. GL\_FLOAT, GL\_FALSE,
47. VERTEX\_ATTRIB\_SIZE \* **sizeof**(**float**),
48. (p + VERTEX\_TEXCOORD1\_OFFSET));

**2. 顶点的每个属性单独存储（Structure of Arrays）**

例子代码如下（当然，此代码在CPU上运动）：

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/7743338) [copy](http://blog.csdn.net/myarrow/article/details/7743338)

[在CODE上查看代码片](https://code.csdn.net/snippets/119336)

1. #define VERTEX\_POS\_SIZE         3 // x, y and z
2. #define VERTEX\_NORMAL\_SIZE      3 // x, y and z
3. #define VERTEX\_TEXCOORD0\_SIZE   2 // s and t
4. #define VERTEX\_TEXCOORD1\_SIZE   2 // s and t
6. #define VERTEX\_POS\_INDX         0
7. #define VERTEX\_NORMAL\_INDX      1
8. #define VERTEX\_TEXCOORD0\_INDX   2
9. #define VERTEX\_TEXCOORD1\_INDX   3

12. #define VERTEX\_ATTRIB\_SIZE  VERTEX\_POS\_SIZE + \
13. VERTEX\_NORMAL\_SIZE + \
14. VERTEX\_TEXCOORD0\_SIZE + \
15. VERTEX\_TEXCOORD1\_SIZE
17. **float** \*position  = malloc(numVertices \* VERTEX\_POS\_SIZE \*
18. **sizeof**(**float**));
19. **float** \*normal    = malloc(numVertices \* VERTEX\_NORMAL\_SIZE \*
20. **sizeof**(**float**));
21. **float** \*texcoord0 = malloc(numVertices \* VERTEX\_TEXCOORD0\_SIZE \*
22. **sizeof**(**float**));
23. **float** \*texcoord1 = malloc(numVertices \* VERTEX\_TEXCOORD1\_SIZE \*
24. **sizeof**(**float**));
26. // position is vertex attribute 0
27. glVertexAttribPointer(VERTEX\_POS\_INDX, VERTEX\_POS\_SIZE,
28. GL\_FLOAT, GL\_FALSE,
29. VERTEX\_POS\_SIZE \* **sizeof**(**float**), position);
31. // normal is vertex attribute 1
32. glVertexAttribPointer(VERTEX\_NORMAL\_INDX, VERTEX\_NORMAL\_SIZE,
33. GL\_FLOAT, GL\_FALSE,
34. VERTEX\_NORMAL\_SIZE \* **sizeof**(**float**), normal);
36. // texture coordinate 0 is vertex attribute 2
37. glVertexAttribPointer(VERTEX\_TEXCOORD0\_INDX, VERTEX\_TEXCOORD0\_SIZE,
38. GL\_FLOAT, GL\_FALSE, VERTEX\_TEXCOORD0\_SIZE \*
39. **sizeof**(**float**), texcoord0);
41. // texture coordinate 1 is vertex attribute 3
42. glVertexAttribPointer(VERTEX\_TEXCOORD1\_INDX, VERTEX\_TEXCOORD1\_SIZE,
43. GL\_FLOAT, GL\_FALSE,
44. VERTEX\_TEXCOORD1\_SIZE \* **sizeof**(**float**),  //也可为0，因为数据是紧接着存放的
45. texcoord1);

**3. 哪种顶点属性数据存储方式在GPU上性能更好？**

**答案是：**把一个顶点的所有属性放在一起（array of structures）。其原因是每个顶点的属性数据以连续的方式读取，使内存访问效率更高。其缺点是，如果要修改其中部分属性数据，将导致整个属性buffer全部重新装载，解决此问题的方法是把这些需要动态修改的属性数据放在单独的buffer中。

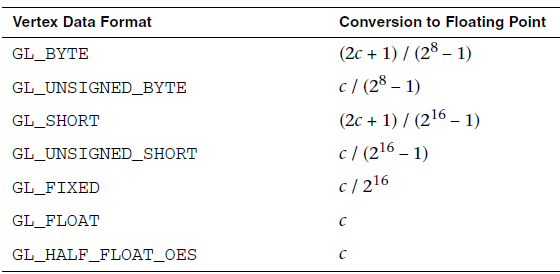
**五、 顶点属性数据类型优化**

       顶点属性数据类型不会影响在GPU上每个顶点占用的内存，但在Render a Frame时，它影响CPU与GPU之间的内存带宽。推荐尽量使用GL\_HALF\_FLOAT\_OES。

**六、glVertexAttribPointer中的归一化参数**

       如果normalized为GL\_FALSE：则直接把数据转换为GLfloat，因为Vertex Shader内部把顶点属性当作GLfloat（32位）来进行存储。

       GL\_BYTE, GL\_SHORT or GL\_FIXED被归一化为[-1，1]；GL\_UNSIGNED\_BYTE or GL\_UNSIGNED\_SHORT被归一化为[0.0，1.0]。具体转换公式为：

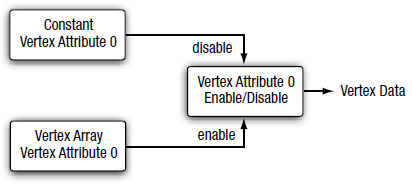


**七、选择常量或顶点数组Vertex Attribute**

可通过以下函数来Enable或Disable顶点数组（Vertex Array）。

void glEnableVertexAttribArray(GLuint index);  
void glDisableVertexAttribArray(GLuint index);

其关系如下图所示：



**八、申明attribute变量（在Vertex Shader中）**

      Attribute变量的数据类型只能为：float, vec2,vec3, vec4, mat2, mat3, and mat4；Attribute变量不能为数组或结构。如下面的申明是错误的：

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/7743338) [copy](http://blog.csdn.net/myarrow/article/details/7743338)

[在CODE上查看代码片](https://code.csdn.net/snippets/119336)

1. attribute foo\_t a\_A; // foo\_t is a structure
2. attribute vec4 a\_B[10];

     每个GPU支持GL\_MAX\_VERTEX\_ATTRIBS vec4。float、vec2和vec3也被当作一个vec4来进行存储；mat2、mat3、mat4被分别当作2、3、和4个vec4来进行存储。

**九、把“顶点属性索引”绑定到“顶点属性名”**

       把“顶点属性索引”绑定到“顶点属性名”有以下两个方法：

       1）**OpenGL ES 2.0**把“顶点属性索引”绑定到“顶点属性名”，当link program时，OpengGL ES 2.0执行此绑定。然后应用程序通过glGetAttribLocation（失败时返回-1）获取“顶点属性索引”。

**[html]** [view plain](http://blog.csdn.net/myarrow/article/details/7743338) [copy](http://blog.csdn.net/myarrow/article/details/7743338)

[在CODE上查看代码片](https://code.csdn.net/snippets/119336)

1. GLint glGetAttribLocation(GLuint program, const GLchar \*name)

       2）**应用程序通过glBindAttribLocation**把“顶点属性索引”绑定到“顶点属性名”，glBindAttribLocation在program被link之前执行。

**[cpp]** [view plain](http://blog.csdn.net/myarrow/article/details/7743338) [copy](http://blog.csdn.net/myarrow/article/details/7743338)

[在CODE上查看代码片](https://code.csdn.net/snippets/119336)

1. **void** glBindAttribLocation(GLuint program, GLuint index,**const** GLchar \*name)

       在link program时，OpenGL ES 2.0对每个顶点属性执行如下操作：

        （1）首先检查属性变量是否被通过glBindAttribLocation绑定了属性索引，如果是，则使用此绑定的属性索引；否则，为之分配一个属性索引。

             在应用程序中，一般使用函数glBindAttribLocation来绑定每个attribute变量的位置，然后用函数glVertexAttribPointer为每个attribute变量赋值。

**十、顶点buffer对象（Vertex Buffer Objects）**

       顶点数组（Vertex Array）被保存在客户端内存，当执行glDrawArrays 或 glDrawElements时，才把它们从客户端内存copy到图形内存。这样占用了大量的内存带宽，Vertex Buffer Objects允许OpengGL ES2.0应用在高性能的图形内存中分配并cache顶点数据，然后从此图形内存中执行render，这样避免了每次画一个原语都要重发送数据。

       Vertex Buffer Objects有以下两种类型：

       （1）array buffer objects：通过GL\_ARRAY\_BUFFER标记创建，并存储vertex data。

       （2）element array buffer objects：通过GL\_ELEMENT\_ARRAY\_BUFFER标记创建，并存储indices of a primitive。

       创建和绑定Vertex Buffer Objects例子代码如下：

**[html]** [view plain](http://blog.csdn.net/myarrow/article/details/7743338) [copy](http://blog.csdn.net/myarrow/article/details/7743338)

[在CODE上查看代码片](https://code.csdn.net/snippets/119336)

1. void initVertexBufferObjects(vertex\_t \*vertexBuffer,
2. GLushort \*indices,
3. GLuint numVertices, GLuint numIndices
4. GLuint \*vboIds)
5. {
6. glGenBuffers(2, vboIds);
7. glBindBuffer(GL\_ARRAY\_BUFFER, vboIds[0]);
8. glBufferData(GL\_ARRAY\_BUFFER, numVertices \* sizeof(vertex\_t),
9. vertexBuffer, GL\_STATIC\_DRAW); //save vertex attribute data
10. // bind buffer object for element indices
11. glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, vboIds[1]);
12. glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER,
13. numIndices \* sizeof(GLushort),indices,
14. GL\_STATIC\_DRAW); // save element indices that make up the primitive
15. }

**有无Vertex Buffer Object画图例子：**

**[html]** [view plain](http://blog.csdn.net/myarrow/article/details/7743338) [copy](http://blog.csdn.net/myarrow/article/details/7743338)

[在CODE上查看代码片](https://code.csdn.net/snippets/119336)

1. #define VERTEX\_POS\_SIZE 3 // x, y and z
2. #define VERTEX\_NORMAL\_SIZE 3 // x, y and z
3. #define VERTEX\_TEXCOORD0\_SIZE 2 // s and t
4. #define VERTEX\_POS\_INDX 0
5. #define VERTEX\_NORMAL\_INDX 1
6. #define VERTEX\_TEXCOORD0\_INDX 2
7. //
8. // vertices – pointer to a buffer that contains vertex attribute
9. data
10. // vtxStride – stride of attribute data / vertex in bytes
11. // numIndices – number of indices that make up primitive
12. // drawn as triangles
13. // indices - pointer to element index buffer.
14. //
15. void drawPrimitiveWithoutVBOs(GLfloat \*vertices, GLint vtxStride,
16. GLint numIndices, GLushort \*indices)
17. {
18. GLfloat \*vtxBuf = vertices;
19. glBindBuffer(GL\_ARRAY\_BUFFER, 0);
20. glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, 0);
21. glEnableVertexAttribArray(VERTEX\_POS\_INDX);
22. glEnableVertexAttribArray(VERTEX\_NORMAL\_INDX);
23. glEnableVertexAttribArray{VERTEX\_TEXCOORD0\_INDX);
24. glVertexAttribPointer(VERTEX\_POS\_INDX, VERTEX\_POS\_SIZE,
25. GL\_FLOAT, GL\_FALSE, vtxStride, vtxBuf);
26. vtxBuf += VERTEX\_POS\_SIZE;
27. glVertexAttribPointer(VERTEX\_NORMAL\_INDX, VERTEX\_NORMAL\_SIZE,
28. GL\_FLOAT, GL\_FALSE, vtxStride, vtxBuf);
29. vtxBuf += VERTEX\_NORMAL\_SIZE;
30. glVertexAttribPointer(VERTEX\_TEXCOORD0\_INDX,
31. VERTEX\_TEXCOORD0\_SIZE, GL\_FLOAT,
32. GL\_FALSE, vtxStride, vtxBuf);
33. glBindAttribLocation(program, VERTEX\_POS\_INDX, "v\_position");
34. glBindAttribLocation(program, VERTEX\_NORMAL\_INDX, "v\_normal");
35. glBindAttribLocation(program, VERTEX\_TEXCOORD0\_INDX,"v\_texcoord");
36. glDrawElements(GL\_TRIANGLES, numIndices, GL\_UNSIGNED\_SHORT,indices);
37. }

**[html]** [view plain](http://blog.csdn.net/myarrow/article/details/7743338) [copy](http://blog.csdn.net/myarrow/article/details/7743338)

[在CODE上查看代码片](https://code.csdn.net/snippets/119336)

1. void drawPrimitiveWithVBOs(GLint numVertices, GLfloat \*vtxBuf,
2. GLint vtxStride, GLint numIndices,
3. GLushort \*indices)
4. {
5. GLuint offset = 0;
6. GLuint vboIds[2];
7. // vboIds[0] – used to store vertex attribute data
8. // vboIds[1] – used to store element indices
9. glGenBuffers(2, vboIds);
10. glBindBuffer(GL\_ARRAY\_BUFFER, vboIds[0]);
11. glBufferData(GL\_ARRAY\_BUFFER, vtxStride \* numVertices,
12. vtxBuf, GL\_STATIC\_DRAW);
13. glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, vboIds[1]);
14. glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER,
15. sizeof(GLushort) \* numIndices,
16. indices, GL\_STATIC\_DRAW);
17. glEnableVertexAttribArray(VERTEX\_POS\_INDX);
18. glEnableVertexAttribArray(VERTEX\_NORMAL\_INDX);
19. glEnableVertexAttribArray{VERTEX\_TEXCOORD0\_INDX);
20. glVertexAttribPointer(VERTEX\_POS\_INDX, VERTEX\_POS\_SIZE,
21. GL\_FLOAT, GL\_FALSE, vtxStride,
22. (const void\*)offset);
23. offset += VERTEX\_POS\_SIZE \* sizeof(GLfloat);
24. glVertexAttribPointer(VERTEX\_NORMAL\_INDX, VERTEX\_NORMAL\_SIZE,
25. GL\_FLOAT, GL\_FALSE, vtxStride,
26. (const void\*)offset);
27. offset += VERTEX\_NORMAL\_SIZE \* sizeof(GLfloat);
28. glVertexAttribPointer(VERTEX\_TEXCOORD0\_INDX,
29. VERTEX\_TEXCOORD0\_SIZE,
30. GL\_FLOAT, GL\_FALSE, vtxStride,
31. (const void\*)offset);
32. glBindAttribLocation(program, VERTEX\_POS\_INDX, "v\_position");
33. glBindAttribLocation(program, VERTEX\_NORMAL\_INDX, "v\_normal");
34. glBindAttribLocation(program, VERTEX\_TEXCOORD0\_INDX,"v\_texcoord");
35. glDrawElements(GL\_TRIANGLES, numIndices, GL\_UNSIGNED\_SHORT, 0);
36. glDeleteBuffers(2, vboIds);
37. }