逻辑教育
Logic education

Hello CC

OpenGL 主题【7】

视觉班—OpenGL 基础纹理(下)

课程研发:CC老师
课程授课:CC老师

一.原始图像数据

1.像素包装

图像存储空间 = 图像的高度 * 图像宽度 * 每个像素的字节数 ？

二.认识函数

```
//改变像素存储方式
void glPixelStorei(GLenum pname,GLint param);
//恢复像素存储方式
void glPixelStoref(GLenum pname,GLfloat param);

//举例:
//参数1: GL_UNPACK_ALIGNMENT 指定OpenGL 如何从数据缓存区中解包图像数据
//参数2: 表示参数GL_UNPACK_ALIGNMENT 设置的值

//GL_UNPACK_ALIGNMENT 指内存中每个像素行起点的排列请求，允许设置为1（byte排列）、2（排列为偶数byte的行）、4（字word排列）、8（行从双字节边界开始）
glPixelStorei(GL_UNPACK_ALIGNMENT,1);
```

课程研发:CC老师
课程授课:CC老师

2.2 . 认识函数 从颜色缓存区内容作为像素图直接读取

```
//参数1: x,矩形左下角的窗口坐标
//参数2: y,矩形左下角的窗口坐标
//参数3: width,矩形的宽，以像素为单位
//参数4: height,矩形的高，以像素为单位
//参数5: format,OpenGL 的像素格式，参考 表6-1
//参数6: type,解释参数pixels指向的数据，告诉OpenGL 使用缓存区中的什么
数据类型来存储颜色分量，像素数据的数据类型，参考 表6-2
//参数7: pixels,指向图形数据的指针
void glReadPixels(GLint x,GLint y,GLSizei width,GLSizei
height, GLenum format, GLenum type,const void * pixels);
```

```
glReadBuffer(mode);—> 指定读取的缓存
glWriteBuffer(mode);—> 指定写入的缓存
```

课程研发:CC老师
课程授课:CC老师

2.2 . 认识函数 从TGA文件中读取像素图

```
GLbyte *gltReadTGABits(const char *szFileName, GLint *iWidth, GLint *iHeight, GLint
*iComponents, GLenum *eFormat);
```

参数1： 纹理文件名称
参数2： 文件宽度地址
参数3：文件高度地址
参数4：文件组件地址
参数5：文件格式地址
返回值：pBits,指向图像数据的指针

课程研发:CC老师
课程授课:CC老师

2.3 . 认识函数 载入纹理

```
void glTexImage1D(GLenum target,GLint level,GLint
internalformat,GLsizei width,GLint border,GLenum
format,GLenum type,void *data);

void glTexImage2D(GLenum target,GLint level,GLint
internalformat,GLsizei width,GLsizei height,GLint
border,GLenum format,GLenum type,void * data);

void glTexImage3D(GLenum target,GLint level,GLint
internalformat,GLSizei width,GLsizei height,GLsizei
depth,GLint border,GLenum format,GLenum type,void *data);
```

* target:`GL_TEXTURE_1D`、`GL_TEXTURE_2D`、`GL_TEXTURE_3D`。
* Level:指定所加载的mip贴图层次。一般我们都把这个参数设置为0。
* internalformat:每个纹理单元中存储多少颜色成分。
* width、height、depth参数:指加载纹理的宽度、高度、深度。==注意！==这些值必须是2的整数次方。（这是因为OpenGL 旧版本上的遗留下的一个要求。当然现在已经可以支持不是2的整数次方。但是开发者们还是习惯使用以2的整数次方去设置这些参数。）
* border参数:允许为纹理贴图指定一个边界宽度。
* format、type、data参数：与我们在讲glDrawPixels 函数对于的参数相同

2.4 更新纹理

```
void glTexSubImage1D(GLenum target,GLint level,GLint xOffset,GLsizei width,GLenum
format,GLenum type,const GLvoid *data);

void glTexSubImage2D(GLenum target,GLint level,GLint xOffset,GLint yOffset,GLsizei
width,GLsizei height,GLenum format,GLenum type,const GLvoid *data);

void glTexSubImage3D(GLenum target,GLint level,GLint xOffset,GLint yOffset,GLint
zOffset,GLsizei width,GLsizei height,GLsizei depth,Glenum type,const GLvoid * data);
```

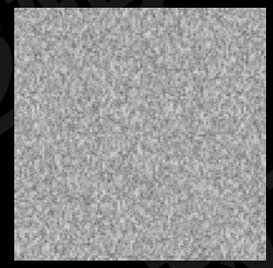
2.5 插入替换纹理

```
void glCopyTexSubImage1D(GLenum target,GLint level,GLint xoffset,GLint x,GLint y,GLsizei
width);

void glCopyTexSubImage2D(GLenum target,GLint level,GLint xoffset,GLint yOffset,GLint x,GLin
y,GLsizei width,GLsizei height);

void glCopyTexSubImage3D(GLenum target,GLint level,GLint xoffset,GLint yOffset,GLint
zOffset,GLint x,GLint y,GLsizei width,GLsizei height);
```

课程研发:CC老师
课程授课:CC老师

2.3 . 认识函数 使用颜色缓存区加载数据,形成新的纹理使用

```
void glCopyTexImage1D(GLenum target,GLint level,GLenum
internalformt,GLint x,GLint y,GLsizei width,GLint border);

void glCopyTexImage2D(GLenum target,GLint level,GLenum
internalformt,GLint x,GLint y,GLsizei width,GLsizei
height,GLint border);
```



x,y 在颜色缓存区中指定了开始读取纹理数据的位置；
缓存区里的数据，是源缓存区通过glReadBuffer设置的。

注意：不存在glCopyTextImage3D ，因为我们无法从2D 颜色缓存区中获取体积数据。

课程研发:CC老师
课程授课:CC老师

3.0 纹理对象

```
//使用函数分配纹理对象
//指定纹理对象的数量 和 指针（指针指向一个无符号整形数组，由纹理对象标识符填充）。
void glGenTextures(GLsizei n,GLuint * textTures);

//绑定纹理状态
//参数target:GL_TEXTURE_1D、GL_TEXTURE_2D、GL_TEXTURE_3D
//参数texture:需要绑定的纹理对象
void glBindTexture(GLenum target,GLunit texture);

//删除绑定纹理对象
//纹理对象 以及 纹理对象指针（指针指向一个无符号整形数组，由纹理对象标识符填充）。
void glDeleteTextures(GLsizei n,GLuint *textures);

//测试纹理对象是否有效
//如果texture是一个已经分配空间的纹理对象，那么这个函数会返回GL_TRUE,否则会返回GL_FALSE。
GLboolean glIsTexture(GLuint texture);
```

课程研发:CC老师
课程授课:CC老师

3.1 设置纹理参数

```
glTexParameterf(GLenum target,GLenum pname,GLFloat param);
glTexParameteri(GLenum target,GLenum pname,GLint param);
glTexParameterfv(GLenum target,GLenum pname,GLFloat *param);
glTexParameteriv(GLenum target,GLenum pname,GLint *param);
```

参数1:target,指定这些参数将要应用在那个纹理模式上，比如GL_TEXTURE_1D、GL_TEXTURE_2D、GL_TEXTURE_3D。
参数2:pname,指定需要设置那个纹理参数
参数3:param,设定特定的纹理参数的值

3.2 设置过滤方式

邻近过滤(GL_NEAREST)

线性过滤(GL_LINEAR)

课程研发:CC老师
课程授课:CC老师

2种纹理过滤方式比较

课程研发:CC老师
课程授课:CC老师

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST)
纹理缩小时,使用邻近过滤

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR)
纹理放大时,使用线性过滤

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
```

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

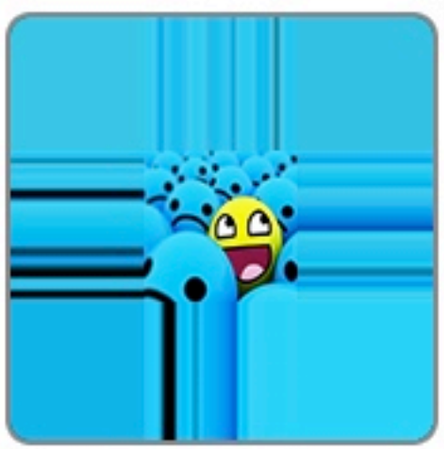
3.3 设置环绕方式

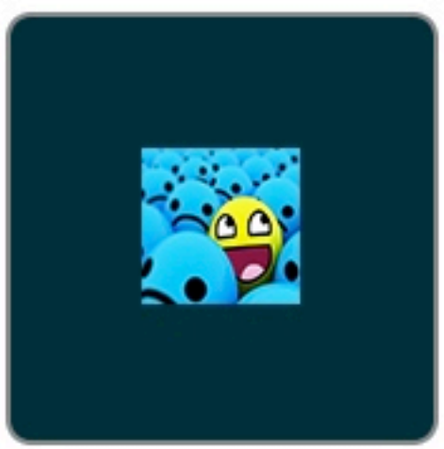
环绕方式	描述
GL_REPEAT	对纹理的默认行为。重复纹理图像。
GL_MIRRORED_REPEAT	和GL_REPEAT一样，但每次重复图片是镜像放置的。
GL_CLAMP_TO_EDGE	纹理坐标会被约束在0到1之间，超出的部分会重复纹理坐标的边缘，产生一种边缘被拉伸的效果。
GL_CLAMP_TO_BORDER	超出的坐标为用户指定的边缘颜色。

当纹理坐标超出默认范围时，每个选项都有不同的视觉效果输出。我们来看看这些纹理图像的例子：


GL_REPEAT


GL_MIRRORED_REPEAT


GL_CLAMP_TO_EDGE


GL_CLAMP_TO_BORDER

课程授课:CC老师

3.3 设置环绕方式

参数1：GL_TEXTURE_1D、GL_TEXTURE_2D、GL_TEXTURE_3D
参数2：GL_TEXTURE_WRAP_S、GL_TEXTURE_T、GL_TEXTURE_R,针对s,t,r坐标
参数3：GL_REPEAT、GL_CLAMP、GL_CLAMP_TO_EDGE、GL_CLAMP_TO_BORDER
GL_REPEAT:OpenGL 在纹理坐标超过1.0的方向上对纹理进行重复；
GL_CLAMP:所需的纹理单元取自纹理边界或TEXTURE_BORDER_COLOR.
GL_CLAMP_TO_EDGE环绕模式强制对范围之外的纹理坐标沿着合法的纹理单元的最后一行或者最后一列来进行采样。
GL_CLAMP_TO_BORDER:在纹理坐标在0.0到1.0范围之外的只使用边界纹理单元。边界纹理单元是作为围绕基本图像的额外的行和列，并与基本纹理图像一起加载的。

```
glTextParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAR_S,GL_CLAMP_TO_EDGE);
glTextParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAR_T,GL_CLAMP_TO_EDGE);
```

课程研发:CC老师
课程授课:CC老师

表6-1 OpenGL 像素格式

常量	描述
GL_RGB	描述红、绿、蓝顺序排列的颜色
GL_RGBA	按照红、绿、蓝、Alpha顺序排列的颜色
GL_BGR	按照蓝、绿、红顺序排列颜色
GL_BGRA	按照蓝、绿、红、Alpha顺序排列颜色
GL_RED	每个像素只包含了一个红色分量
GL_GREEN	每个像素只包含了一个绿色分量
GL_BLUE	每个像素只包含了一个蓝色分量
GL_RG	每个像素依次包含了一个红色和绿色的分量
GL_RED_INTEGER	每个像素包含了一个整数形式的红色分量
GL_GREEN_INTEGER	每个像素包含了一个整数形式的绿色分量
GL_BLUE_INTEGER	每个像素包含了一个整数形式的蓝色色分量
GL_RG_INTEGER	每个像素依次包含了一个整数形式的红色、绿色分量
GL_RGB_INTEGER	每个像素包含了一个整数形式的红色、蓝色、绿色分量
GL_RGBA_INTEGER	每个像素包含了一个整数形式的红色、蓝色、绿色、Alpah分
GL_BGR_INTEGER	每个像素包含了一个整数形式的蓝色、绿色、红色分量
GL_BGRA_INTEGER	每个像素包含了一个整数形式的蓝色、绿色、红色、Alpah分
GL_STENCIL_INDEX	每个像素只包含了一个模板值
GL_DEPTH_COMPONENT	每个像素值包含一个深度值
GL_DEPTH_STENCIL	每个像素包含一个深度值和一个模板值

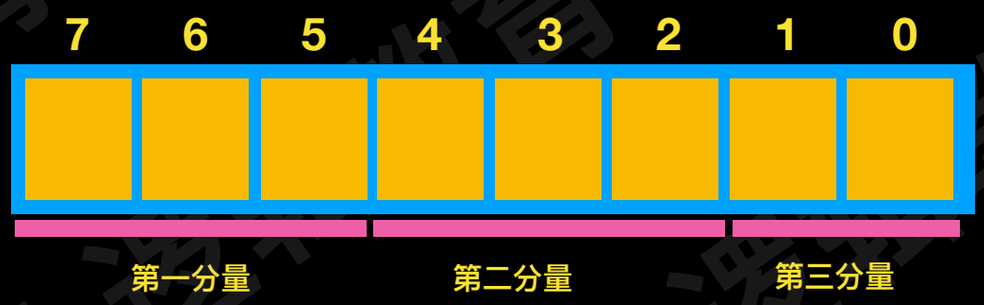
课程研发:CC老师
课程授课:CC老师

表6-2 像素数据的数据类型

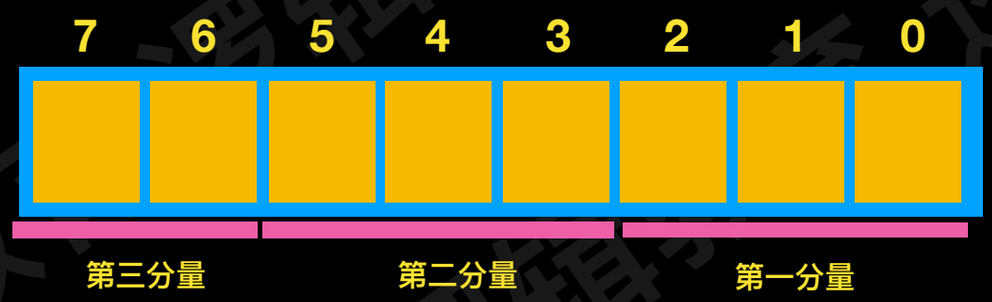
常量	描述
GL_UNSIGNED_BYTE	每种颜色分量都是一个8位无符号整数
GL_BYTE	8位有符号整数
GL_UNSIGNED_SHORT	16位无符号整数
GL_SHORT	16位有符号整数
GL_UNSIGNED_INT	32位无符号整数
GL_INT	32位有符号整数
GL_FLOAT	单精度浮点数
GL_HALF_FLOAT	半精度浮点数
GL_UNSIGNED_BYTE_3_2_2	包装的RGB值
GL_UNSIGNED_BYTE_2_3_3_REV	包装的RGB值
GL_UNSIGNED_SHORT_5_6_5	包装的RGB值
GL_UNSIGNED_SHORT_5_6_5_REV	包装的RGB值
GL_UNSIGNED_SHORT_4_4_4_4	包装的RGB值
GL_UNSIGNED_SHORT_4_4_4_4_REV	包装的RGB值
GL_UNSIGNED_SHORT_5_5_5_1	包装的RGB值
GL_UNSIGNED_SHORT_1_5_5_5_REV	包装的RGB值
GL_UNSIGNED_INT_8_8_8_8	包装的RGB值
GL_UNSIGNED_INT_8_8_8_8_REV	包装的RGB值
GL_UNSIGNED_INT_10_10_10_2	包装的RGB值
GL_UNSIGNED_INT_2_10_10_10_REV	包装的RGB值
GL_UNSIGNED_INT_24_8	包装的RGB值
GL_UNSIGNED_INT_10F_11F_REV	包装的RGB值
GL_FLOAT_24_UNSIGNED_INT_24_8_REV	包装的RGB值

课程研发:CC老师
课程授课:CC老师

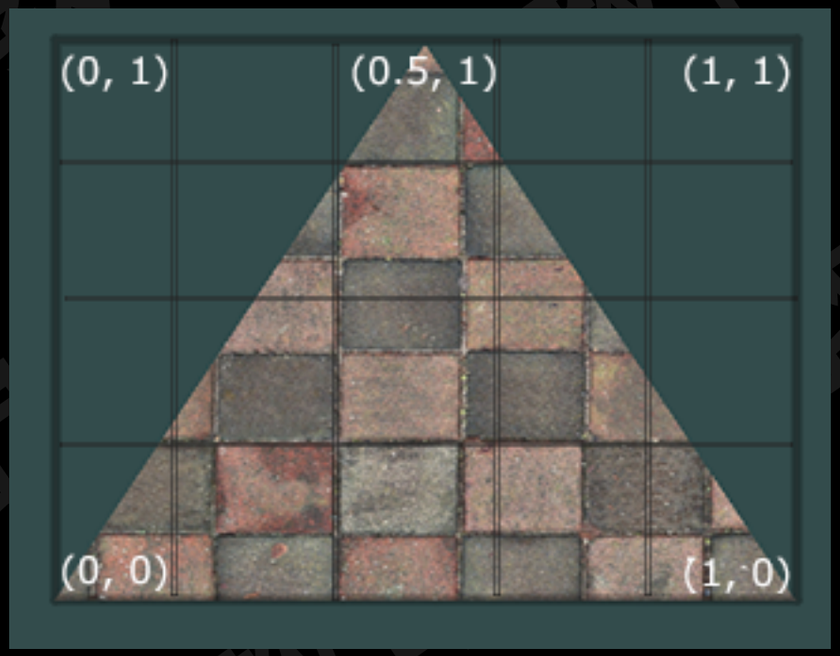
UNSIGNED_BYTE_3_3_2



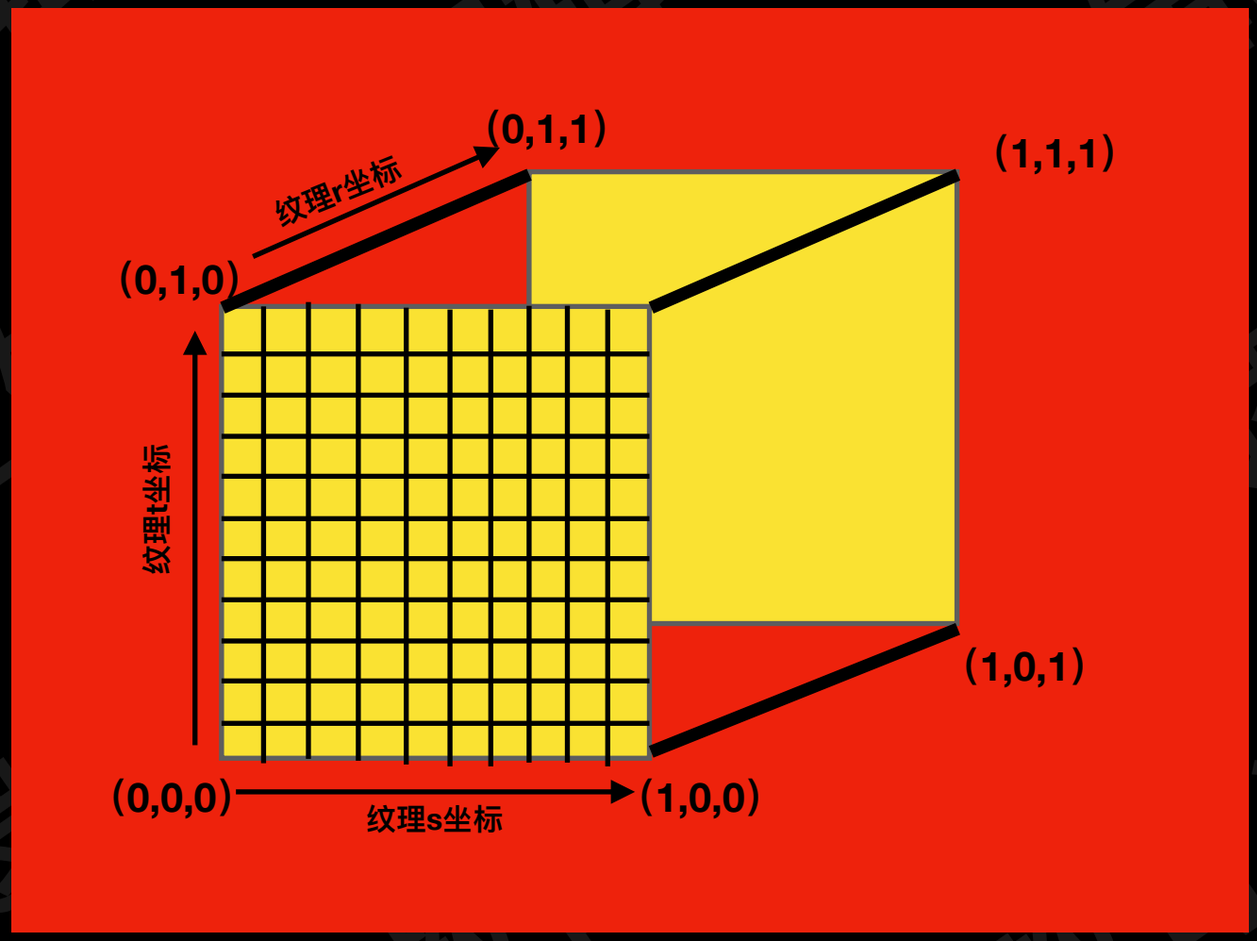
UNSIGNED_BYTE_2_3_3_REV

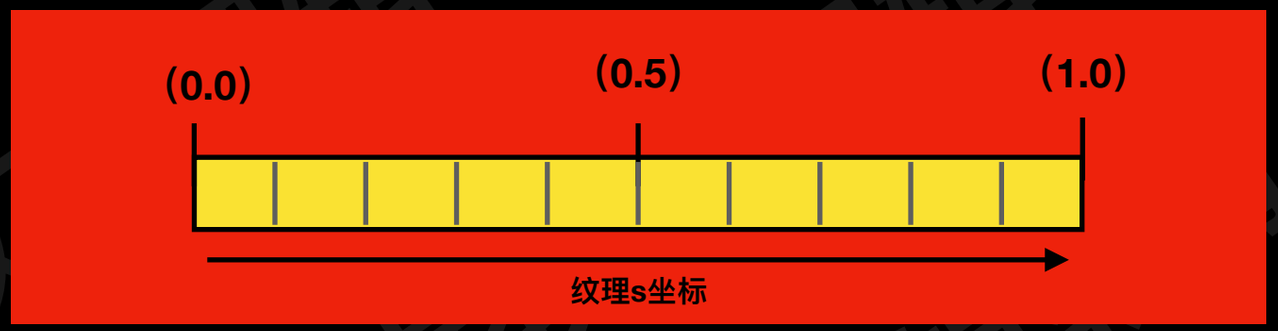


指定分量RGBA的排列顺序根据format参数确定。分量是按照分量高位到低位排列

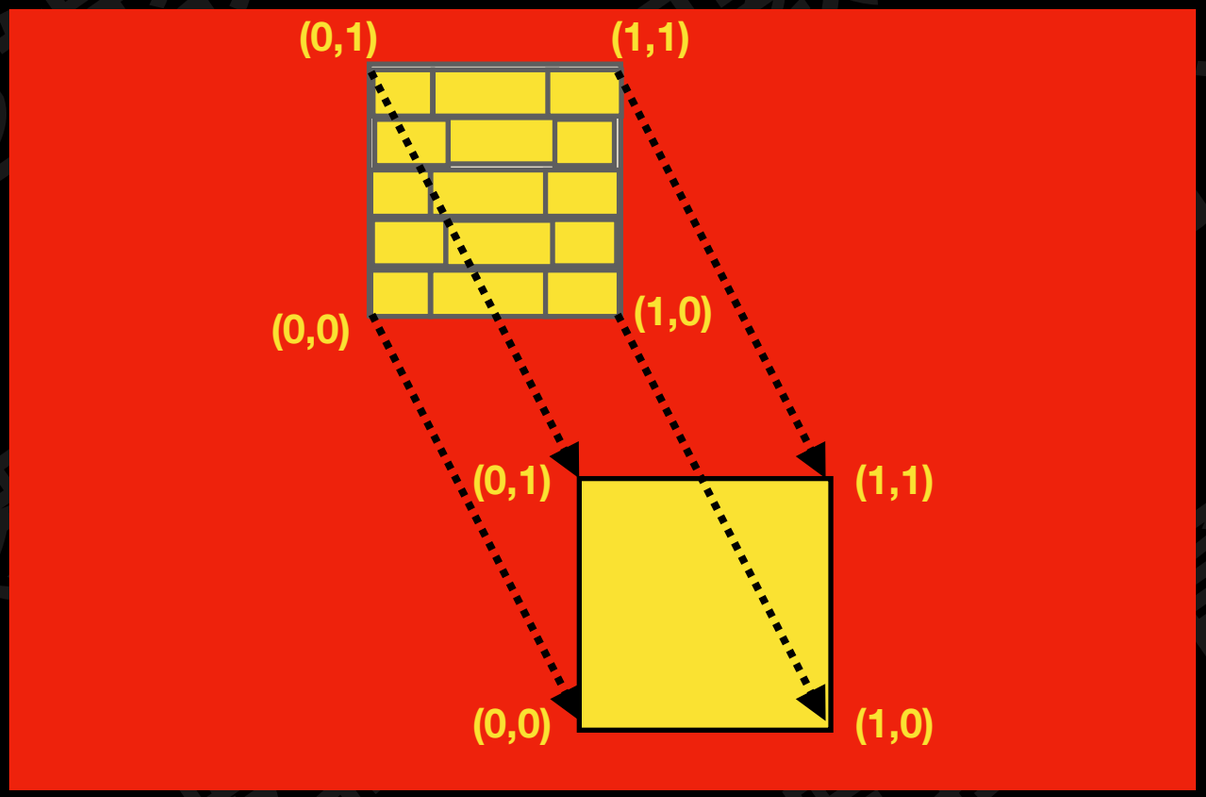


课程研发:CC老师
课程授课:CC老师



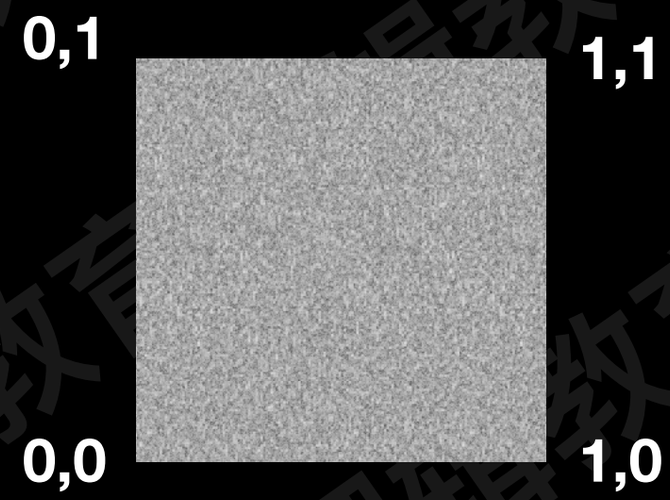


课程研发:CC老师
课程授课:CC老师



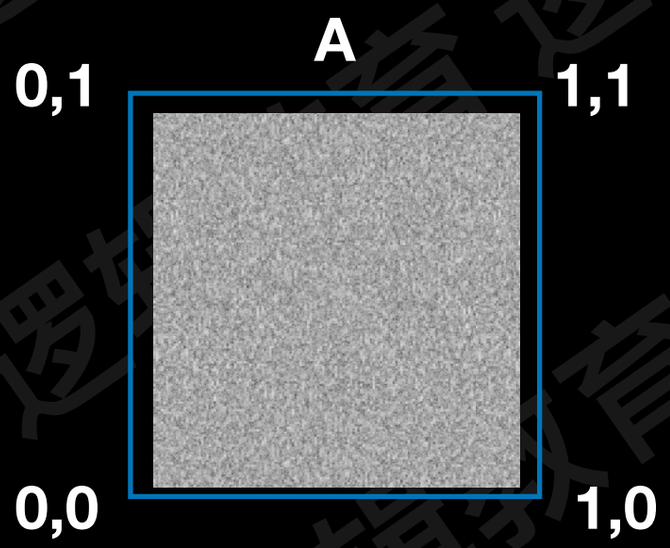
课程研发:CC老师
课程授课:CC老师

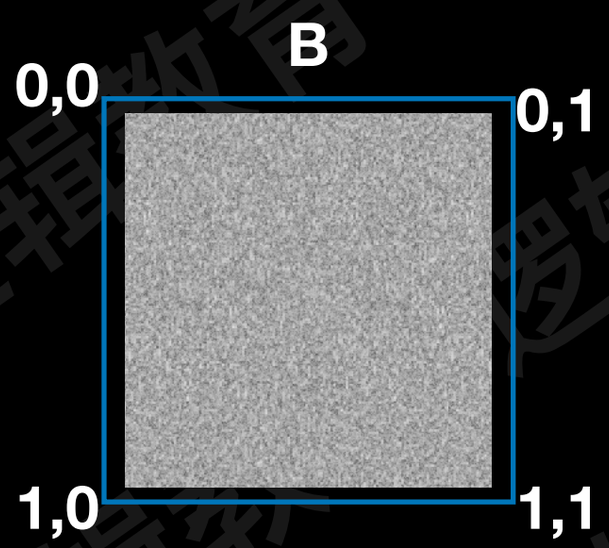
纹理坐标!

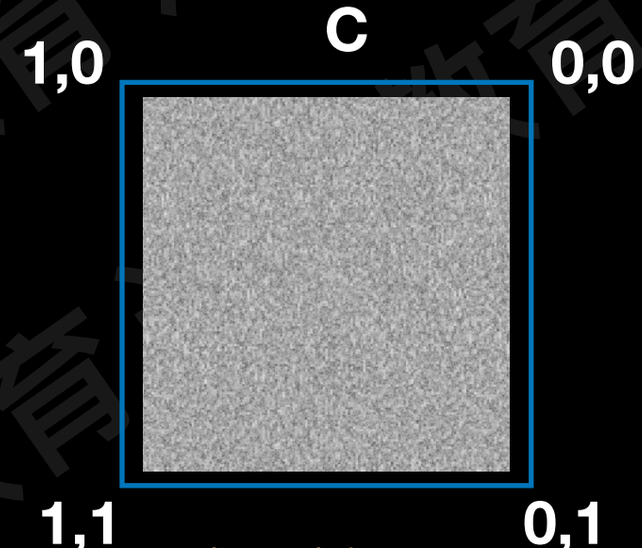


以下纹理在贴图时的映射关系. 简单说,就是一张贴纸,你是可以随意的贴.但是不能让图片交叉.

下图,演示了3种,纹理填充到正方形上,填充方式.







课程研发:CC老师
课程授课:CC老师

纹理坐标!

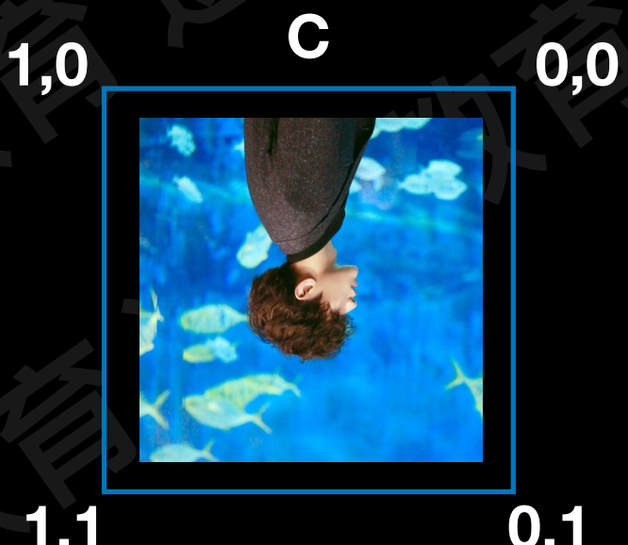
以下纹理在贴图时的映射关系. 简单说,就是一张贴纸,你是可以随意的贴.但是不能让图片交叉.

下图,演示了3种,纹理填充到正方形上,填充方式.

A

B

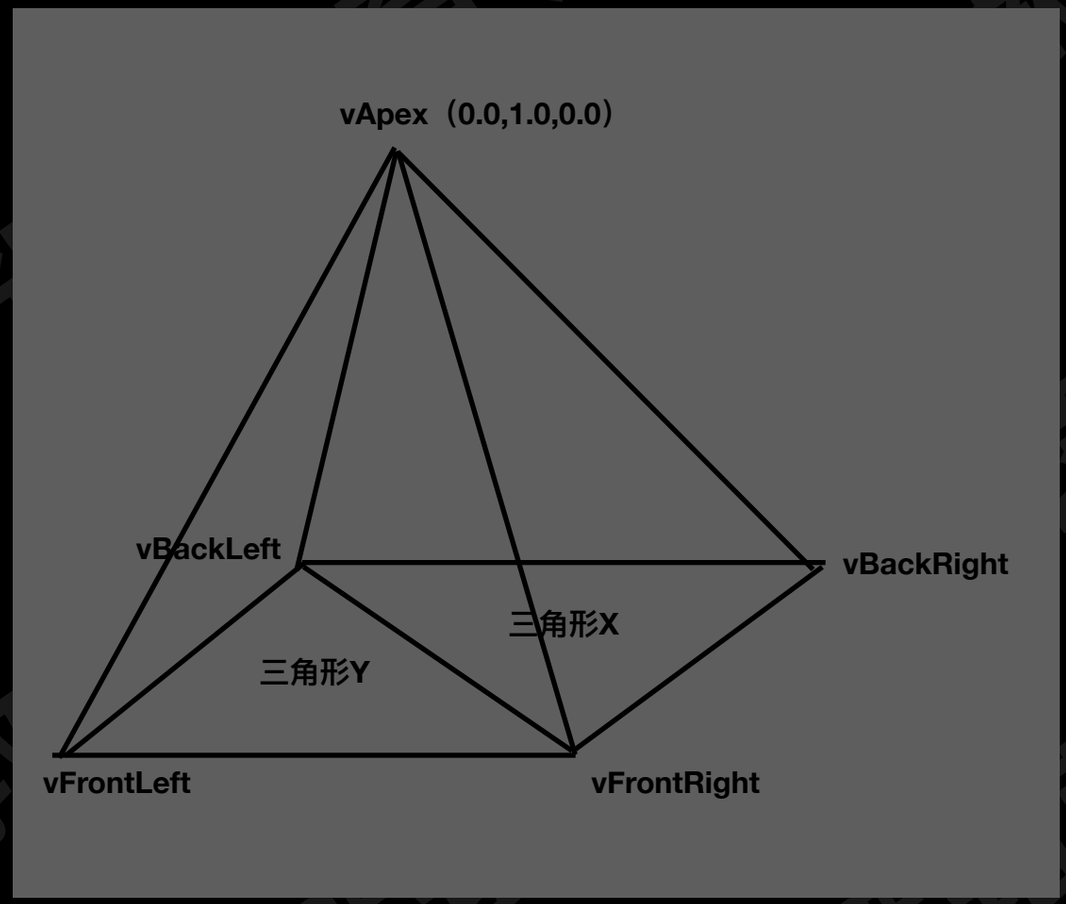
C



课程研发:CC老师
课程授课:CC老师

金字塔图形解析：

底部四边形 = 三角形X + 三角形Y



顶点坐标

三角形X的坐标如下：
vBackLeft(-1.0,-1.0,-1.0)
vBackRight(1.0,-1.0,-1.0)
vFrontRight(1.0,-1.0,1.0)
三角形Y的坐标如下：
vFrontLeft(-1.0,-1.0,1.0)
vBackLeft(-1.0,-1.0,-1.0)
vFrontRight(1.0,-1.0,1.0)

纹理坐标

三角形X的2D纹理坐标如下：
vBackLeft(0.0,0.0,0.0)
vBackRight(0.0,1.0,0.0)
vFrontRight(0.0,1.0,1.0)
三角形Y的2D纹理坐标如下：
vFrontLeft(0.0,0.0,1.0)
vBackLeft(0.0,0.0,0.0)
vFrontRight(0.0,1.0,1.0)

纹理坐标参考图6-6理解

课程研发:CC老师
课程授课:CC老师

Mip 贴图

与其他场景中的对象一样，纹理对象也可以从不同的视点距离进行观察。在一个动态的场景中，当贴了纹理的物体远离视点运动时，屏幕像素与纹理纹素之间的比率会变得非常低，因此纹理的采样频率也会变得非常低。这样会产生渲染图像上的瑕疵，因为有纹理数据的下采样（undersampling）的缘故。举例来说，如果要渲染一面砖墙，可能会用到一张很大的纹理图像（比如 1024 × 1024 个纹素），在观察者距离墙很近的时候这样是没问题的。但是如果这面墙正在远离观察者运动，直到它在屏幕上变成了一个像素点，那么纹理采样的结果可能会在某个过渡点上发生突然的变化。

为了降低这个效果的影响，可以对纹理贴图进行提前滤波，并且将滤波后的图像存储为连续的低分辨率的版本（原始图像为全分辨率）。这就叫做 mipmap，如图 6-17 所示。

mipmap 这个名词是 Lance Williams 创造的，他在自己的论文《Pyramidal Parametrics》（SIGGRAPH 1983 Proceedings）中介绍了这一概念。mip 来自于拉丁语 multum in parvo，也就是“在一个小区域的很多东西”的意思。mipmap 技术使用了一些聪明的做法将图像数据打包到内存中。

原始图像

节自<OpenGL 编程指南第9版 >—第6章纹理与帧缓存

Mip 贴图

打包到内存中。

OpenGL 在使用 mipmap 的时候会自动判断当前应当使用纹理贴图的哪个分辨率层级，这是基于被映射的物体的尺寸（像素为单位）来决定的。通过这种方法，我们可以根据纹理贴图的细节层次（level of detail），找到当前绘制到屏幕的最合适的图像；如果物体的图像变得更小，那么纹理贴图层次的尺寸也会减小。mipmap 需要一些额外的计算和纹理存储的区域。不过如果我们不使用 mipmap 的话，纹理被映射到较小的运动物体上之后可能会产生闪烁的问题。

我们介绍 OpenGL mipmap 的时候，并没有讨论纹素尺寸和多边形尺寸之间的缩放参数（被称作 λ）。此外还假设 mipmap 的所有参数都使用了默认值。我们可以在 6.11.3 节中了解有关 λ 的解释，以及 mipmap 参数的效果。更多有关用户程序如何控制 λ 的方法可以在 6.11.4 节中了解。

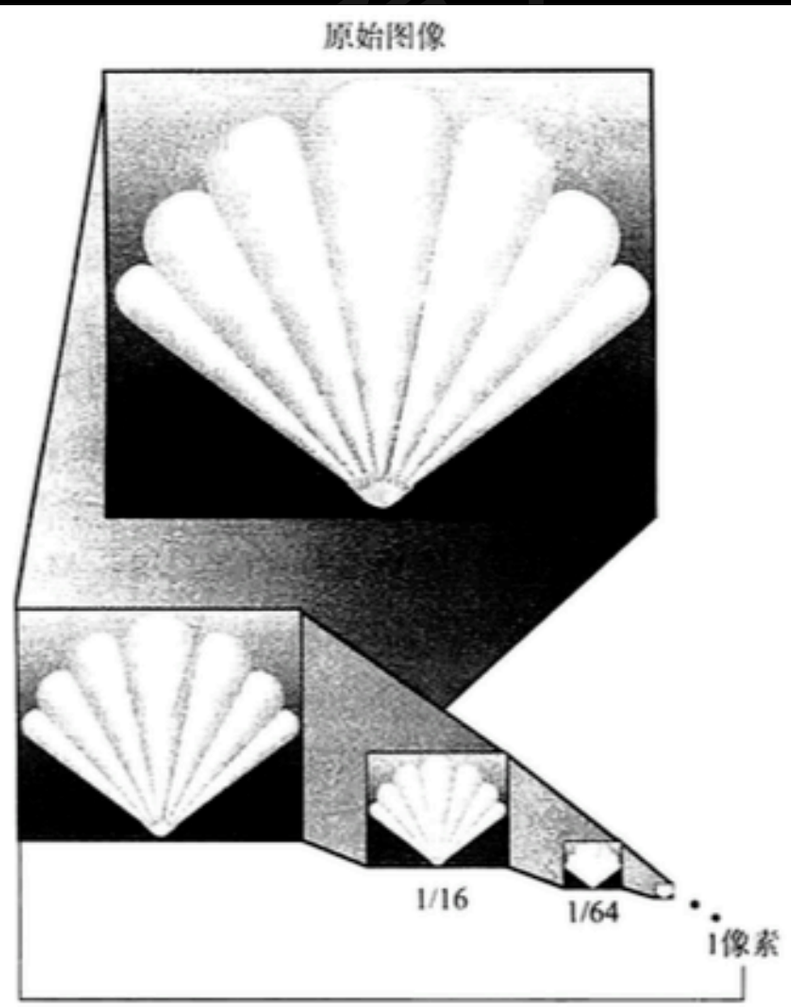


图 6-17　预先滤波后的 mipmap 金字塔

节自<OpenGL 编程指南第9版 >—第6章纹理与帧缓存

Mip 贴图

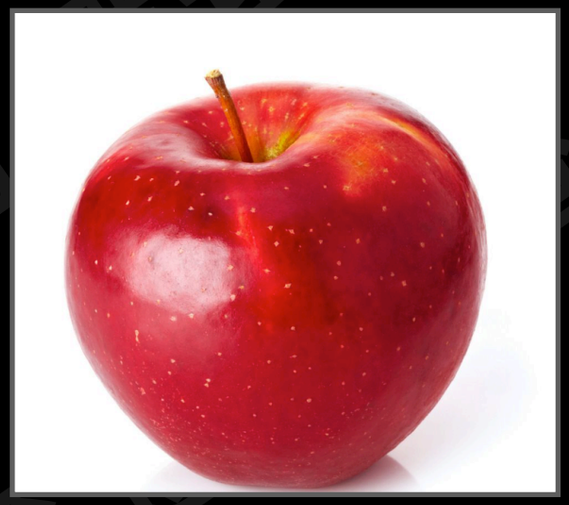
参数 GL_TEXTURE_MIN_FILTER 负责控制 mipmap 层次大于 0 的时候，纹素构建的方式。这个参数总共有 6 个可以设置的参数值。前 2 个值和纹理放大（magnification）的参数值是相同的：GL_NEAREST 和 GL_LINEAR。选择这 2 个参数，OpenGL 会关闭 mipmap 并且仅仅使用纹理的基本层（0 层）。其他 4 个参数分别是：GL_NEAREST_MIPMAP_NEAREST、GL_NEAREST_MIPMAP_LINEAR、GL_LINEAR_MIPMAP_NEAREST、GL_LINEAR_MIPMAP_LINEAR。注意每个模式参数都是由两部分组成的，名称的结构总是 GL_{A}_MIMPMAP_{B} 的形式。这里的 {A} 和 {B} 都可以是 NEAREST 或者 LINEAR 中的一个。第一部分 {A} 负责控制每个 mipmap 层次的纹素构成方式，它与 GL_TEXTURE_MAG_FILTER 中的设置是按照同样的方式工作的。第二部分 {B} 负责控制采样点在两个 mipmap 层次之间的融合方式。如果设置为 NEAREST，那么计算只用到最近的 mipmap 层。如果设置为 LINEAR，那么两个最近的 mipmap 层数据之间会进行线性的插值计算。

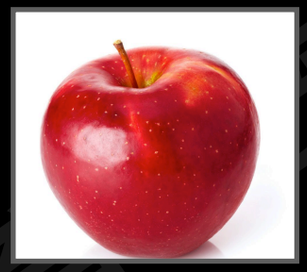
节自<OpenGL 编程指南第9版 >—第6章纹理与帧缓存

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

一系列Mip贴图图像





课程研发:CC老师
课程授课:CC老师

4 设置Mip 贴图

```
//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_MAG_LEVEL,0);
```


课程研发:CC老师
课程授课:CC老师

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

floorBatch几何坐标计算

几何坐标参考右图
纹理坐标参考图6-6

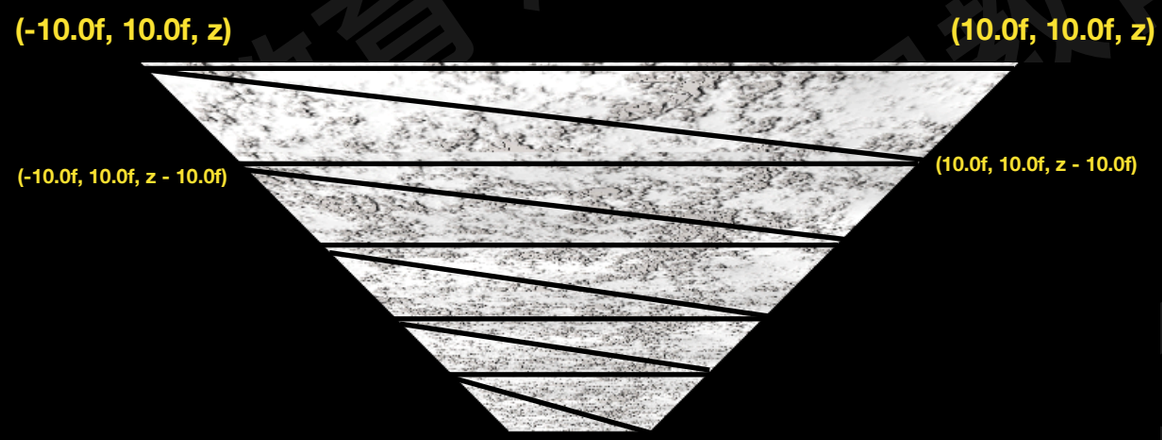
```
floorBatch.Begin(GL_TRIANGLE_STRIP, 28, 1);
    //Z表示深度，隧道的深度
    for(z = 60.0f; z >= 0.0f; z -=10.0f)
    {
        floorBatch.MultiTexCoord2f(0, 0.0f, 0.0f);
        floorBatch.Vertex3f(-10.0f, -10.0f, z);

        floorBatch.MultiTexCoord2f(0, 1.0f, 0.0f);
        floorBatch.Vertex3f(10.0f, -10.0f, z);

        floorBatch.MultiTexCoord2f(0, 0.0f, 1.0f);
        floorBatch.Vertex3f(-10.0f, -10.0f, z - 10.0f);

        floorBatch.MultiTexCoord2f(0, 1.0f, 1.0f);
        floorBatch.Vertex3f(10.0f, -10.0f, z - 10.0f);
}
floorBatch.End();
```

课程研发:CC老师
课程授课:CC老师

ceilingBatch几何坐标计算

几何坐标参考右图
纹理坐标参考图6-6

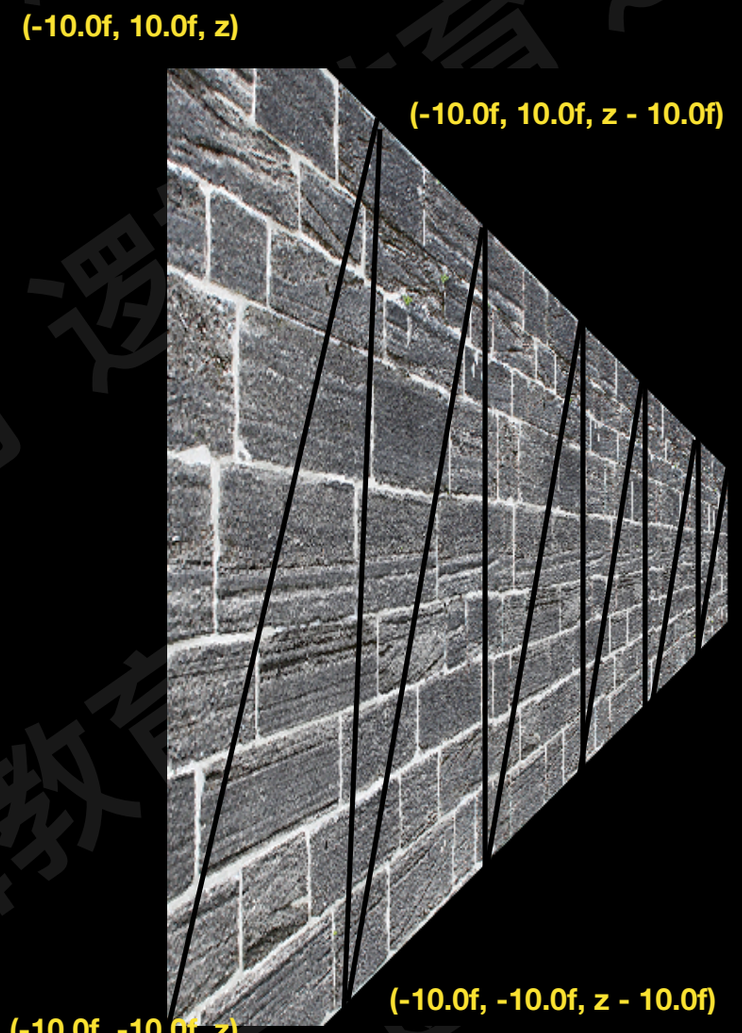
```
ceilingBatch.Begin(GL_TRIANGLE_STRIP, 28, 1);
    for(z = 60.0f; z >= 0.0f; z -=10.0f)
    {
        ceilingBatch.MultiTexCoord2f(0, 0.0f, 1.0f);
        ceilingBatch.Vertex3f(-10.0f, 10.0f, z - 10.0f);

        ceilingBatch.MultiTexCoord2f(0, 1.0f, 1.0f);
        ceilingBatch.Vertex3f(10.0f, 10.0f, z - 10.0f);

        ceilingBatch.MultiTexCoord2f(0, 0.0f, 0.0f);
        ceilingBatch.Vertex3f(-10.0f, 10.0f, z);

        ceilingBatch.MultiTexCoord2f(0, 1.0f, 0.0f);
        ceilingBatch.Vertex3f(10.0f, 10.0f, z);
    }
    ceilingBatch.End();
```



leftWallBatch几何坐标计算

几何坐标参考右图
纹理坐标参考图6-6

```
leftWallBatch.Begin(GL_TRIANGLE_STRIP, 28, 1);
    for(z = 60.0f; z >= 0.0f; z -=10.0f)
    {
        leftWallBatch.MultiTexCoord2f(0, 0.0f, 0.0f);
        leftWallBatch.Vertex3f(-10.0f, -10.0f, z);

        leftWallBatch.MultiTexCoord2f(0, 0.0f, 1.0f);
        leftWallBatch.Vertex3f(-10.0f, 10.0f, z);

        leftWallBatch.MultiTexCoord2f(0, 1.0f, 0.0f);
        leftWallBatch.Vertex3f(-10.0f, -10.0f, z - 10.0f);

        leftWallBatch.MultiTexCoord2f(0, 1.0f, 1.0f);
        leftWallBatch.Vertex3f(-10.0f, 10.0f, z - 10.0f);
    }
    leftWallBatch.End();
```

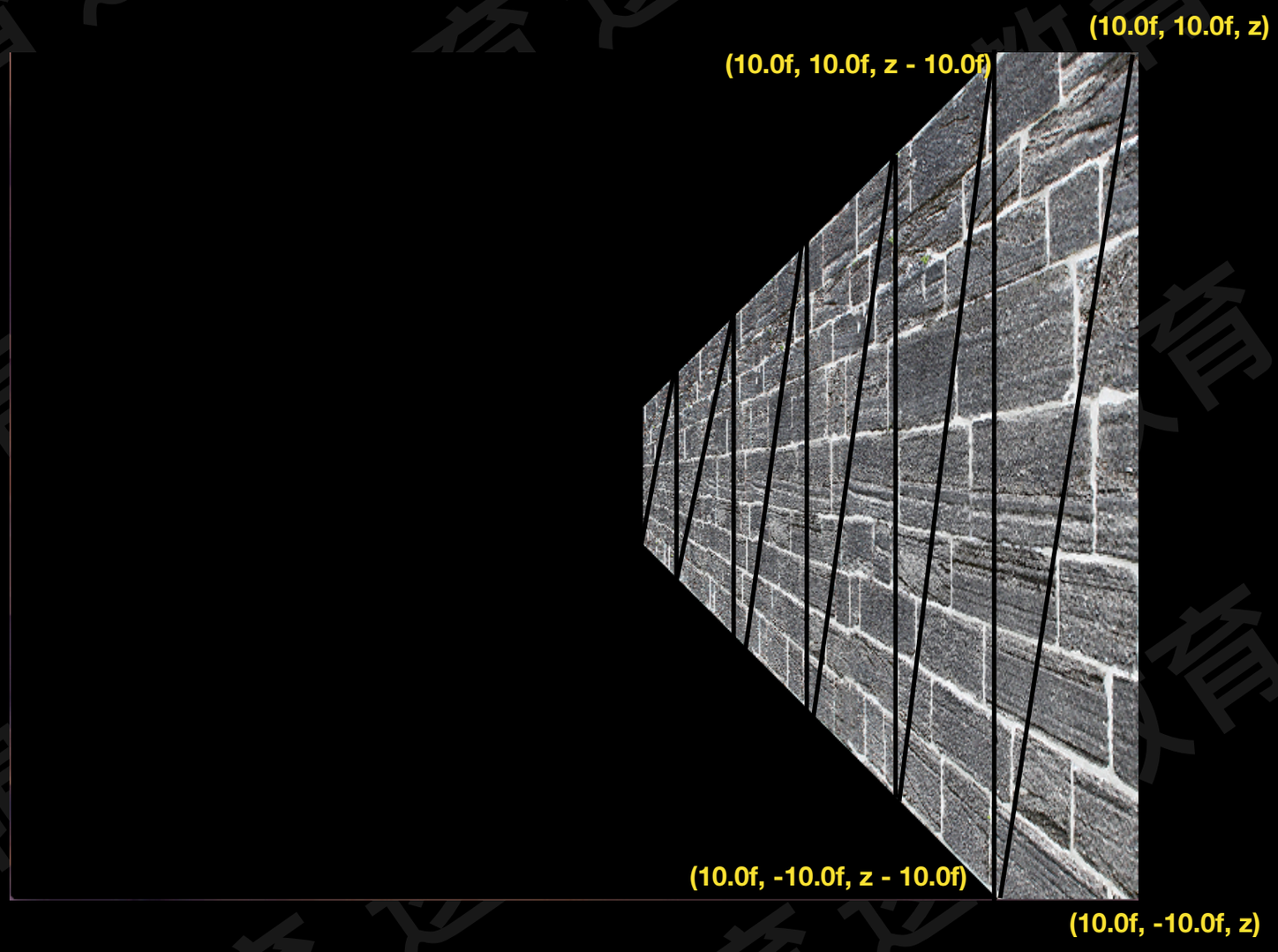


课程研发:CC老师
课程授课:CC老师

rightWallBatch几何坐标计算

几何坐标参考右图
纹理坐标参考图6-6

```
rightWallBatch.Begin(GL_TRIANGLE_STRIP, 28, 1);
    for(z = 60.0f; z >= 0.0f; z -=10.0f)
    {
        rightWallBatch.MultiTexCoord2f(0, 0.0f, 0.0f);
        rightWallBatch.Vertex3f(10.0f, -10.0f, z);

        rightWallBatch.MultiTexCoord2f(0, 0.0f, 1.0f);
        rightWallBatch.Vertex3f(10.0f, 10.0f, z);

        rightWallBatch.MultiTexCoord2f(0, 1.0f, 0.0f);
        rightWallBatch.Vertex3f(10.0f, -10.0f, z - 10.0f);

        rightWallBatch.MultiTexCoord2f(0, 1.0f, 1.0f);
        rightWallBatch.Vertex3f(10.0f, 10.0f, z - 10.0f);
    }
rightWallBatch.End();
```



课程研发:CC老师
课程授课:CC老师

5 压缩纹理

通用压缩纹理格式

压缩格式	基本内部格式
GL_COMPRESSED_RGB	GL_RGB
GL_COMPRESSED_RGBA	GL_RGBA
GL_COMPRESSED_SRGB	GL_RGB
GL_COMPRESSED_SRGB_ALPHA	GL_RGBA
GL_COMPRESSED_RED	GL_RED
GL_COMPRESSED_RG	GL_RG

5.1 判断压缩 与 选择压缩方式

```
GLint comFlag;
//判断纹理是否被成功压缩
glGetTexLevelParameteriv(GL_TEXTURE_2D,0,GL_TEXTURE_COMPRESSED,&comFlag);
```

```
//根据选择的压缩纹理格式，选择最快、最优、自行选择的算法方式选择压缩格式。
glHint(GL_TEXTURE_COMPRESSION_HINT,GL_FASTEST);
glHint(GL_TEXTURE_COMPRESSION_HINT,GL_NICEST);
glHint(GL_TEXTURE_COMPRESSION_HINT,GL_DONT_CARE);
```

课程研发:CC老师
课程授课:CC老师

5.2 加载压缩纹理

```
void glCompressedTexImage1D(GLenum target,GLint level,GLenum internalFormat,GLsizei
width,GLint border,GLsizei imageSize,void *data);

void glCompressedTexImage2D(GLenum target,GLint level,GLenum internalFormat,GLsizei
width,GLint heigth,GLint border,GLsizei imageSize,void *data);

void glCompressedTexImage3D(GLenum target,GLint level,GLenum internalFormat,GLsizei
width,GLsizei heigth,GLsizei depth,GLint border,GLsizei imageSize,void *data);
```

target:`GL_TEXTURE_1D`、`GL_TEXTURE_2D`、`GL_TEXTURE_3D`。
Level:指定所加载的mip贴图层次。一般我们都把这个参数设置为0。
internalformat:每个纹理单元中存储多少颜色成分。
width、height、depth参数:指加载纹理的宽度、高度、深度。==注意！==这些值必须是2的整数次方。（这是因为Op
旧版本上的遗留下的一个要求。当然现在已经可以支持不是2的整数次方。但是开发者们还是习惯使用以2的整数次方去
参数。）
border参数:允许为纹理贴图指定一个边界宽度。
format、type、data参数：与我们在讲glDrawPixels 函数对于的参数相同

课程研发:CC老师
课程授课:CC老师

glGetTexLevelParameter函数提取的压缩纹理格式

参数	返回
GL_TEXTURE_COMPRESSED	如果纹理被压缩，返回1，否则返回0
GL_TEXTURE_COMPRESSED_IMAGE_SIZE	压缩后的纹理的大小（以字节为单位）
GL_TEXTURE_INTERNAL_FORMAT	所使用的压缩格式
GL_NUM_COMPRESSED_TEXTURE_FORMATS	受支持的压缩纹理格式数量
GL_COMPRESSED_TEXTURE_FORMATS	一个包含了一些常量值的数组，每个常量值对应于一种受支持的压缩纹理格式
GL_TEXTURE_COMPRESSION_HINT	压缩纹理提示的值（GL/NICEST/GL_FASTEST）

GL_EXT_texture_compression_s3tc压缩格式

格式	描述
GL_COMPRESSED_RGB_S3TC_DXT1	RGB数据被压缩，alpha值始终是1.0
GL_COMPRESSED_RGBA_S3TC_DXT1	RGBA数据被压缩，alpha值返回1.0或者0.0
GL_COMPRESSED_RGAB_S3TC_DXT3	RGB值被压缩，alpha值用4位存储
GL_COMPRESSED_RGBA_SETC_DXT5	RGB数据被压缩，alpha值是一些8位值的加权平均值

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

Thanks For Your Attention !
See You Next Time!

课程研发:CC老师
课程授课:CC老师