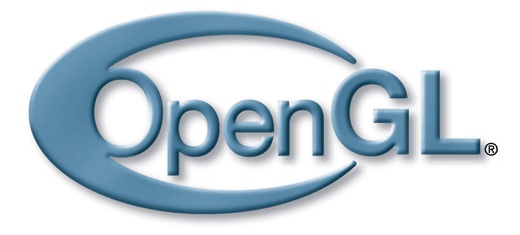
## 图形引擎



OpenGL（全写Open Graphics Library）是个定义了一个跨编程语言、跨平台的程序接口(Application programming interface)的规格，它用于生成二维、三维图像。这个接口由近三百五十个不同的函数调用组成，用来从简单的图形比特绘制复杂的三维景象。而另一种程序接口系统是仅用于Microsoft Windows上的Direct3D。OpenGL常用于CAD、虚拟实境、科学可视化程序和电子游戏开发。

游戏开发人员是一个有着独立思想的群体，很多重要的开发人员目前仍然在使用OpenGL。

WebGL测试：

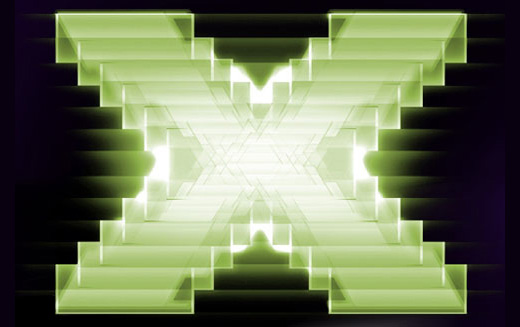
<http://www.irrlicht3d.org/pivot/entry.php?id=1360>

<http://www.hiwebgl.com/?cat=16>

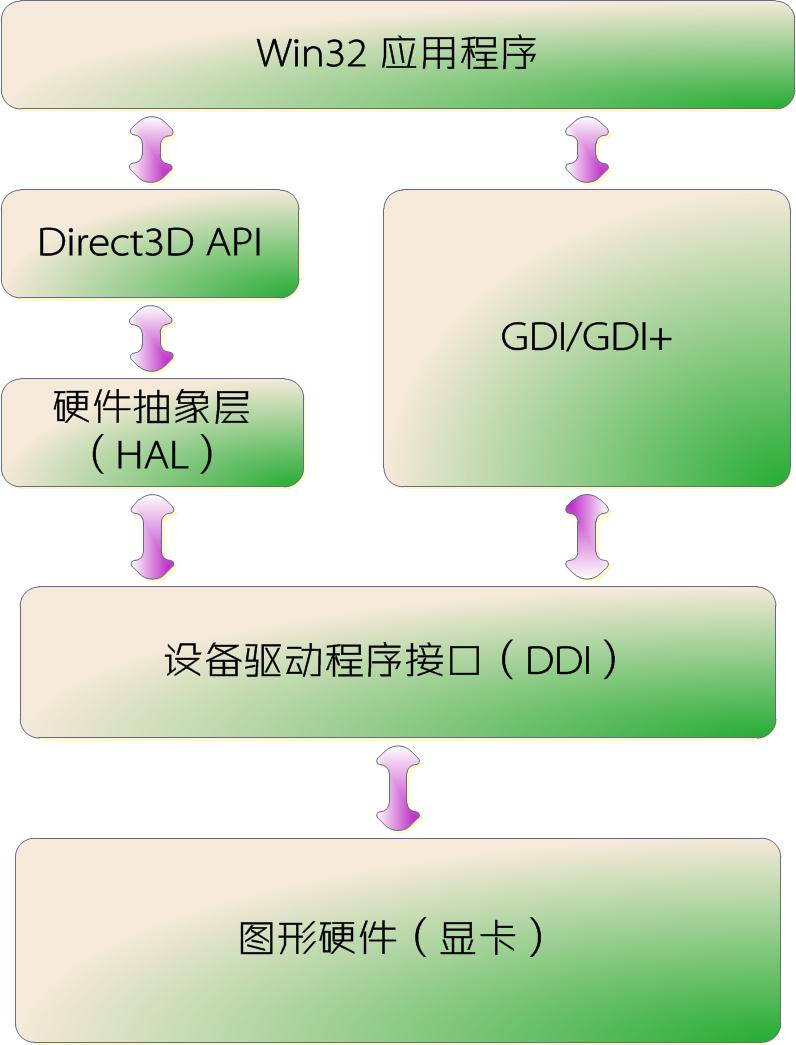
WebGL (Web Graphics Library) is a JavaScript API for rendering interactive 3D graphics and 2D graphics within any compatible web browser without the use of plug-ins.

WebGL is based on OpenGL ES 2.0 and provides an API for 3D graphics.

Mesa3D是一个几乎和Opengl相同的免费开源图形API，它在功能上一点也不输给Opengl。作为免费开源软件，免去了诸多侵权问题的烦恼。



DirectX(Direct eXtension)占据着游戏市场绝对的优势，据估计目前市面上几乎所有的电脑游戏都是基于DirectX而开发的。《百度今日网游排行榜》上的前五十名，无一例外，全部拜倒在DirectX的石榴裙下。这里我们列举出前十名，他们依次是《地下城与勇士》，《穿越火线》 ，《英雄联盟》，《魔兽世界》，《QQ飞车》，《QQ炫舞》，《梦幻西游》《龙之谷》，《倩女幽魂》，《第九大陆》 。（2012年4月29号榜单）



虽然DirectX在家用市场全面领先，但在专业高端绘图领域，OpenGL是不能被取代的主角。

Direct3D目前还不能支持高端的图形设备和专业应用； OpenGL在这些领域占据着统治地位。

关于跨平台，这是OpenGL对比DirectX绝对的优势。值得一提的是，由于最近几年移动开发市场的迅猛发展，Android和iOS需要一款的专业的图形库，跨平台的OpenGL无疑是最好的选择。

**暴雪总是同时发布游戏的mac版本，这是他们成功的一项法宝。**

## OpenGL ES介绍

OpenGL ES (OpenGL for Embedded Systems) 是 OpenGL 三维图形 API 的子集，针对手机、PDA和游戏主机等嵌入式设备而设计。该API由Khronos集团定义推广，Khronos是一个图形软硬件行业协会，该协会主要关注图形和多媒体方面的开放标准。

OpenGL ES 是从 OpenGL 裁剪定制而来的，去除了 glBegin/glEnd，四边形（GL\_QUADS）、多边形（GL\_POLYGONS）等复杂图元等许多非绝对必要的特性。

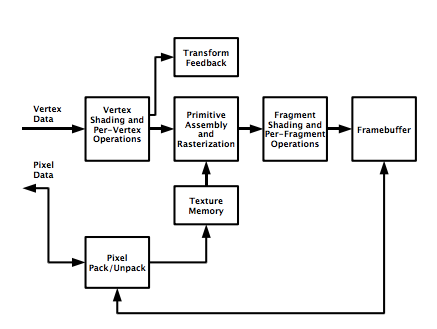
　android系统支持opengl es.但是android2.1及其以下版本不支持OpenGL ES2.0。另外，到3.0（包括3.0）的模拟器都不支持OpenGL ES2.0(2.1等)。

　android2.2及以上版本的实际设备上可以运行OpenGL ES2.0写的程序。

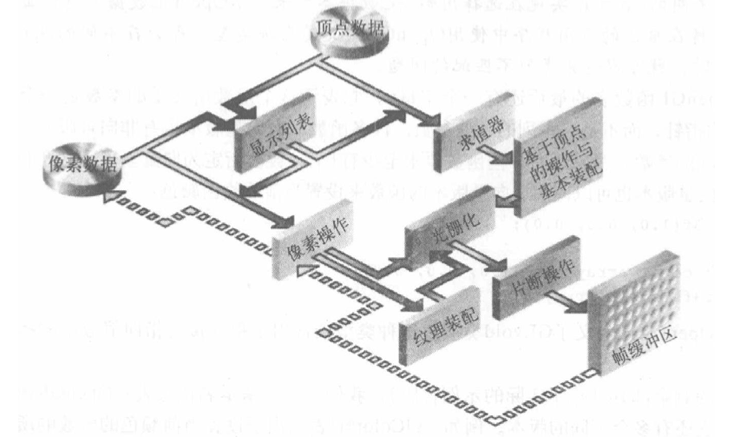
OpenGL ES 1.0 和 1.1API 规范从Android1.0就开始支持．从Android2.2 (API Level 8)开始，框架支持OpenGL ES 2.0 API规范．OpenGL ES 2.0被大多数Android设备所支持并被推荐在新的基于OpenGL的应用中使用．Android 4.3开始支持OpenGL ES 3.0。

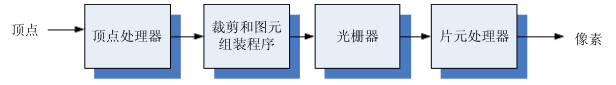
## 基本的3D渲染流水线

流程：顶点🡪图元🡪片元🡪像素

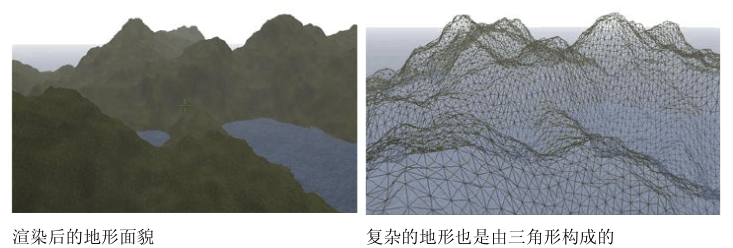


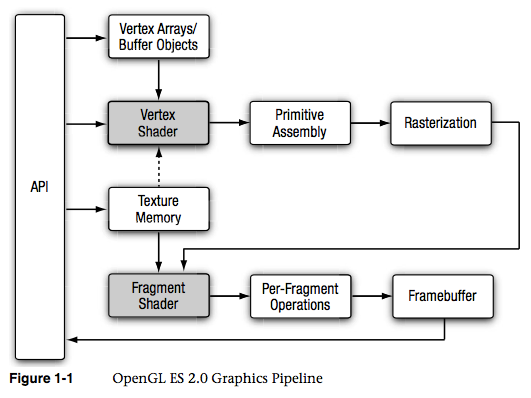
渲染管线





OpenGL着色语言允许程序员编写自己的像素和顶点着色器。





* 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。在今天的示例中，简单起见，使用的是 Vertex Array；
* 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；
* 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。
* 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。
  + 光栅化是将一个图元转变为一个二维图象的过程。二维图象上每个点都包含了颜色、深度和纹理数据。将该点和相关信息叫做一个片元（fragment）。（yuyu注：这就是片元和像素之间的关键区别，虽然两者的直观印象都是的像素，但是片元比像素多了许多信息，在光栅化中纹理映射之后图元信息转化为了像素）, 在这个阶段，对象素绘制和位图进行操作需要用到当前栅格位置（用glRasterPos\*()定义）。正如上面讨论的，三种图元的光栅化方法是不同的，另外，象素块和位图也需要光栅化。
* 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。
* 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。
* 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，但不能读取其他值（如深度值，模版值等）。

终端厂商在宣传他们手机芯片的GPU如何强劲时，往往会提到一些参数，最常见的就是三角形生成率和填充率。

三角形生成率对应：Vertex Shader

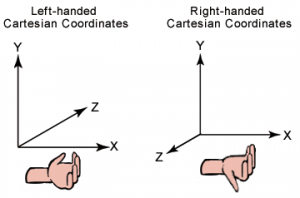
填充率对应：纹理贴图输出

## 基本知识

### 点或顶点vertex

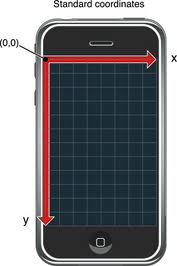
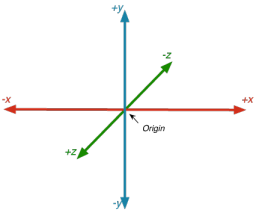
3D图像的最小单位称为 **点（point）** 或者 **顶点vertex**。它们代表三维空间中的一个点并用来建造更复杂的物体。多边形就是由点构成，而物体是由多个多边形组成。尽管通常OpenGL支持多种多边形，但OpenGL ES只支持三边形（即三角形）。

坐标



Direct3D坐标系：DirectX坐标系使用的是左手“笛卡尔”坐标系。

OpenGL坐标系：OpenGL坐标系使用的是右手“笛卡尔”坐标系。



OpenGL原点设置在左下角。iOS,Android操作系统的屏幕原点设置在左上角。

typedef struct {   
    GLfloat x;  
    GLfloat y;  
    GLfloat z;  
} Vertex3D;

空间中两点间的距离是使用下面公式计算的：



我们可以在一个简单的内联函数中实现这个公式来计算三维空间中任何两点间的直线距离：  
static inline GLfloat Vertex3DCalculateDistanceBetweenVertices (Vertex3D first, Vertex3D second)  
{  
    GLfloat deltaX = second.x - first.x;  
    GLfloat deltaY = second.y - first.y;  
    GLfloat deltaZ = second.z - first.z;  
    return sqrtf(deltaX\*deltaX + deltaY\*deltaY + deltaZ\*deltaZ );  
};

### 三角形Triangle

为什么是三角形，原来OpenGL ES仅支持三角形，因此我们可以通过创建一个数据结构将三个顶点组合成一个三角形物体。OpenGL不会。

typedef struct {

Vertex3D v1;

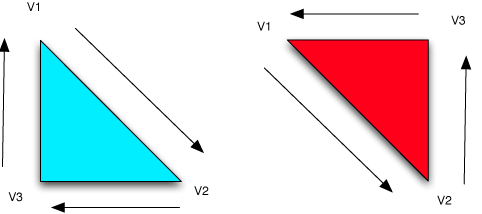
Vertex3D v2;

Vertex3D v3;

} Triangle3D;

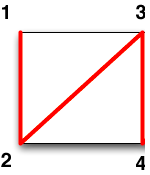
OpenGL中有一个概念叫卷绕(winding)， 它表示顶点绘制的次序.

OpenGL是通过观察顶点的绘制次序来确定front face（前面可见）和backface（背面不可见）。以反时针次序绘制顶点的构成的面是frontface（默认，可以改变）。



上图中，左边青色的三角形是backface，因此将不可见。而右方的三角形是frontface，所以将被绘制。

如果要绘制四方形则需要绘制两个三角形，如：



### 视角/摄像机ViewPort

OpenGL ES中具有的两种不同的视口类型：正交和透视。

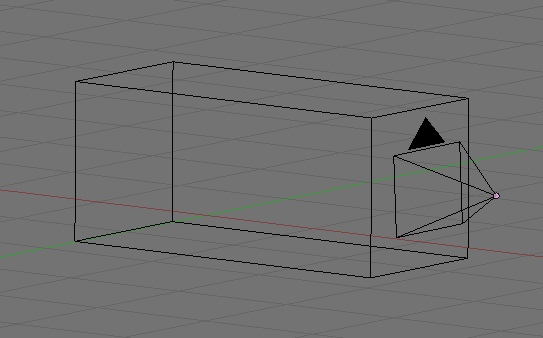
**透视（perspective**）

OpenGL可以设定的视口中的一种就是使用透视。当你这样设置视口时，物体会随着移远而越来越小，视线会在物体移离观察者时最终交汇。这是对真实视觉的模拟；人们就是以这种方式观察世界的。



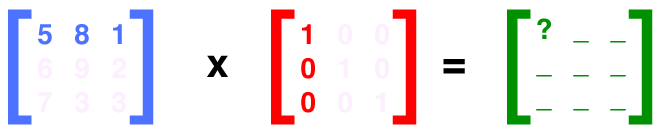
正交（orthogonal）

另一种看设置的视口称为**正交（orthogonal）** 视口。这种类型的视口，视线永远不会交汇而且物体不会改变其大小。没有透视效果。对于CAD程序以及其他各种目的是十分方便的，但因为它不像人们眼睛观察的方式所以看上去是不真实的，通常也不是你所希望的。



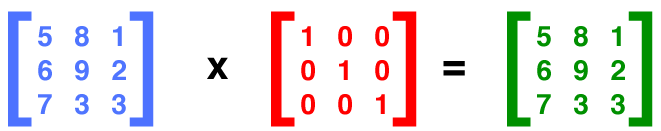
### 变换(scaling，rotation，translate)

矩阵相乘是矩阵组合的关键。如果你有一个定义了旋转的矩阵和一个定义了旋转的矩阵，如果将它们相乘，你将得到一个既定义了旋转又定义了转移的矩阵。



将左边矩阵行中第一个值乘以右边矩阵列的第一个值，左边矩阵行第二个值乘以右边矩阵列的第二个值，左边矩阵行第三个值乘以右边矩阵列的第三个值，然后再将它们相加。

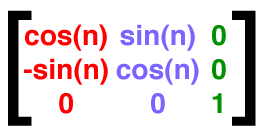




任何矩阵与单元矩阵（归一向量）相乘，矩阵值不变。当你调用glLoadIdentity()时， 你就是加载了一个单元矩阵。

如果说我们希望绕Z轴旋转一个物体，那么Z轴将保持不变，而X和Y轴将变化。

需要调整通过修改X轴向量的X值和Y向量的Y值为旋转角度的余弦值：



实际OpenGL使用的是4×4的GLfloat矩阵，分别为x,y,z加上a透明。

有兴趣的可以，基于苹果机器的向量处理器重写变换实现（scaling，rotation）。性能会有相应提高：从.7% 的处理时间降低为 .1% 的处理时间。

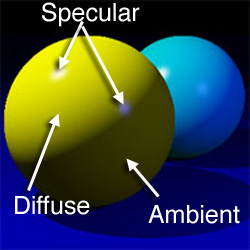
### 阴影shade

OpenGL ES实际上定义了两种阴影模型shade model， GL\_FLAT(90年底不考虑)和GL\_SMOOTH.

GL\_FLAT: 

OpenGL ES允许你创建8个光源

光由三个元素组成，分别是环境元素（ambient component）， 散射元素（diffuse component）和 高光元素（specular component）



* 高光元素定义了光线直接照射并反射到观察者从而形成了物体上的“热点”或光泽。光点的大小取决于一些因素，但是如果你看到如上图黄球所示一个区域明显的光斑，那通常就是来自于一个或多个光源的高光部分。
* 散射元素定义了比较平均的定向光源，在物体面向光线的一面具有光泽。
* 环境光则没有明显的光源。其光线折射与许多物体，因此无法确定其来源。环境元素平均作用于场景中的所有物体的所有面。

名词：

法线：始终垂直于某平面的虚线，公正无私，像个法官一样，故取名为法线，光效就需要顶点法线

### 纹理（texture）

当使用未压缩纹理时，将采用 PNG 等未压缩文件格式，并将其上传到 GPU。 由于 GPU 本身不支持 PNG 文件格式，您的纹理实际存储在 CPU 内存中。 这一点同样也适用于 JPEG 图像，图形芯片集对 JPEG 全然不知，因此将在 CPU 内存中对它们进行解码。

最好使用 GPU 内存，而不是 CPU 内存。 然而，为了使用 GPU 内存，您必须对 GPU 使用适当类型的纹理文件。

纹理格式

通用的纹理会有：RGBA的PNG，TGA，或者ETC1纹理

TGA是由美国Truevision公司为其显示卡开发的一种图像文件格式，已被国际上的图形、图像工业所接受。

Truevision公司推出TGA的目的是为了采集、输出电视图像，所以TGA文件总是按行存储、按 行进行压缩的，这使得它同时也成为计算机生成图像向电视转换的一种首选格式。

在工业设计领域，使用三维软件制作出来的图像可以利用TGA格式的优势，在图像内部生成一个Alpha（通道），这个功能方便了在平面软件中的工作。

Texture Compress

ETC1，这个是OPENGL ES 2.0支持的纹理格式，大家都得支持。但这个纹理的一个缺点是不支持alpha通道，所以对于有alpha通道的纹理，就要拆成2个纹理去读取，效率低，浪费了带宽

PVRTC是PowerVR自家的纹理格式，同样ATITC是高通Adreno的纹理格式，此外S3TC就是桌面很常见的DXT，微软DirectX 3D的纹理格式，这些都是支持alpha通道

PVRTC：PowerVR系列GPU支持 （苹果）  
ATITC：高通Adreno系列GPU支持，来自以前的ATI （小米）  
DXTC：nVIDIA Tegra系列/Geforce ULP系列，VivanteGC 系列  
**ETC1**：ARM的Mali系列GPU支持，以上四家也支持 （三星）(pkm或ktx扩展名)

PC：

DDS是一种[图片格式](http://baike.baidu.com/view/19666.htm" \t "_blank)。DirectDraw Surface的缩写，它是DirectX纹理压缩（DirectX Texture Compression，简称DXTC）的产物。

苹果跟Imagination 是一对好机友，GPU都是Imagination 的PowerVR 系列，所以iOS游戏厂商只要做支持PVRTC纹理就可以了。

Android的通用：ETC1

GLUT 代表OpenGL Utility Tookit

**二的幂次方**  
**OpenGL在申请内存存放纹理时，是按2的幂次方申请的，即对应480\*320的图片，它申请的是512\*512空间。可见，会有相当多的内存被浪费。所以，我们设计的图片，最好是2的幂次方，不然OpenGL最终还是申请2的幂次方内存空间。  
  
由于不是所有图片都刚好能设计成2的幂次方，因此通过“拼大图”方法解决。这就是为啥要使用TexturePacker的原因。**

纹理的宽高必须是2的n次幂，最大限制受硬件的影响。

**如，在iphone3g 里面，这个最大限制是1024\*1024,**

**在 iphone4g 里面，这个最大限制是 2048\*2048。**

另外，纹理不一定要是正方形的，128 \* 512 就非常完美的满足了这点需求。

**色深优化**  
图片所占内存的大小用这条公式计算：高度像素\*宽度像素\*色深。如色深RGBA8888，是32个bit，每8bit是一个字节，因此一个像素点占4字节，就是一个整形字符大小。例子。一个480\*320的图片大小，占内存480\*320\*4字节。

**如果对图片的色彩要求不是很高的话。ARGB8888（占4字节）可以改为，ARGB1555（2字节）(透明通道A占1位一般是用做图片混合运算)或者ARGB4444（如果不是做混合运算，用4位比较合适）。或者，如果不需要使用图片透明，就不需要A通道，直接使用RGB888（3字节）改为RGB565（2字节）。这样，图片的最终内存大小就会占用更少的空间。**

**TODO:使用PVR的时候需要去判断占用的是CPU还是GPU。**

将纹理的GL TEXTURE\_WRAP\_S与GL\_TEXTURE\_WRAP\_T属性值设置为GL\_ REPEAT而不是GL CLAMP\_TO\_EDGE。

线性过滤

选择了GL\_LINEAR，那么OpenGL就会对靠近像素中心的一块2×2纹理矩形单元取加权平均值，用于放大和缩小。

glTexParameteri( GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR );

### 动画

关键帧动画：我们存储独立关键位置的每一个顶点

骨骼动画：存储虚拟骨骼的位置信息，并且用一些方法指定哪个骨骼会影响动作中的哪些顶点

4元数

**骨骼动画**  
cocos2d-x 2.0.3开始支持骨骼动画，cocos2dBuilder2.1版本也可以进行动画编辑。在处理动画时，对内存优化也是很好的方案。

参考：

手机游戏开发纹理图片优化心得

<http://blog.csdn.net/langresser_king/article/details/9313255>

ATF SDK 压缩纹理简介

<http://www.adobe.com/cn/devnet/flashruntimes/articles/introducing-compressed-textures.html>

**OpenGL ES 从零开始系列**

<http://www.cocoachina.com/bbs/read.php?tid=84802>

OpenGL ES 2.0 iOS教程

http://blog.csdn.net/kesalin/article/details/8223649

顶点（vertexs） 图元（primitives） 片元（fragments片断） 像素（pixels）

http://blog.csdn.net/phpxin123/article/details/7409005