9.2 在OpenGL中绘图和变换

- 图形的绘制
- 图形变换

图形的绘制——绘制命令

- 对数组从**first**到**first+ count-1**位置的数据按照**mode**的规则生成图形

void glDrawArrays(GLenum mode,GLint first,GLsizei count);

图形的绘制——绘制命令

- 用**count**个元素定义一系列图元，元素的索引值保存在缓存中，**indices**定义缓存的偏移量。

void glDrawElements(GLenum mode,Glsizei count,GLenum type,const Glvoid* indices)；

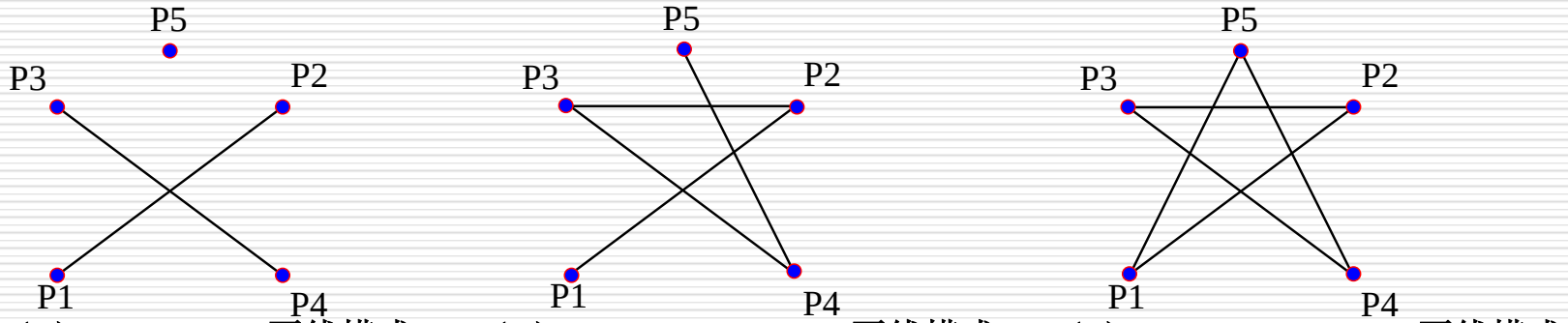
图形的绘制——点的绘制

- 点的绘制**GL_POINTS**
- 点的属性（大小）

void glPointSize(float size)；

图形的绘制——直线的绘制

- 直线的绘制模式
  - GL_LINES
  - GL_LINE_STRIP
  - GL_LINE_LOOP



（a）GL_LINES画线模式　（b）GL_LINE_STRIP画线模式　（c）GL_LINE_LOOP画线模式

图5-53 OpenGL画线模式

图形的绘制——直线的绘制

- 直线的属性:线宽

  void glLineWidth(float width)

图形的绘制——多边形面的绘制

- 三角形面的绘制
  - GL_TRIANGLES
  - GL_TRIANGLE_STRIP
  - GL_TRIANGLE_FAN
- 四边形面的绘制
  - GL_QUADS
  - GL_QUADS_STRIP
- 多边形面的绘制（GL_POLYGON）

图形的绘制——多边形面的属性

- 多边形面的绘制规则
  - 所有多边形都必须是平面的。
  - 多边形的边缘决不能相交，而且多边形必须是凸的。
- 解决：对于非凸多边形，可以把它分割成几个凸多边形（通常是三角形），再将它绘制出来。

图形的绘制——多边形面的属性

- 多边形面的正反属性（绕法）

  指定顶点时顺序和方向的组合称为“绕法”。绕法是任何多边形图元的一个重要特性。一般默认情况下，**OpenGL**认为逆时针绕法的多边形是正对着的。

  glFrontFace(GL_CW);

图形的绘制——多边形面的属性

- 多边形面的显示模式

  glPolygonMode(GLenum face，GLenum mode)；

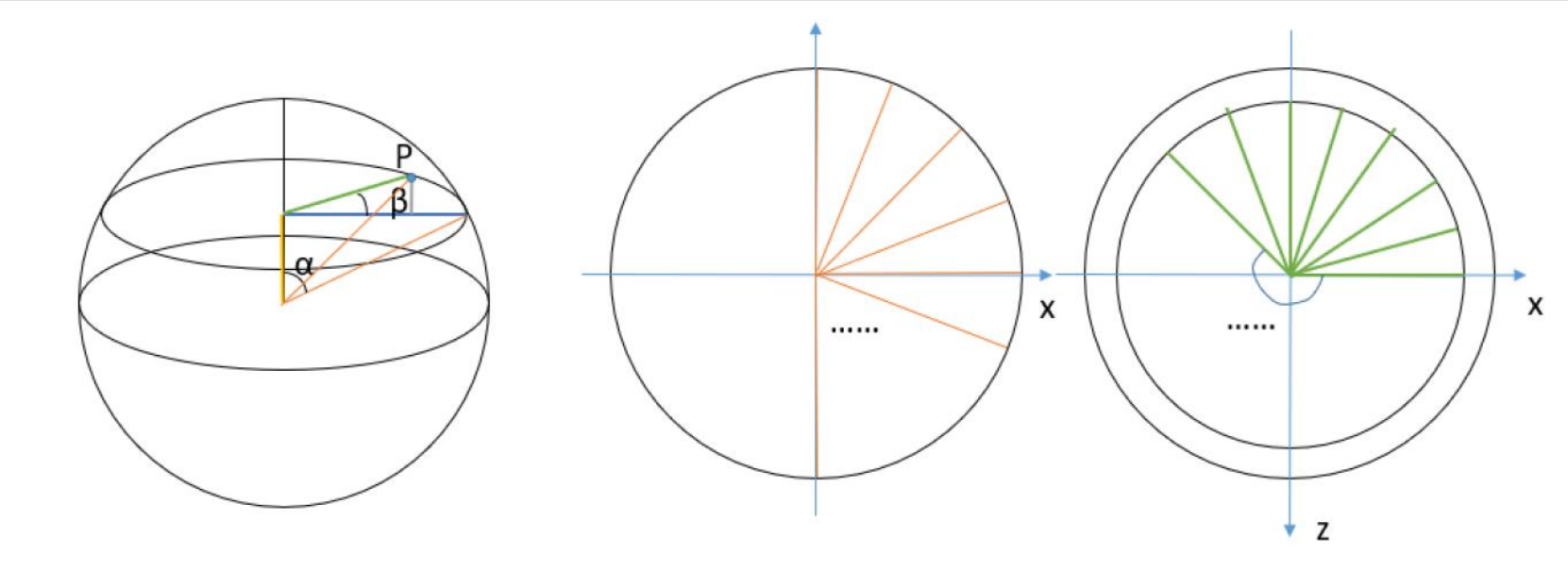
  - 参数face用于指定多边形的哪一个面受到模式改变的影响。
  - 参数mode用于指定新的绘图模式。

图形的绘制——使用EBO

- 可以将多个顶点放置在缓存中，绘图时可以选择指定的顶点进行绘制
- 定义索引数组
- 生成并绑定EBO
- 绘图时使用glDrawElements

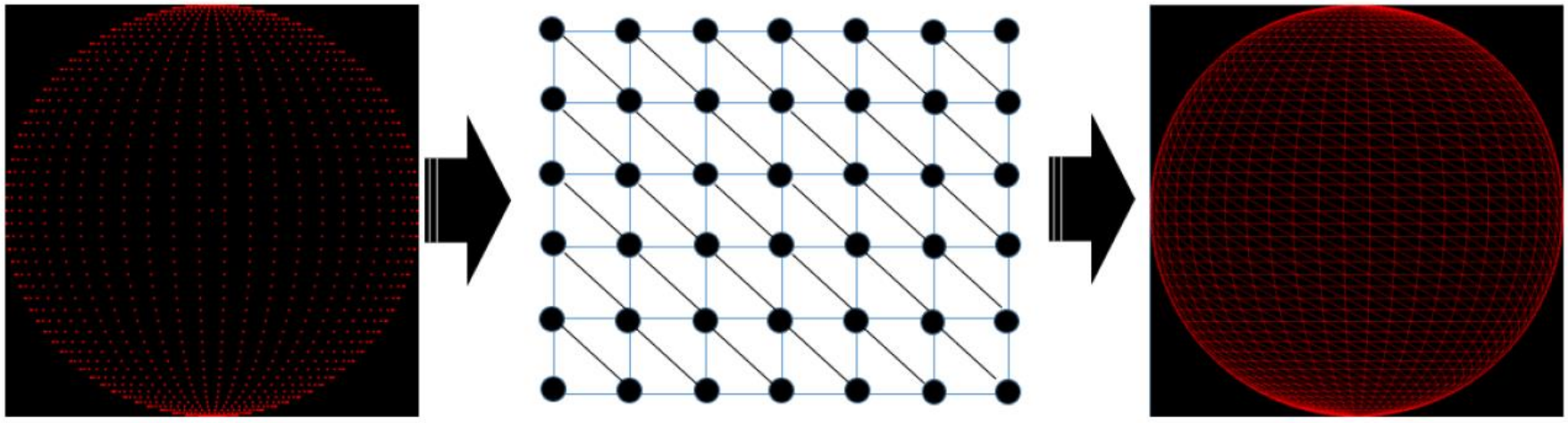
图形的绘制——绘制球体

- 球体上所有的点的坐标



图形的绘制——绘制球体

- 按照三角形构造顶点索引



OpenGL中的图形变换

- 变换种类
- 模型视图矩阵
- 投影变换

变换种类

- 视图变换：指定观察者或摄影机的位置；
- 模型变换：在场景中移动对象；
- 模型视图变换：描述视图变换与模型变换的对偶性；
- 投影变换：对视见空间进行修剪和改变大小；
- 视见区变换：对窗口的最终输出进行缩放；

模型视图矩阵

- 平移

  vmath::mat4 vmath:: translate (float x，float y，float z)；

- 旋转

  vmath::mat4 vmath:: rotate (float angle，const vecN<T,3>& axis)；

  vmath::mat4 vmath:: rotate (float x, float y, float z)；

模型视图矩阵

- 比例

```
vmath::mat4 vmath::scale (float x, float y, float z);
vmath::mat4 vmath::scale (float x);
vmath::mat4 vmath::scale (const Tvec3<T>& v);
```

模型视图矩阵

- 视图变换函数（定义观察坐标系）

```
vmath::mat4 vmath:: lookat（ const vecN<T,3>& eye， const vecN<T,3>& center, const vecN<T,3>& up）；
```

投影变换

- OpenGL中只提供了两种投影方式，一种是平行投影（正射投影），一种是透视投影。

投影变换

- 平行投影：视景体是一个矩形的平行管道，也就是一个长方体，其特点是无论物体距离相机多远，投影后的物体大小尺寸不变。

vmath::mat4 vmath::ortho (float left, float right, float bottom, float top, float near, float far);

透视投影

vmath::mat4 vmath:: frustum (float left, float right, float bottom, float top, float near, float far);

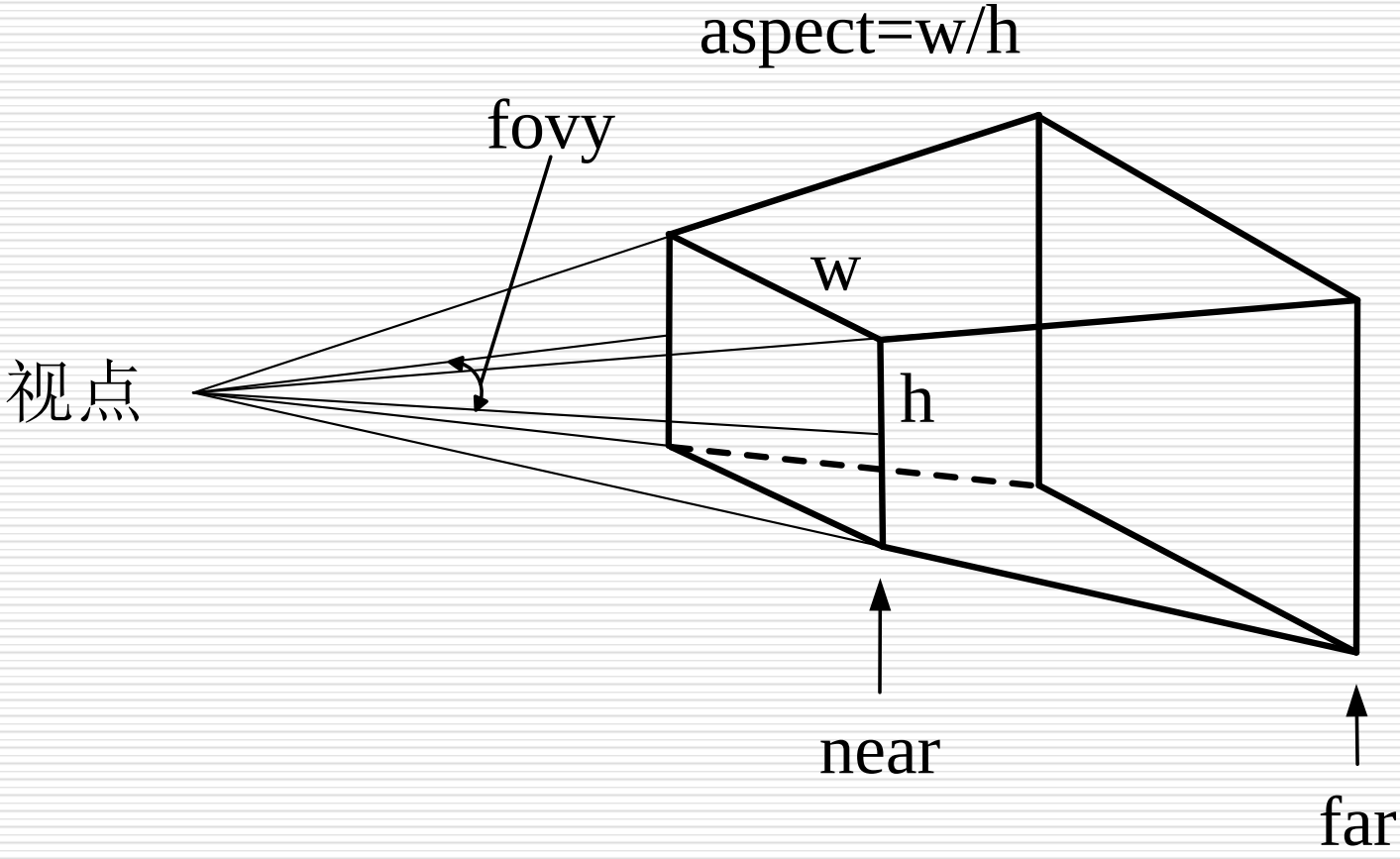
此函数创建一个透视投影矩阵，并且用这个矩阵乘以当前矩阵。它的参数只定义近裁剪平面的左下角点和右上角点的三维空间坐标，即（left，bottom，-near）和（right，top，-near）；最后一个参数far是远裁剪平面的Z负值，其左下角点和右上角点空间坐标由函数根据透视投影原理自动生成。

vmath::mat4 vmath::perspective (float fovy, float aspect, float zNear, float zFar);

它也创建一个对称透视视景体，但它的参数定义于前面的不同，其操作是创建一个对称的透视投影矩阵，并且用这个矩阵乘以当前矩阵。

参数**fovy**定义视野在**X-Z**平面（垂直方向上的可见区域）的角度，范围是**[0.0, 180.0]**；参数**aspect**是投影平面的纵横比（宽度与高度的比值）；参数**zNear**和**Far**分别是远近裁剪面沿**Z**负轴到视点的距离。

投影变换



OpenGL中的图形变换实现

- 相机移动到合适的位置：平移和旋转

  View=ViewR*ViewT

- 将模型移动到视野内：平移、旋转和缩放

  Model=ModelR*ModelT*ModelS

- 应用投影变换

  Project

深度测试

- 开启深度测试

```
glEnable(GL_DEPTH_TEST);
```

- 指定深度测试的比较函数

```
glDepthFunc(GL_LEQUAL);
```

- 使用前清除缓冲区

```
glClear(GL_DEPTH_BUFFER_BIT);
```

9.3 纹理

- 基本概念
- 纹理映射的实现

基本概念

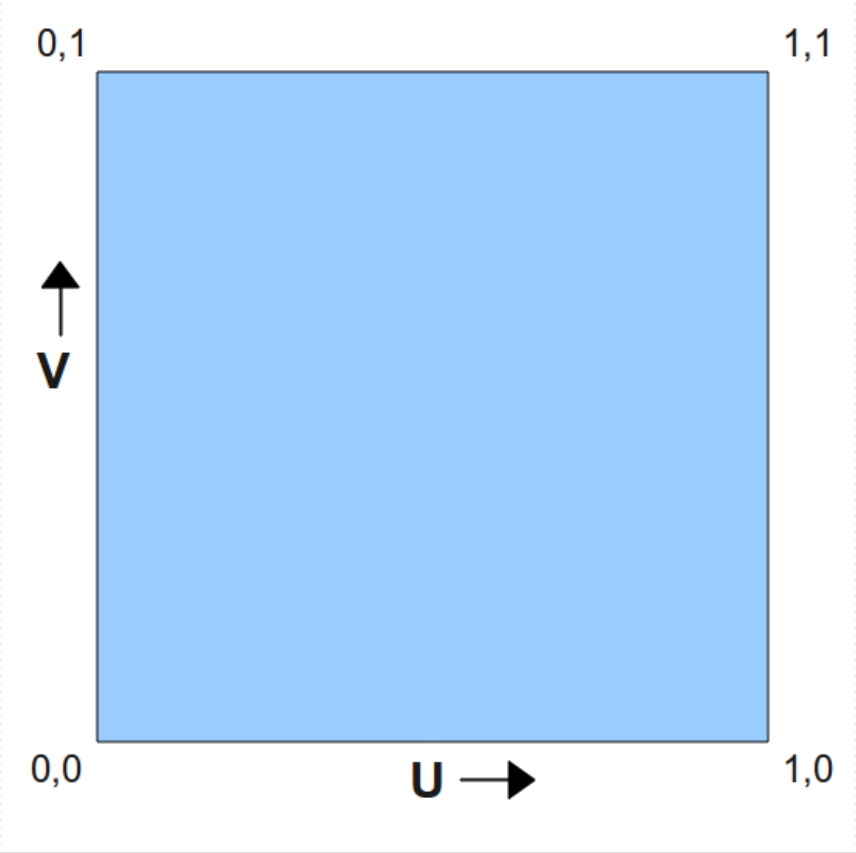
- 纹素(texel)
  - 表示纹理对象中的显示元素
  - 纹理对象通常是通过纹理图片读取，数据保存到一个二维数组中，这个数组中的元素称为纹素(texel)，纹素包含颜色值和alpha值
  - 纹理对象的大小的宽度和高度应该为2的整数幂

基本概念

- 纹理坐标
  - 通过纹理坐标(texture coordinate)获取纹理对象中的纹素
  - 纹理坐标与纹理对象大小无关
  - 纹理坐标使用规范化的值，大小范围为[0,1]，纹理坐标使用uv表示

基本概念

- 纹理坐标



纹理映射

- 纹理映射的步骤
  - 创建一个纹理对象并加载纹素数据
  - 为顶点坐标指定纹理坐标
  - 将纹理贴图关联到着色器，以便使用纹理采样器
  - 在着色器中通过纹理采样器获取纹素信息

创建纹理对象

- 创建纹理对象

glGenTextures(GLsizei n, GLunit *textures);

- 删除纹理对象

Void glDeleteTextures(GLsizei n, GLunit *textures)

绑定纹理对象

- 绑定纹理对象

Void glBindTextureUnit(

GLuint unit, GLuint texture)

设置纹理对象参数

- **WRAP参数(纹理环绕)**

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

设置纹理对象参数

- **WRAP参数(纹理环绕)**
  - **GL_REPEAT:**坐标的整数部分被忽略，重复纹理，这是**OpenGL**纹理默认的处理方式.
  - **GL_MIRRORED_REPEAT:** 纹理也会被重复，但是当纹理坐标的整数部分是奇数时会使用镜像重复。

设置纹理对象参数

- **WRAP**参数(纹理环绕)
  - **GL_CLAMP_TO_EDGE:** 坐标会被截断到**[0,1]**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:** 不在**[0,1]**范围内的纹理坐标会使用用户指定的边缘颜色。

设置纹理对象参数

- **WRAP参数**

GL_REPEAT

GL_MIRRORED_REPEAT

GL_CLAMP_TO_EDGE

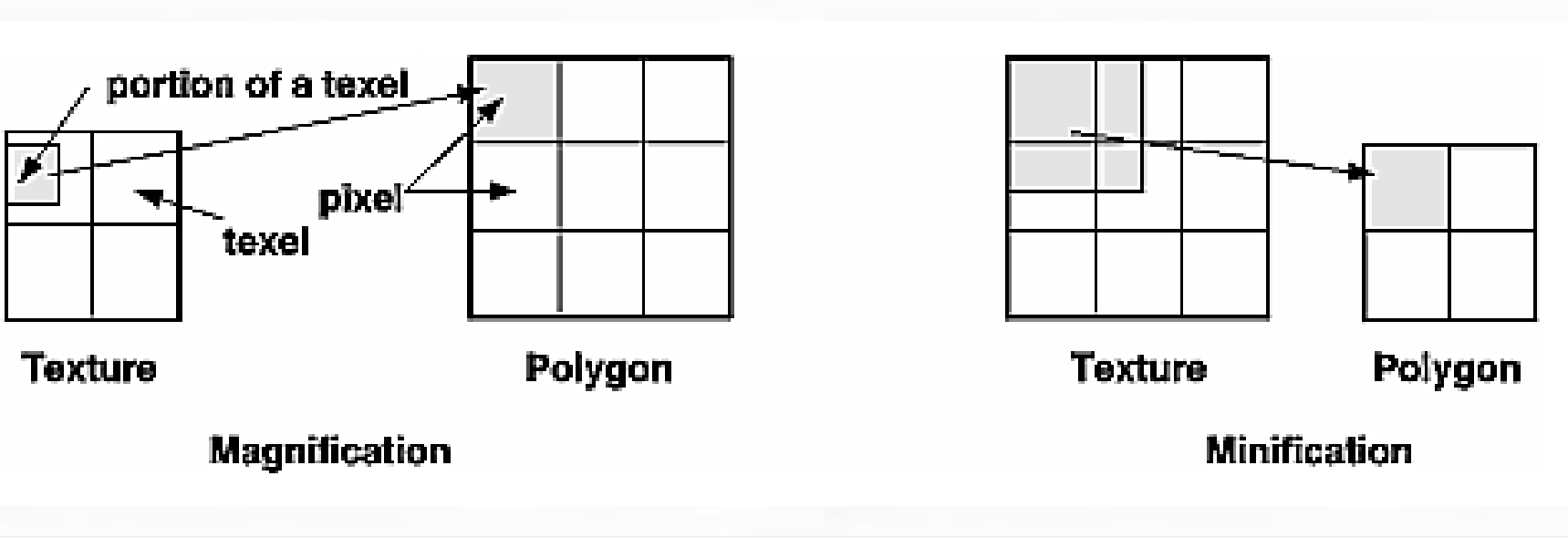
GL_CLAMP_TO_BORDER

设置纹理对象参数

- 纹素与像素的对应关系
  - 一个纹素最终对应屏幕上的多个像素，这称之为放大**(magnification)**
  - 一个纹素对应屏幕上的一个
  - 一个纹素对应少于一个像素，或者说多个纹素对应屏幕上的一个像素，这个称之为缩小**(minification)**

设置纹理对象参数

- 纹素与像素的对应关系



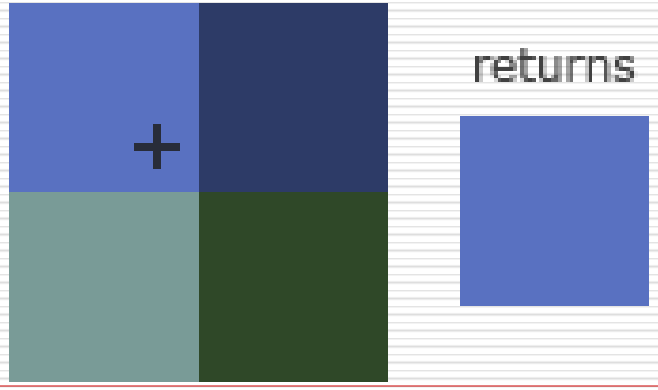
设置纹理对象参数

- **Filter参数**

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

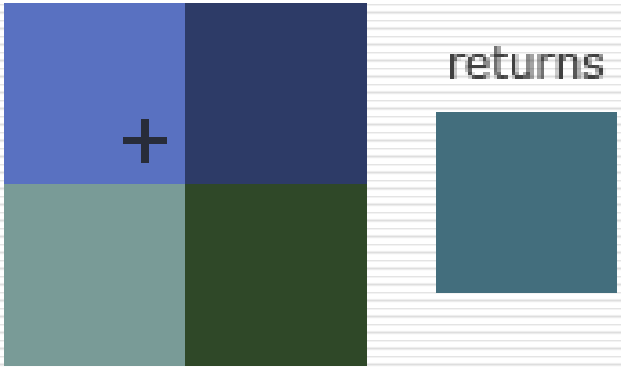
设置纹理对象参数

- Filter参数
  - GL_NEAREST:最近邻滤波(nearest neighbor filtering),对这个坐标进行取整，使用最佳逼近点来获取纹素。



设置纹理对象参数

- **Filter参数**
  - **GL_LINEAR :线性滤波方法(linear filtering)**，使用一组纹素的加权平均值来确定最终的纹素值。



分配内存空间

glTexImage2D(GLenum target, GLint level, GLint internalFormat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid * data);

分配内存空间

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);

- target为纹理对象的名称
- Level是纹理对象的Mipmap的层数，0表示基础层

分配内存空间

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);

- **Internalformat**设置纹理存储时使用的内部数据格式
- **width，height**表示纹理层的宽度，高度和深度

分配内存空间

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);

- **border**指定边框的宽度。必须为**0**
- **format**指定纹理数据的格式。必须匹配**internalformat**。

分配内存空间

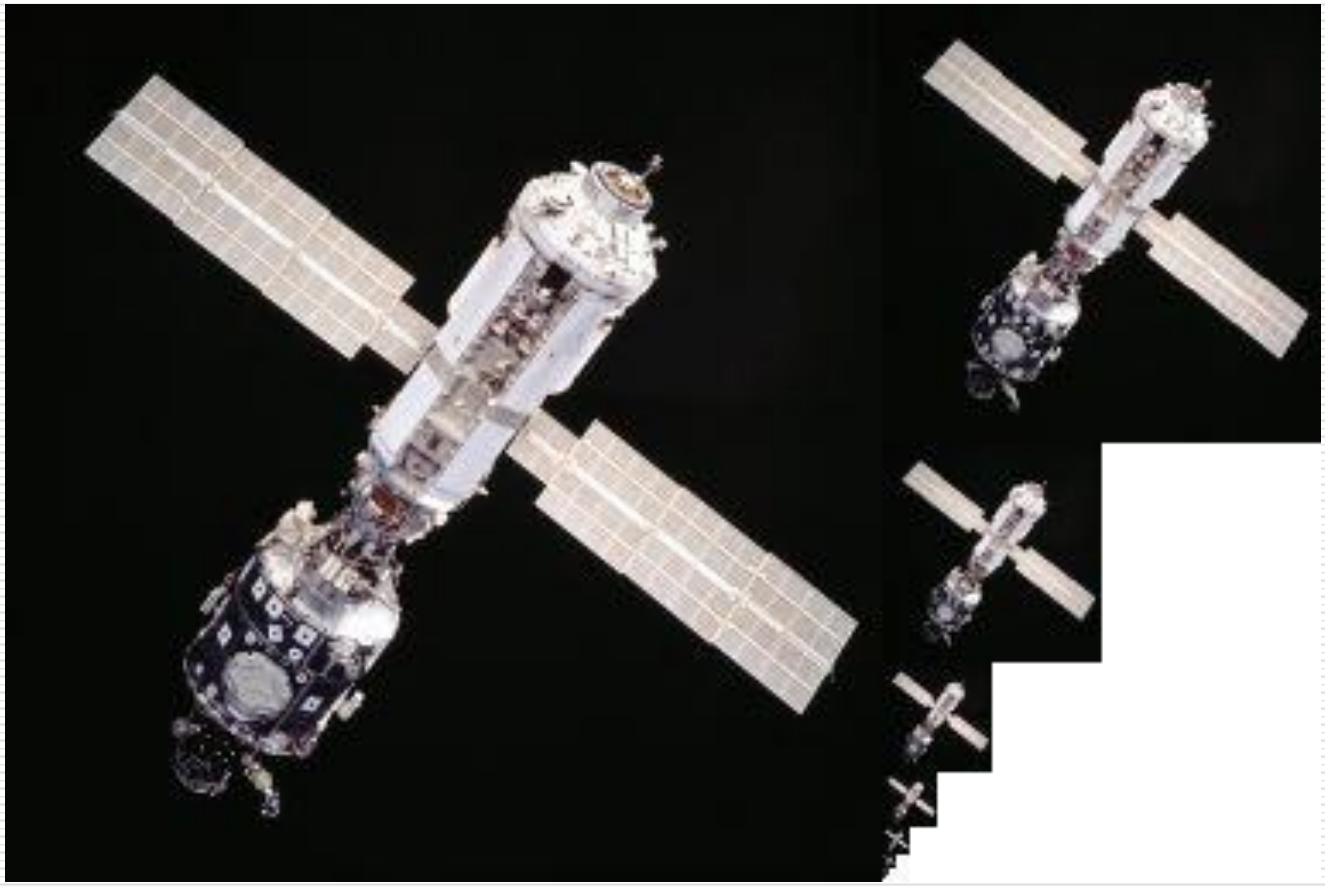
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);

- type指定纹理数据的数据类型。
- data指定一个指向内存中图像数据的指针。

分配内存空间——Mipmap

- 当景物与视点方向发生变化，多边形面对应像素的多少发生变化，在与纹素对应时，简单使用滤波计算，计算量较大。
- 提前计算一组纹理用来满足这种需求。

分配内存空间——Mipmap



分配内存空间——Mipmap

```
glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);

glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MAG_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
```

分配内存空间——Mipmap

- **GL_NEAREST_MIPMAP_NEAREST:** 使用最接近像素大小的**Mipmap**，纹理内部使用最近邻滤波。
- **GL_LINEAR_MIPMAP_NEAREST:** 使用最接近像素大小的**Mipmap**，纹理内部使用线性滤波。

分配内存空间——Mipmap

- **GL_NEAREST_MIPMAP_LINEAR:** 在两个最接近像素大小的**Mipmap**中做线性插值，纹理内部使用最近邻滤波。
- **GL_LINEAR_MIPMAP_LINEAR:** 在两个最接近像素大小的**Mipmap**中做线性插值，纹理内部使用线性滤波。

分配内存空间

- 纹理内存空间存储了给定纹理中的所有纹素
- 纹理内存空间大小由选定的内部格式和对应的分辨率决定
- 内存空间一旦被定义，无法重新定义
- 纹理的内容可以调用函数**glTextureSubImage2D**进行更改

分配内存空间

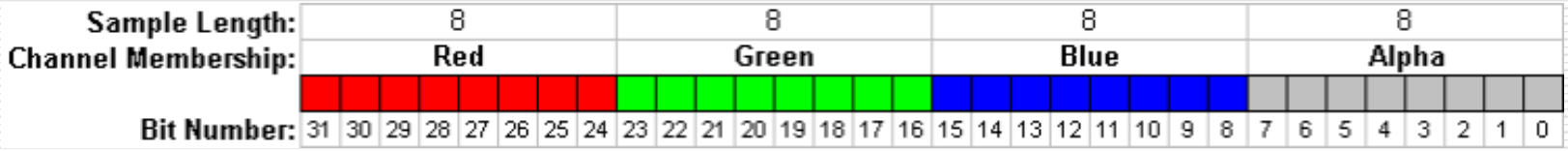
```
Void glTextureSubImage2D(
GLuint texture, GLint level,
GLint xoffset, GLint xoffset,
GLsizei width, GLsizei height,
Glenum format, Glenum type,
Const void *pixels)
```

分配内存空间

- 内部格式**internalformat**
  - **internalformat** 指定了颜色都有那些成分构成，如**RGB**表示颜色组成由**Red**分量**Green**分量**Blue**分量构成，**RGBA**表示颜色组成由**Red**分量**Green**分量**Blue**分量**Alpha**分量构成。

分配内存空间

- 外部格式format
  - 表示了颜色数据分量在内存中的数据存储格式



生成纹理

- 调用**glGenerateMipmap**。这会为当前绑定的纹理自动生成所有需要的多级渐远纹理

```
glGenerateMipmap(GL_TEXTURE_2D);
```

着色器

- 在顶点着色器中处理纹理坐标，并输出纹理坐标；
- 使用**GLSL**内建的**texture**函数来采样纹理的颜色。**texture**函数会使用之前设置的纹理参数对相应的颜色值进行采样。片段着色器的输出就是纹理的（插值）纹理坐标上的**(**过滤后的**)**颜色。

加载纹理图像

- 使用图像解码库**stb_image**进行纹理图像的加载
- **stb_image.h**用于图像加载
- **stb_image_write.h**用于写入图像文件
- **stb_image_resize.h**用于改变图像尺寸

```
unsigned char *data =
stbi_load("container.jpg", &width,
&height, &nrChannels, 0);
```

纹理与颜色

- 在片段着色器中可以实现纹理和颜色的混合

color = texture(tex, tex_coord)*ncolor

多重纹理

- 纹理单元(Texture Unit)：纹理的位置值。在着色器中允许使用多于一个的纹理。把纹理单元赋值给采样器，可以一次绑定多个纹理。

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture);
```

多重纹理

- OpenGL至少保证有16个纹理单元供使用，也就是说可以激活从GL_TEXTURE0到GL_TEXTRUE15。它们都是按顺序定义的。
- GLSL内建的mix函数实现纹理混合。参数1和2是纹理单元，参数3是混合的方式。