Logic_视觉班课堂笔记[CC007]

- **日期**: 2019年5月24日星期五
- **授课**: CC老师
- **课程次数**: 视觉班第7次课--共计(22次课)
- **主题**: OpenGL 主题

课程内容(纹理)

- 生成Mip贴图/各向异性同性过滤
- 案例(2)--隧道案例
- 案例(3)--球体世界(纹理填充)

课后作业:

1.完成隧道案例&球体世界(纹理填充)代码实现

隧道/球体世界代码基础.核心代码.

一.上节课回顾

- 纹理常用API
- 纹理坐标
- 金字塔案例(加光照/不加光照)

二.课程内容笔记

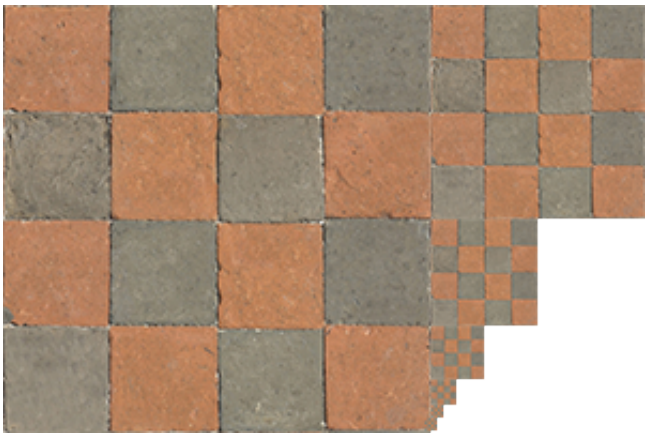
Mip贴图(多级渐远纹理)

什么是Mip贴图?
Mip贴图是一种功能强大的纹理技巧。它可以提高渲染性能同时可以改善场景的显示质量。

想象一下，假设我们有一个包含着上千物体的大房间，每个物体上都有纹理。有

些物体会很远，但其纹理会拥有与近处物体同样高的分辨率。由于远处的物体可能只产生很少的片段，OpenGL从高分辨率纹理中为这些片段获取正确的颜色值就很困难，因为它需要对一个跨过纹理很大部分的片段只拾取一个纹理颜色。在小物体上这会产生不真实的感觉，更不用说对它们使用高分辨率纹理浪费内存的问题了。

OpenGL使用一种叫做多级渐远纹理(Mipmap)的概念来解决这个问题，它简单来说就是一系列的纹理图像，后一个纹理图像是前一个的二分之一。多级渐远纹理背后的理念很简单：距观察者的距离超过一定的阈值，OpenGL会使用不同的多级渐远纹理，即最适合物体的距离的那个。由于距离远，解析度不高也不会被用户注意到。同时，多级渐远纹理另一加分之处是它的性能非常好。让我们看一下多级渐远纹理是什么样子的：



手工为每个纹理图像创建一系列多级渐远纹理很麻烦，幸好`OpenGL`有一个`glGenerateMipmaps`函数，在创建完一个纹理后调用它`就会承担接下来的所有工作了

常见问题：
1.闪烁问题，当屏幕上被渲染物体的表面与它所应用的纹理图像相比显得非常小时，就会出现闪烁效果。类似闪光，当纹理图像采样区域的移动幅度与它在屏幕大小相比显得不成比例时，也会发生这种现象。处于运动状态时，会比较容易看到闪烁的负面效果。

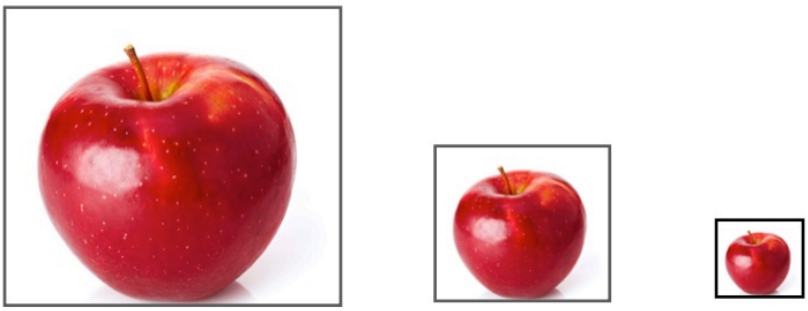
2.性能问题，加载大佬的纹理内存并对它们进行过滤处理，但屏幕上实际只是显示的只是很少一部分的片段。纹理越大，这个问题所造成的性能影响也很明显

这2种问题，我当然可以用很简单的方法解决，就是**使用更小的纹理图像**。但是这种解决方法又将产生一个新的问题，就是当一个物体更靠近观察者时，它必须渲染的比原来更大一些，这样，纹理就不得不拉伸。拉伸的结果，形成视觉效果很差的模糊或者斑驳状的纹理化效果。

从根本上解决方案，就是使用Mib贴图。不是单纯的把单个图像加载到纹理状态中，而是把一系列从最大到最小的图像加载到单个"Mib贴图"纹理状态。然后,OpenGL使用一组的新的过滤模式，为一个特定的几何图形选择具有最佳过滤效果的纹理。

Mib纹理由一系列的纹理图像组成，每个图像大小在每个轴的方向上都缩小一半。或者是原来图像像素的总是的四分之一。Mip贴图每个图像大小都依次减半，直到最后一个图像大小是1*1的纹理单元为止。

一系列Mip贴图图像



当我们在讲 `glTexParameteri`()函数时，谈到了 `level` 参数。 `level` 参数在 `Mib` 贴图发挥作用。因为它指定图像数据用于那个 `mip` 层。第一层是0，后面依次类推。如果 `Mib` 贴图未使用，那么只有第0层才会被加载。在默认情况下，为了能使用 `mip` 贴图，所以的 `mip` 层都要加载。我们可以通过设置 `GL_TEXTURE_BASE_LEVEL` 和 `GL_TEXTURE_MAX_LEVEL` 纹理参数特别设置需要使用的基层和最大层。

```
//设置mip贴图基层
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_BASE_LEVEL,0);
//设置mip贴图最大层
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAX_LEVEL,0);
```

我们虽然可以通过设置 `GL_TEXTURE_BASE_LEVEL` 和 `GL_TEXTURE_MAX_LEVEL` 纹理参数控制那些mip层被加载。但是我们仍然可以使用 `GL_TEXTURE_MIN_LOD` 和 `GL_TEXTURE_MAX_LOD` 参数限制已加载的Mip层的使用范围。

什么时候生成Mip贴图?

只有minFilter 等于以下四种模式，才可以生成Mip贴图

- `GL_NEAREST_MIPMAP_NEAREST` 具有非常好的性能，并且闪烁现象非常弱
- `GL_LINEAR_MIPMAP_NEAREST` 常常用于对游戏进行加速，它使用了高质量的线性过滤器
- `GL_LINEAR_MIPMAP_LINEAR` 和 `GL_NEAREST_MIPMAP_LINEAR` 过滤器在Mip层之间执行了一些额外的插值，以消除他们之间的过滤痕迹。
- `GL_LINEAR_MIPMAP_LINEAR` 三线性Mip贴图。纹理过滤的黄金准则，具有最高的精度。

```
if(minFilter == GL_LINEAR_MIPMAP_LINEAR ||
     minFilter == GL_LINEAR_MIPMAP_NEAREST ||
     minFilter == GL_NEAREST_MIPMAP_LINEAR ||
     minFilter == GL_NEAREST_MIPMAP_NEAREST)
  //4.纹理生成所有的Mip层
  //参数：GL_TEXTURE_1D、GL_TEXTURE_2D、GL_TEXTURE_3D
  glGenerateMipmap(GL_TEXTURE_2D);
```

glGenerateMipmap 函数解析

目的：为纹理对象生成一组完整的mipmap
void glGenerateMipmap（GLenum target）；

参数： 指定将生成mipmap的纹理对象绑定到的活动纹理单元的纹理目标.GL_TEXTURE_1D、GL_TEXTURE_2D、GL_TEXTURE_3D

描述： glGenerateMipmap计算从零级数组派生的一组完整的mipmap数组。 无论先前的内容如何，最多包括1x1维度纹理图像的数组级别都将替换为派生数组。 零级纹理图像保持不变（原图）。

派生的mipmap数组的内部格式都与零级纹理图像的内部格式相匹配。 通过将零级纹理图像的宽度和高度减半来计算派生数组的尺寸，然后将每个阵列级别的尺寸减半，直到达到1x1尺寸纹理图像。

Mip贴图过滤

经过Mip贴图的纹理过滤

常量	描述
GL_NEAREST	在Mip基层上执行最邻近过滤
GL_LINEAR	在Mip基层执行线性过滤
GL_NEAREST_MIPMAP_NEAREST	在最邻近Mip层，并执行最邻近过滤
GL_NEAREST_MIPMAP_LINEAR	在Mip层之间执行线性插补，并执行最邻近过滤
GL_LINEAR_MIPMAP_NEAREST	选择最邻近Mip层，并执行线性过滤
GL_LINEAR_MIPMAP_LINEAR	在Mip层之间执行线性插补，并执行线性过滤，又称三线性Mip贴图

一个常见的错误是，将放大过滤的选项设置为多级渐远纹理过滤选项之一。这样没有任何效果，因为多级渐远纹理主要是使用在纹理被缩小的情况下的：纹理放大不会使用多级渐远纹理，为放大过滤设置多级渐远纹理的选项会产生一个`GL_INVALID_ENUM`错误代码。

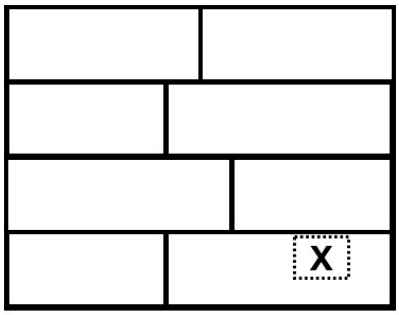
```
 //GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER(缩小过滤器)，GL_NEAREST（最邻近过滤）
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

//GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER(缩小过滤器)，GL_LINEAR（线性过滤）
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

 //GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER(缩小过滤器)，GL_NEAREST_MIPMAP_NEAREST（选择最邻近的Mip层，并执行最邻近过滤）
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST);

 //GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER(缩小过滤器)，GL_NEAREST_MIPMAP_LINEAR（在Mip层之间执行线性插补，并执行最邻近过滤）
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR);

//GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER(缩小过滤器)，GL_NEAREST_MIPMAP_LINEAR（选择最邻近Mip层，并执行线性过滤）
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_NEAREST);

//GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER(缩小过滤器)，GL_LINEAR_MIPMAP_LINEAR（在Mip层之间执行线性插补，并执行线性过滤，又称为三线性过滤）
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
```

各向异性过滤

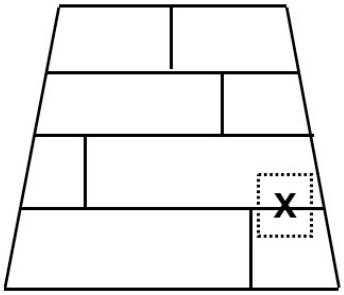
各向异性纹理过滤（`Anisotropic texture filteri`ng）并不是OpenGL 核心规范中的一部分。但它是一种得到广泛使用的扩展。可以极大提高纹理过滤操作的质量。

前面我讲过，2种基本过滤，最邻近过滤（`GL_NEAREST`）和线性过滤（`GL_LINEAR`）。当一个纹理贴图被过滤时，OpenGL 使用纹理坐标来判断一个特定的几何片段将落在纹理什么地方。然后，紧邻这个位置的纹理单使用`GL_NEAREST`和`GL_LINEAR`过滤操作进行采样。

我们开始的案例中，为什么会看上去显得模糊?
当几何图形进行纹理贴图时，如果它的观察方向和观察点恰好垂直。那么这个过程是相当完美的。如下图，



各向同性采样



各向异性采样

当我们从一个角度倾斜地观察这个几何图形时，对周围纹理单元进行常规采样，会导致一些纹理信息丢失（看上去显得模糊）.
为了更加逼真和准确的采样应该沿着包含纹理的平面方向进行延伸。如果我们进行处理纹理过滤时，考虑了观察角度，那么这个过滤方法就叫“各向异性过滤”。

在Mip贴图纹理过滤模型中，或者其他所有的基本纹理过滤我们都可以应用各向异性过滤。

应用各向异性过滤，需要2个步骤。

- 第一,查询得到支持的各向异性过滤的最大数量,可以使用`glGetFloatv`函数，并以`GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT`参数.

```
GLfloat flargest;
glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT,&fLargest);
```

- 第二,我们可以使用`glTexParameter`函数以及`GL_TEXTURE_MAX_ANISOTROPY_EXT`,设置各向异性过滤数据。

```
//设置纹理参数(各向异性采样)
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAX_ANISOTROPY_EXT,fLargest)
;
//设置各向同性过滤，数量为1.0表示(各向同性采样)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, 1.0f);
```

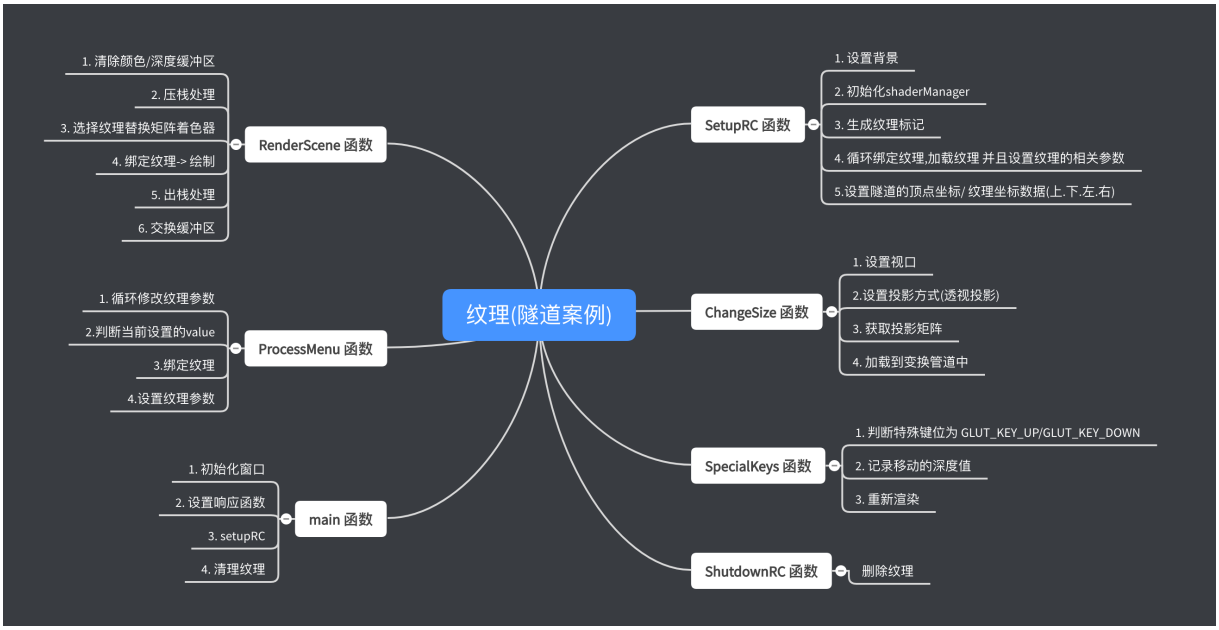
注意:

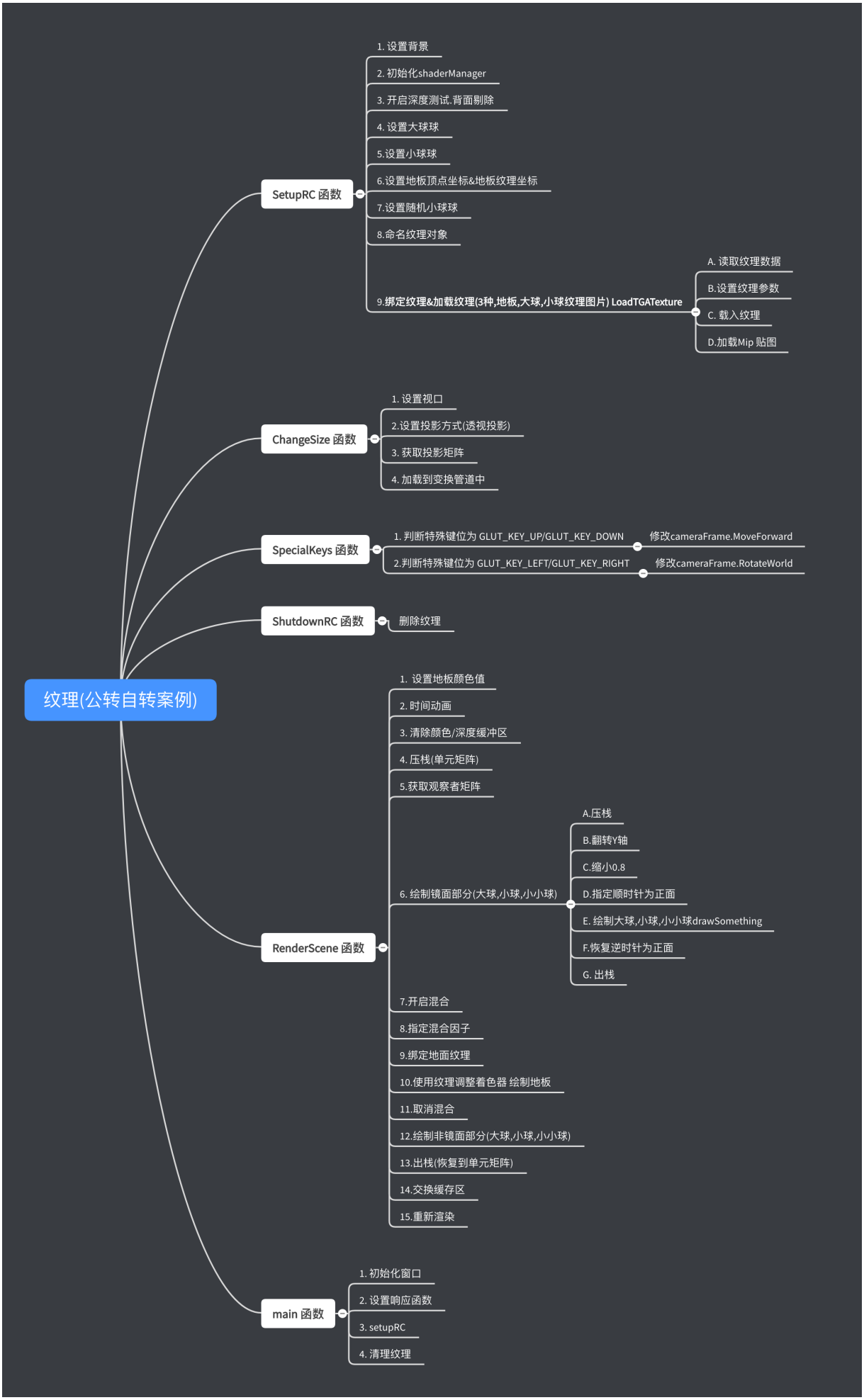
各向异性过滤所应用的数量越大，沿着最大变化方向（沿最强的观察点）所采样的纹理单元就越多。值1.0表示常规的纹理过滤（各向同性过滤）。

各向异性过滤，是会增加额外的工作，包括其他纹理单元。很可能对性能造成影响。但是，在现代硬件上，应用这个特性对速度造成影响不大。

最重要的是，目前它已经成为流行游戏、动画和模拟程序的一个标准特性。

三.课程总结




纹理(公转自转案例)
SetupRC 函数
1. 设置背景
2. 初始化shaderManager
3. 开启深度测试.背面剔除
4. 设置大球球
5.设置小球球
6.设置地板顶点坐标&地板纹理坐标
7.设置随机小球球
8.命名纹理对象
9.绑定纹理&加载纹理(3种,地板,大球,小球纹理图片) LoadTGATexture
A. 读取纹理数据
B.设置纹理参数
C. 载入纹理
D.加载Mip 贴图
ChangeSize 函数
1. 设置视口
2.设置投影方式(透视投影)
3. 获取投影矩阵
4. 加载到变换管道中
SpecialKeys 函数
1. 判断特殊键位为 GLUT_KEY_UP/GLUT_KEY_DOWN
修改cameraFrame.MoveForward
2.判断特殊键位为 GLUT_KEY_LEFT/GLUT_KEY_RIGHT
修改cameraFrame.RotateWorld
ShutdownRC 函数
删除纹理
RenderScene 函数
1. 设置地板颜色值
2. 时间动画
3. 清除颜色/深度缓冲区
4. 压栈(单元矩阵)
5.获取观察者矩阵
6. 绘制镜面部分(大球,小球,小小球)
A.压栈
B.翻转Y轴
C.缩小\0.8
D.指定顺时针为正面
E. 绘制大球,小球,小小球drawSomething
F.恢复逆时针为正面
G. 出栈
7.开启混合
8.指定混合因子
9.绑定地面纹理
10.使用纹理调整着色器 绘制地板
11.取消混合
12.绘制非镜面部分(大球,小球,小小球)
13.出栈(恢复到单元矩阵)
14.交换缓存区
15.重新渲染
main 函数
1. 初始化窗口
2. 设置响应函数
3. setupRC
4. 清理纹理

四.课程答疑

- OpenGL ES .
-