

# 基于视锥体（平截体）的OpenGL ES性能优化



作者 落影loyinglin (/u/815d10a4bdce) [+ 关注](#)

2016.05.03 15:29\* 字数 1435 阅读 978 评论 10 喜欢 16 阅读 978 评论 10 喜欢 16 (/u/815d10a4bdce)

## 教程

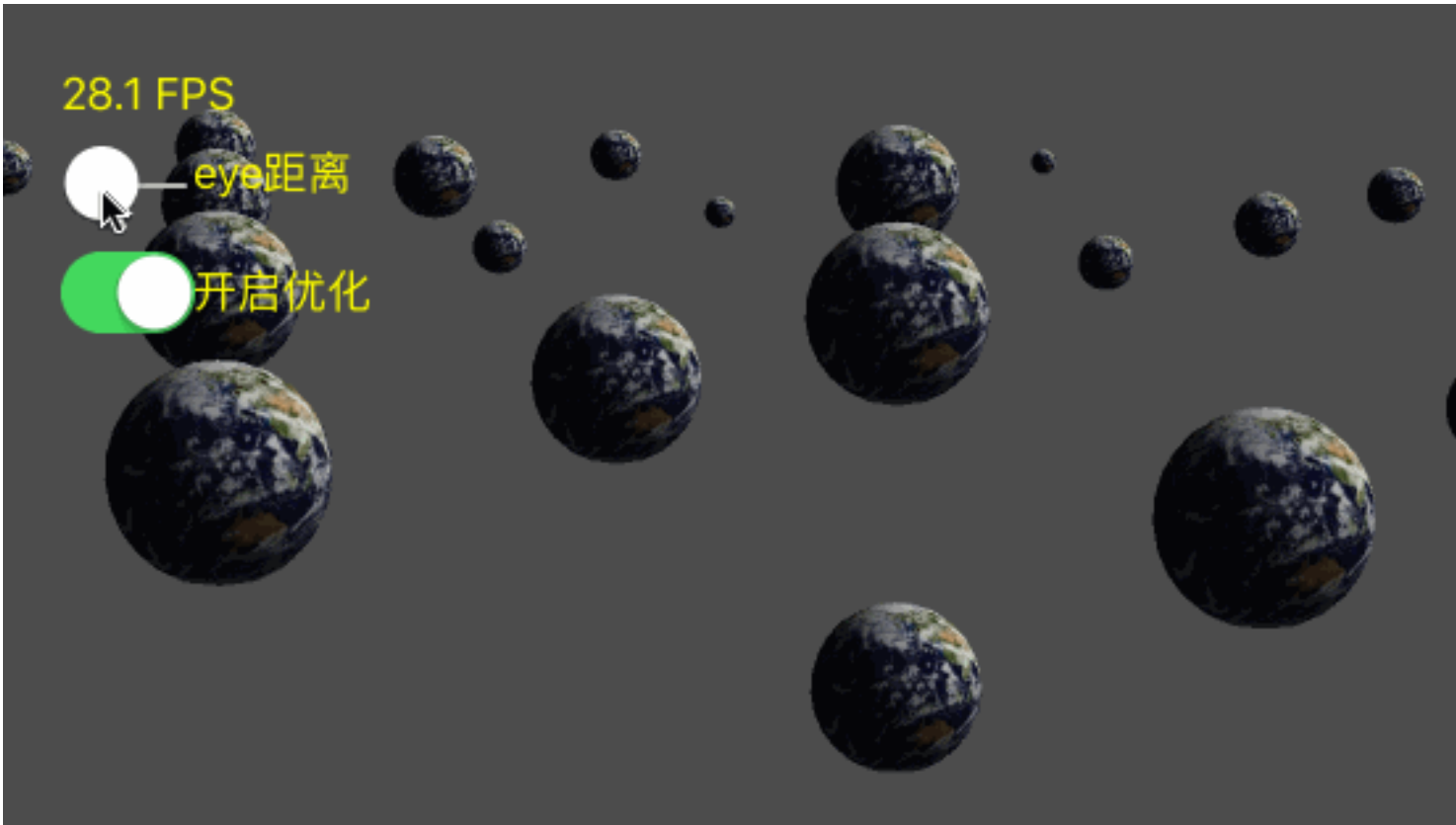
OpenGL ES入门教程1-Tutorial01-GLKit (<http://www.jianshu.com/p/750fde1d8b6a>)  
OpenGL ES入门教程2-Tutorial02-shader入门 (<http://www.jianshu.com/p/ee597b2bd399>)  
OpenGL ES入门教程3-Tutorial03-三维变换 (<http://www.jianshu.com/p/87c5413c1fc7>)  
OpenGL ES入门教程4-Tutorial04-GLKit进阶 (<http://www.jianshu.com/p/ed7fb9555839>)  
OpenGL ES进阶教程1-Tutorial05-地球月亮 (<http://www.jianshu.com/p/a82f3f66dddd>)  
OpenGL ES进阶教程2-Tutorial06-光线 (<http://www.jianshu.com/p/4e1a28f23e75>)  
OpenGL ES进阶教程3-Tutorial07-粒子效果 (<http://www.jianshu.com/p/b6d2441209f8>)  
OpenGL ES进阶教程4-Tutorial08-帧缓存 (<http://www.jianshu.com/p/1193b98634a2>)  
OpenGL ES进阶教程5-Tutorial09-碰碰车 (<http://www.jianshu.com/p/3b532f6fcedf>)  
这一次的是性能优化。

## 概要

渲染的优化不是仅仅提高渲染的速度，超过60Hz的渲染速度没有任何意义，用户永远看不到这些信息。同时在考虑用电消耗的情况下，30Hz的刷新率能延长电池的使用时间。以下的渲染优化策略总是管用的：

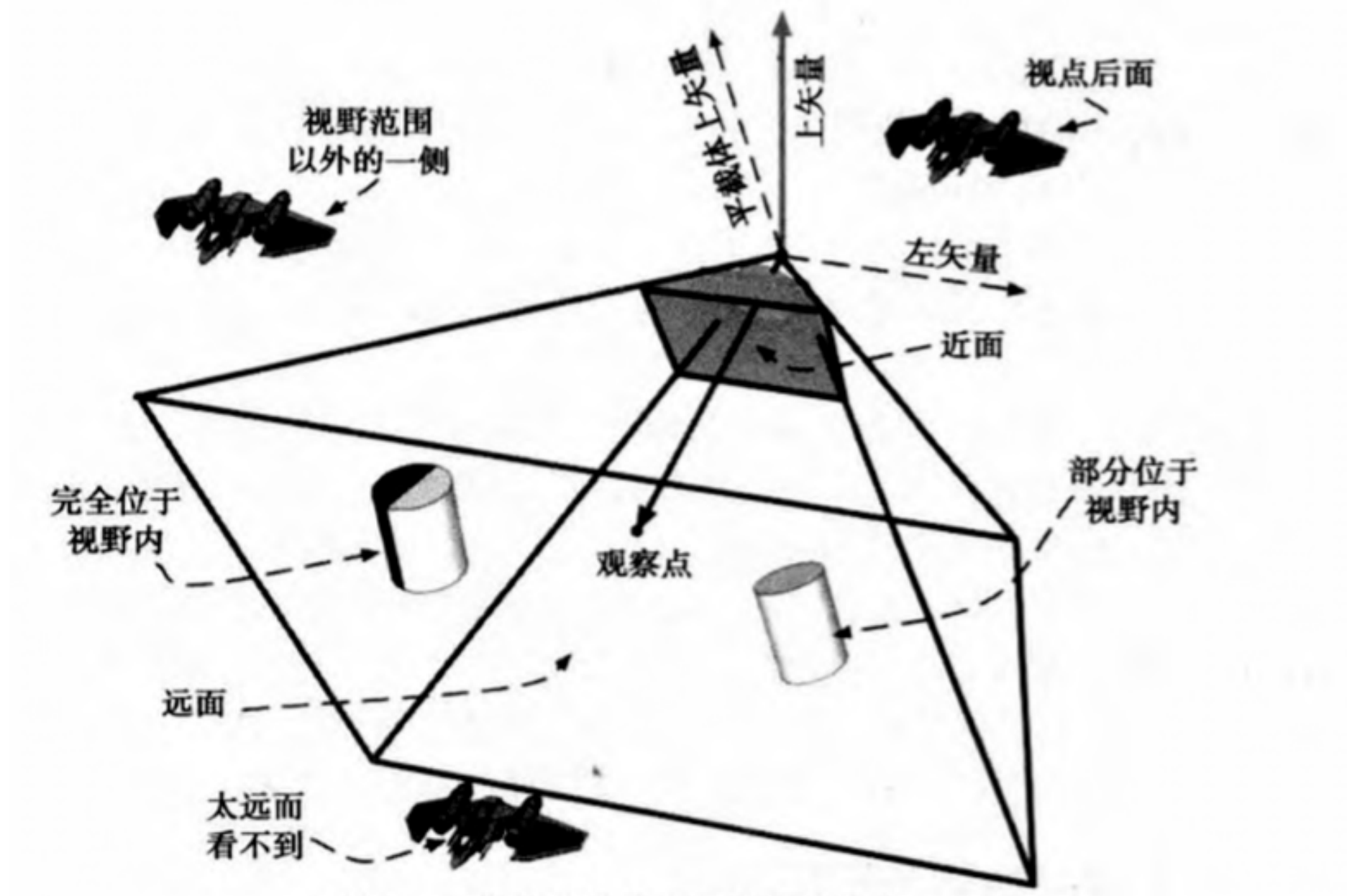
- 减少I/O
- 渲染更少的几何对象
- 减少内存访问

## 效果展示



## 核心思路

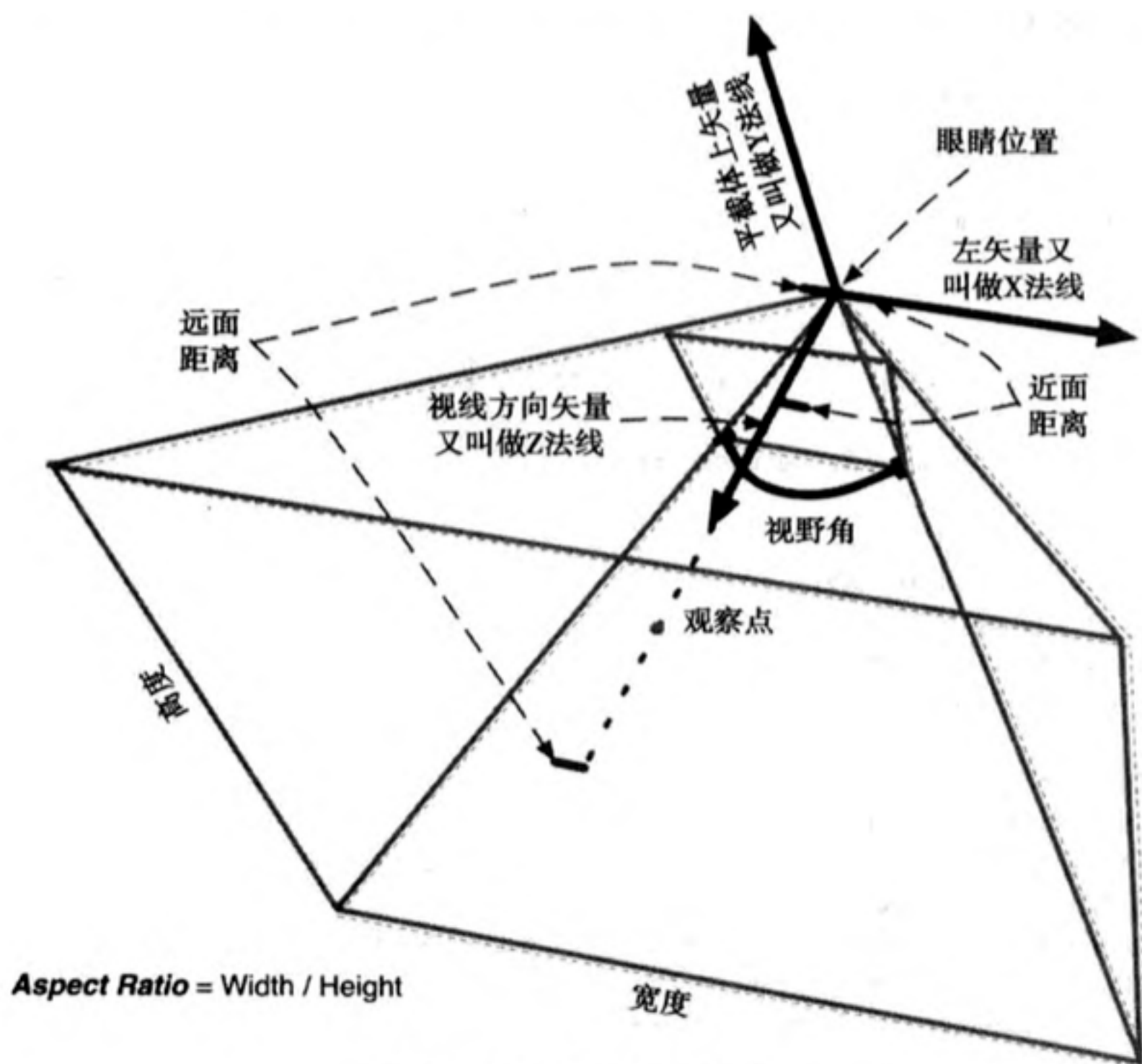
通过减少渲染的几何对象，在不影响显示效果的前提下，尽可能减少需要绘制的图元。在一个场景中，很多物体是处于平截体外部，这些物体是用户永远看不到的对象。



## 具体细节

### a.测试点是否在平截体内

计算眼睛到当前测试点的向量，提取这个向量关于平截体X、Y、Z轴的分量，分别进行判断。



- 1、计算眼睛到当前测试点的向量。

```
// eye到point的向量
const GLKVector3 eyeToPoint = GLKVector3Subtract(frustumPtr->eyePosition, point);
```

- 2、测试Z轴分量，这个分量要在区间[nearDistance, farDistance]。如果是，继续步骤3。

```
// z轴分量
const GLfloat pointZComponent = GLKVector3DotProduct(eyeToPoint, frustumPtr->
zUnitVector);

if(pointZComponent > frustumPtr->farDistance || pointZComponent < frustumPtr->
nearDistance)
{
    result = AGLKFrustumOut;
}
```

- 3、Y轴分量要小于被测点所在的平截体高度，高度的可以通过Z轴分量\*视角/2的正切值计算。如果在区间内，继续步骤4。

```
// y轴分量
const GLfloat pointYComponent =
GLKVector3DotProduct(eyeToPoint,
    frustumPtr->yUnitVector);
const GLfloat frustumHeightAtZ = pointZComponent * frustumPtr->tangentOfH
alfFieldOfView;

if(pointYComponent > frustumHeightAtZ || pointYComponent < -frustumHeight
AtZ)
{
    result = AGLKFrustumOut;
}
```

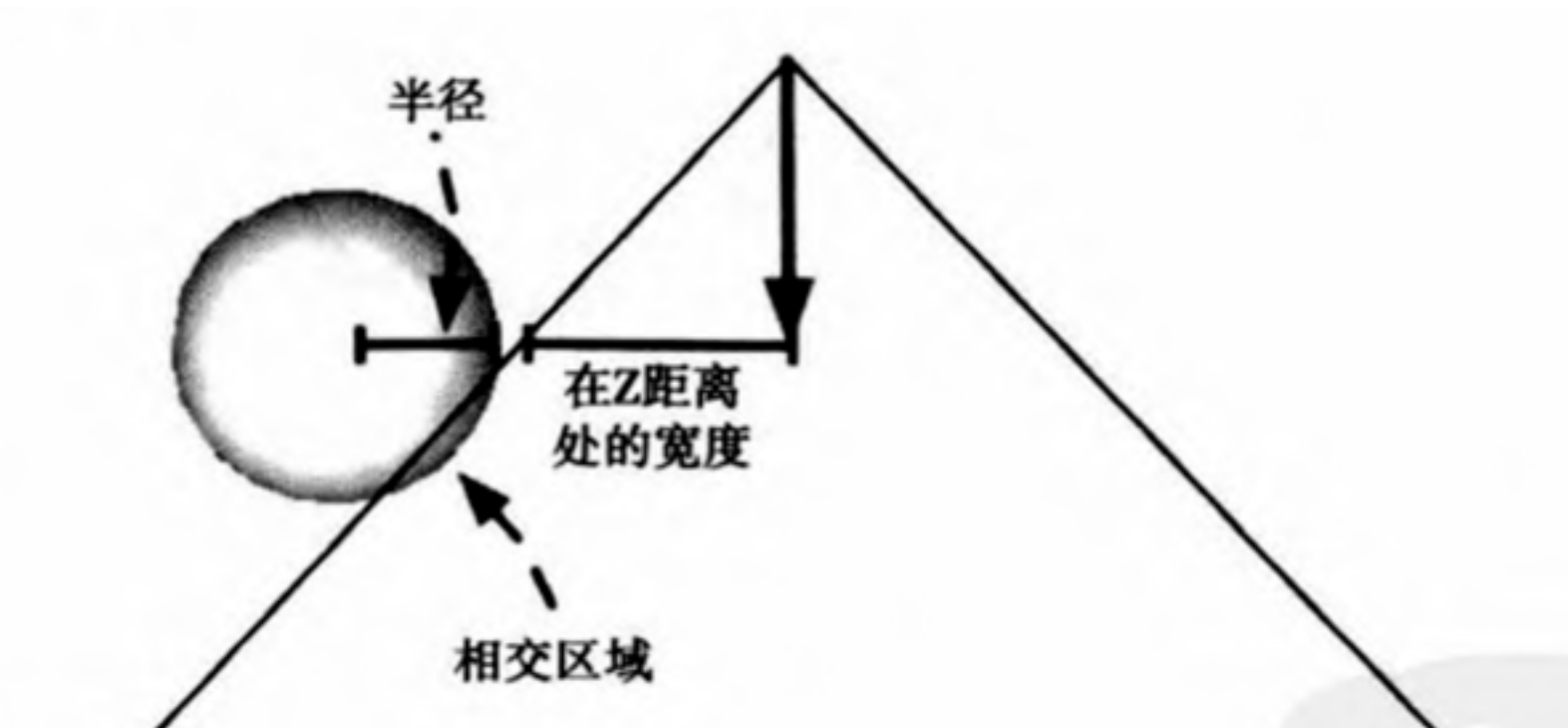
- 4、X轴分量要小于被测点锁在的平截体的宽度，宽度可以通过平截体高度值 \* 宽高比。

```
//X轴分量
const GLfloat pointXComponent =
GLKVector3DotProduct(eyeToPoint,
    frustumPtr->xUnitVector);
const GLfloat frustumWidthAtZ = frustumHeightAtZ *
frustumPtr->aspectRatio;

if(pointXComponent > frustumWidthAtZ ||
    pointXComponent < -frustumWidthAtZ)
{
    result = AGLKFrustumOut;
}
```

**b.判断球体是否在平截体内**

测试球体会测试点更复杂，同样是对比X/Y/Z轴分量，在判断的范围加上半径的距离。但是，考虑下面的情况

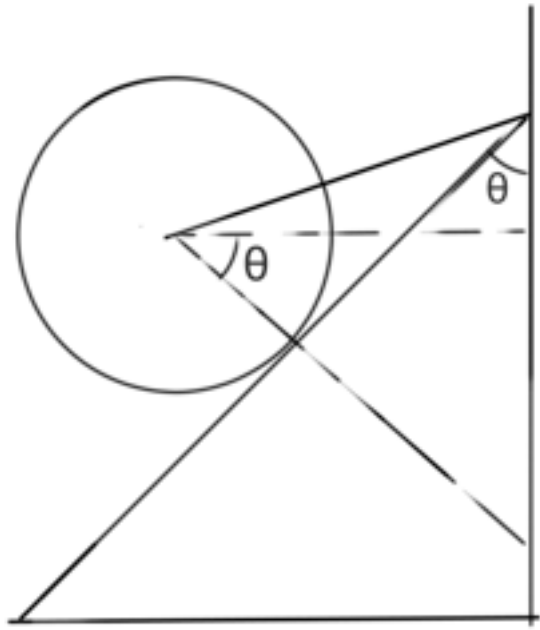


按照上面的判断，球体是在平截体之外，但是实际上是相交的。

**解决方案**

把半径乘以特定的因子。

如下图，考虑球体被外切情况，得出相应的放大因子。



- 1、Y轴因子

```
sphereFactorY = 1.0f/cosf(halfFieldOfViewRad);
```

- 2、X轴因子

```
const GLfloat angleX = atanf(frustumPtr->tangentOfHalfFieldOfView * aspectRatio);
frustumPtr->sphereFactorX = 1.0f/cosf(angleX);
```

## 扩展

- 场景图（scene graph）辅助剔除  
用树形数据结的几何对象层次组织。树形结构有一个根元素。根元素是子元素的父，子元素可能是其他元素的父。参考Cocoa的视图层次结构，2DUIView实例的场景图。同样的概念也使用与3D对象的层次结构。如果父元素在平截体外部，根据定义所有它的子元素也在平截体外部，没有必要再单独测试每个子元素。  
关键词：**Ochre** 八叉树。
- 减少缓存复制  
为GPU提供一个顶点属性缓存后，用CPU处理另一个。在所有渲染指令发送完后，通过glBindBuffer()函数来切换缓存。（苹果公司官网有例子，OpenGLApplicationDesign.html)
- 减少状态变换  
OpenGL ES上下文存储了大量的用于控制渲染运算的信息。信息缓存可能在CPU控制的内存，也可能在GPU的寄存器。  
调用glEnable(GL\_DEPTH\_TEST)多次会浪费时间更新上下文的状态，即使值是相同的。
- OES  
OES扩展是OpenGL ES标准的维护者，提出的一个非标准的扩展。

## 思考

### 为什么FPS会在20FPS和30FPS之间摆动？

绘制 和 显示 并不一样。

通过CADisplayLink（hardware generated），绘制的速率可能是60FPS。

如果绘制的时间超过1/60s，理论上帧率最多为30FPS。

想象一条1s的线段，分隔成60小段，每个小段的起点都可以作为绘制的起点。

如果绘制的时间超过1/60s，那么绘制的终点会延伸到第二个小段。

这样，一条1s的线段，最多有30个绘制的时间段。

Since CADisplayLink is hardware generated the only thing you can do with it is divide the time, that's what the frameInterval is there to do.

frameInterval = 1 gets you 60 fps

frameInterval = 2 gets you 30 fps

frameInterval = 3 gets you 20 fps

I use a lot frameInterval = 5 for menus for example, it still gives me 12fps (about the minimum for reasonable simple animation) and the battery consumption reduces drastically.

你能得到FPS，但是它不代表真正的性能，每帧持续时间是一个更佳选择。FPS不能线性评判性能表现。

For example, if your FPS goes from 30 to 20, then that's a pretty significant decrease in frame performance time (33 ms to 50 ms)

However if your FPS goes from 2000 to 400, that's a miniscule difference in real time (only 2ms difference).

最后，即使你自己通过自定义线程（不采用CADisplayLink），把绘制时间的空缺填补，实际上绘制的速率并不会变快。

## 总结

主要讲解的是数学部分的知识，OpenGL ES的部分没有引入新的技术点。

工作原因，以后更新会慢一些。能看到这里，你也是喜欢技术的，谢谢支持。来一波关注和喜欢如何 -> 我会加油更新。

附上源码 (<https://github.com/loyinglin/LearnOpenGL/tree/master/Tutorial10-%E5%B9%B3%E6%88%AA%E4%BD%93%E4%BC%98%E5%8C%96>)



落影loyinglin (/u/815d10a4bdce)

写了 171405 字, 被 4763 人关注, 获得了 2745 个喜欢

(/u/815d10a4bdce) 写了 171405 字, 被 4763 人关注, 获得了 2745 个喜欢

+ 关注

工程师一枚，喜欢思考，喜欢游戏，喜欢运动。做过什么已经不重要，未来的方向以及当下的准备是生活的...

♡ 喜欢 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-like-button)

16









更多分享

(<http://cwb.assets.jianshu.io/notes/images/3744602>)


被以下专题收入，发现更多相似内容




iOS学习 (/c/1332c736fe39?utm\_source=desktop&utm\_medium=notes-included-collection)






iOS Dev... (/c/3233d1a249ca?utm\_source=desktop&utm\_medium=notes-included-collection)




IOS技术归纳 (/c/d6982156df92?utm\_source=desktop&utm\_medium=notes-included-collection)



iOS 开发 (/c/2ffaa203eb6a?utm\_source=desktop&utm\_medium=notes-


- included-collection)
-  OpenGL ... (/c/044a5240577d?utm\_source=desktop&utm\_medium=notes-included-collection)
-  OPenGL ... (/c/408442c9c764?utm\_source=desktop&utm\_medium=notes-included-collection)
-  OpenGL+... (/c/5d2c87603bd3?utm\_source=desktop&utm\_medium=notes-included-collection)

展开更多

 登录/注册

为你个性化推荐内容

(/sign\_in?utm\_source=desktop&utm\_medium=notes-bottom-bind)

 下载简书App

随时随地发现和创作内容

(/app/download?utm\_source=desktop&utm\_medium=click-note-bottom-bind)