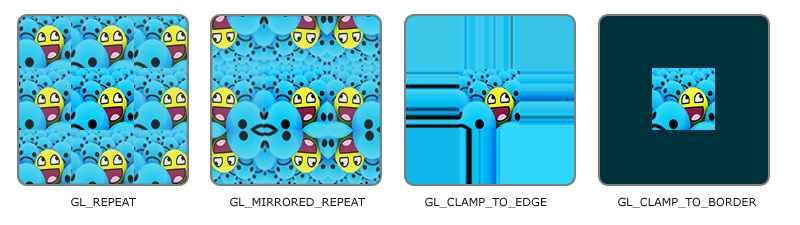
# 入门

## 文理

### 纹理环绕方式

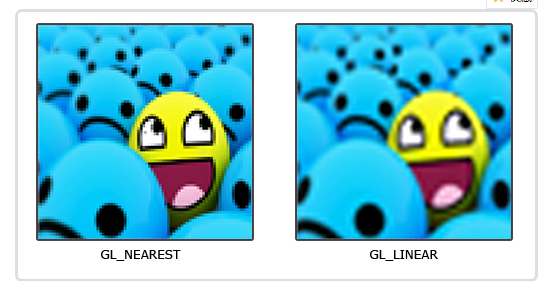
| **环绕方式** | **描述** |
| --- | --- |
| GL\_REPEAT | 对纹理的默认行为。重复纹理图像。 |
| GL\_MIRRORED\_REPEAT | 和GL\_REPEAT一样，但每次重复图片是镜像放置的。 |
| GL\_CLAMP\_TO\_EDGE | 纹理坐标会被约束在0到1之间，超出的部分会重复纹理坐标的边缘，产生一种边缘被拉伸的效果。 |
| GL\_CLAMP\_TO\_BORDER | 超出的坐标为用户指定的边缘颜色。 |



### 纹理过滤

GL\_NEAREST 邻近过滤

GL\_LINEAR 线性过滤



### 多级渐远纹理

解决近大远小分辨率问题

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。

| **过滤方式** | **描述** |
| --- | --- |
| GL\_NEAREST\_MIPMAP\_NEAREST | 使用最邻近的多级渐远纹理来匹配像素大小，并使用邻近插值进行纹理采样 |
| GL\_LINEAR\_MIPMAP\_NEAREST | 使用最邻近的多级渐远纹理级别，并使用线性插值进行采样 |
| GL\_NEAREST\_MIPMAP\_LINEAR | 在两个最匹配像素大小的多级渐远纹理之间进行线性插值，使用邻近插值进行采样 |
| GL\_LINEAR\_MIPMAP\_LINEAR | 在两个邻近的多级渐远纹理之间使用线性插值，并使用线性插值进行采样 |

主要函数:glTexParameteri()

### 加载与创建纹理

stb\_image.h库

函数:

stbi\_load(),

glGenTextures()生成文理 输入生成纹理的数量，然后把它们储存在第二个参数的 unsigned int数组中

glBindTexture(GL\_TEXTURE\_2D, texture);绑定文理

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, width, height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, data); glGenerateMipmap(GL\_TEXTURE\_2D);

* 第一个参数指定了纹理目标(Target)。设置为GL\_TEXTURE\_2D意味着会生成与当前绑定的纹理对象在同一个目标上的纹理（任何绑定到GL\_TEXTURE\_1D和GL\_TEXTURE\_3D的纹理不会受到影响）。
* 第二个参数为纹理指定多级渐远纹理的级别，如果你希望单独手动设置每个多级渐远纹理的级别的话。这里我们填0，也就是基本级别。
* 第三个参数告诉OpenGL我们希望把纹理储存为何种格式。我们的图像只有RGB值，因此我们也把纹理储存为RGB值。
* 第四个和第五个参数设置最终的纹理的宽度和高度。我们之前加载图像的时候储存了它们，所以我们使用对应的变量。
* 下个参数应该总是被设为0（历史遗留的问题）。
* 第七第八个参数定义了源图的格式和数据类型。我们使用RGB值加载这个图像，并把它们储存为char(byte)数组，我们将会传入对应值。
* 最后一个参数是真正的图像数据。

unsigned int texture;

glGenTextures(1, &texture);

glBindTexture(GL\_TEXTURE\_2D, texture);

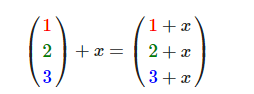
stbi\_image\_free(data);

## 变换

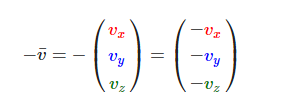
### 向量

向量有一个方向(Direction)和大小(Magnitude，也叫做强度或长度)

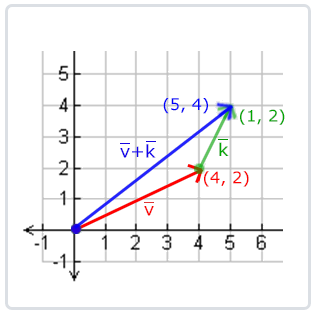
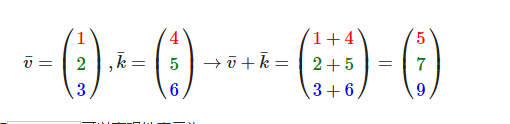
向量与标量运算

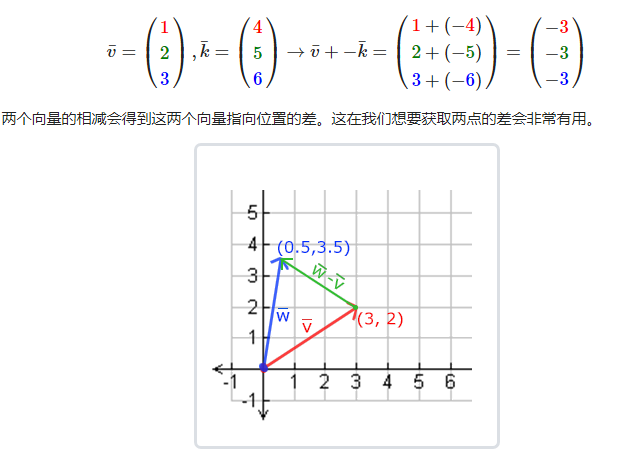


向量取反



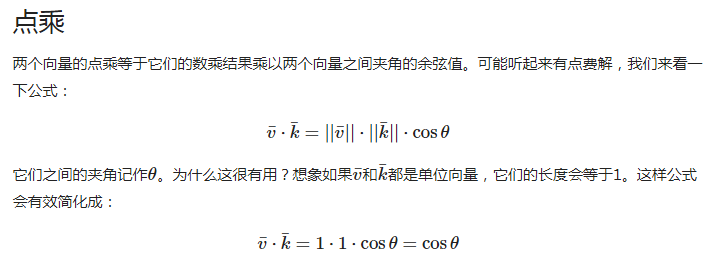
向量加减

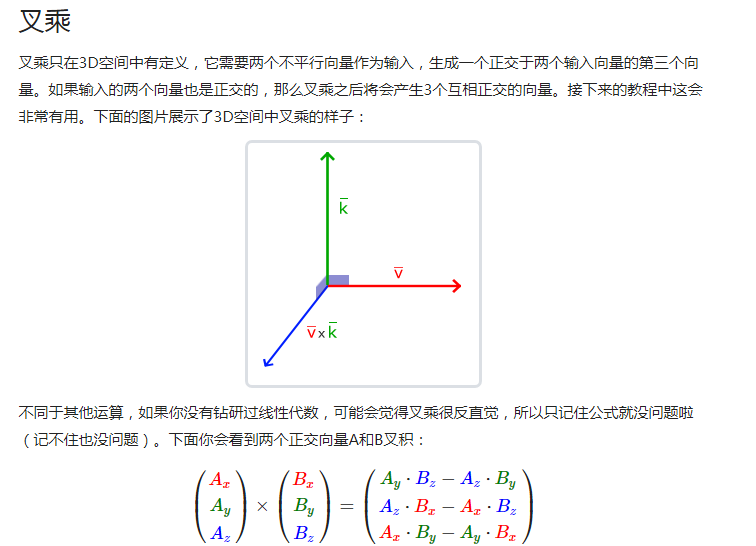




长度

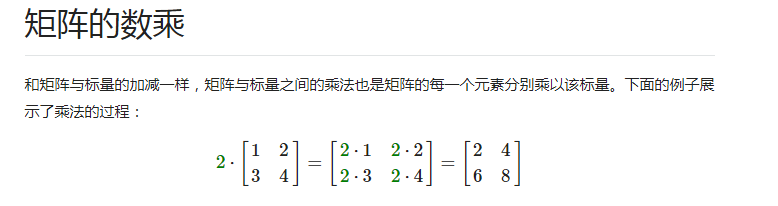
向量相乘



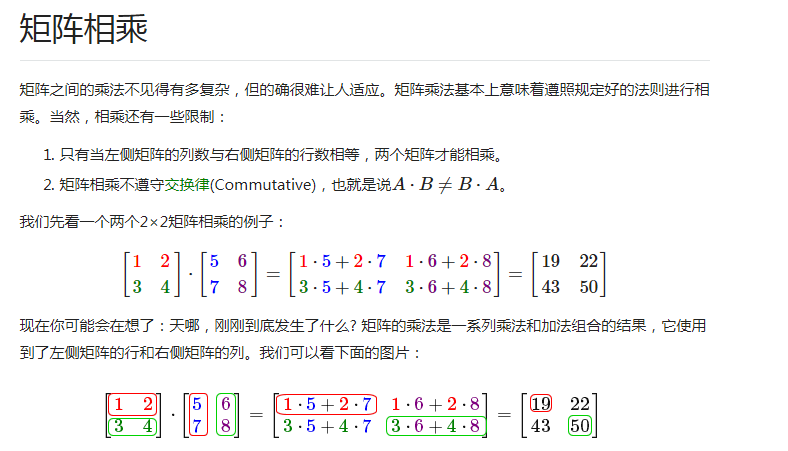


### 矩阵



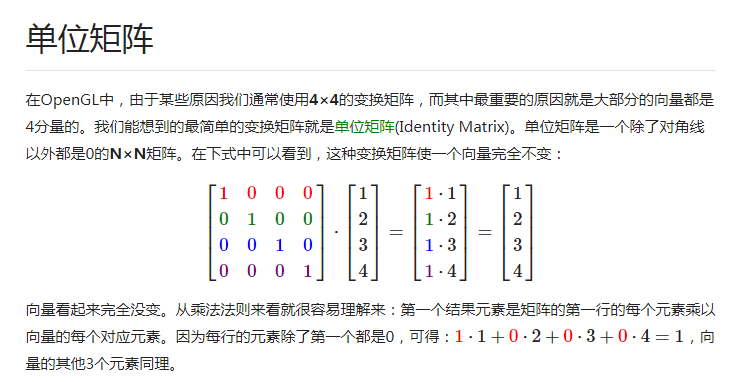


标量就是用它的值****缩放****(Scale)矩阵的所有元素

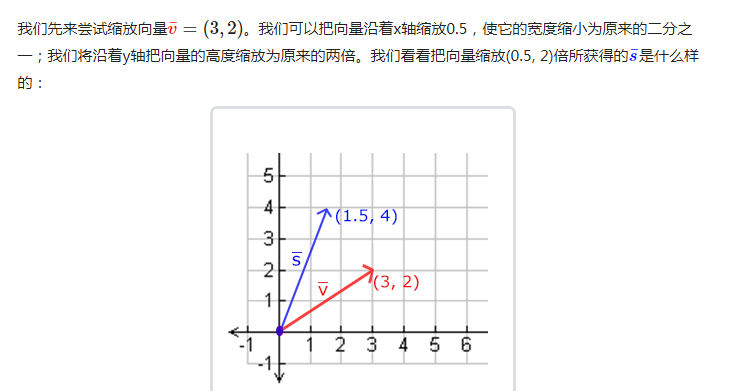


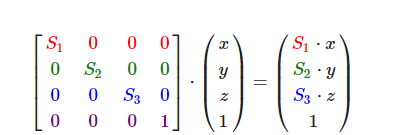
矩阵与向量相乘

用这个矩阵乘以我们的向量将****变换****(Transform)这个向量。



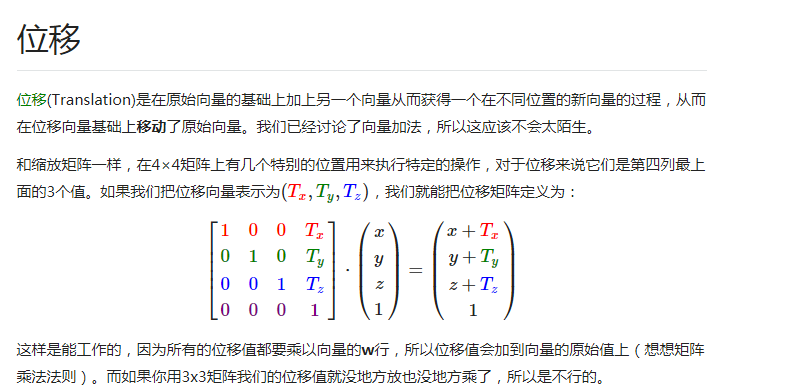
**缩放**



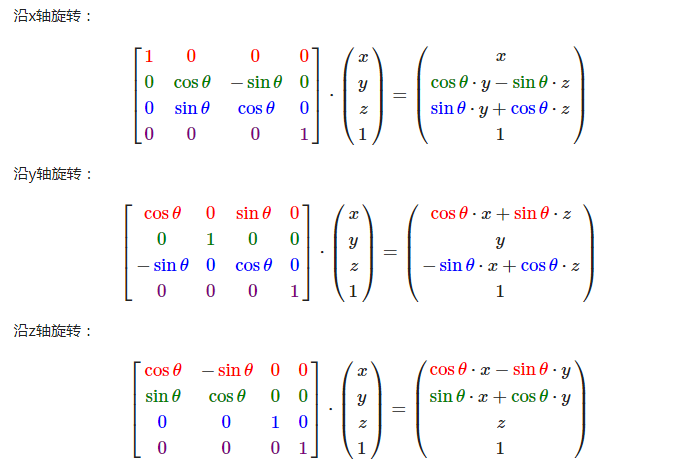


不均匀)缩放 : 每个轴的缩放因子(Scaling Factor)都不一样

均匀缩放: 每个轴的缩放因子都一样



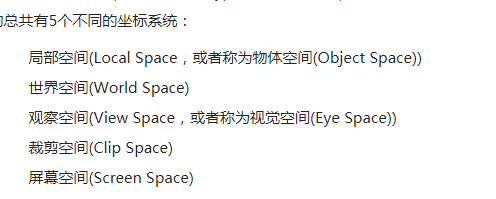
**旋转**

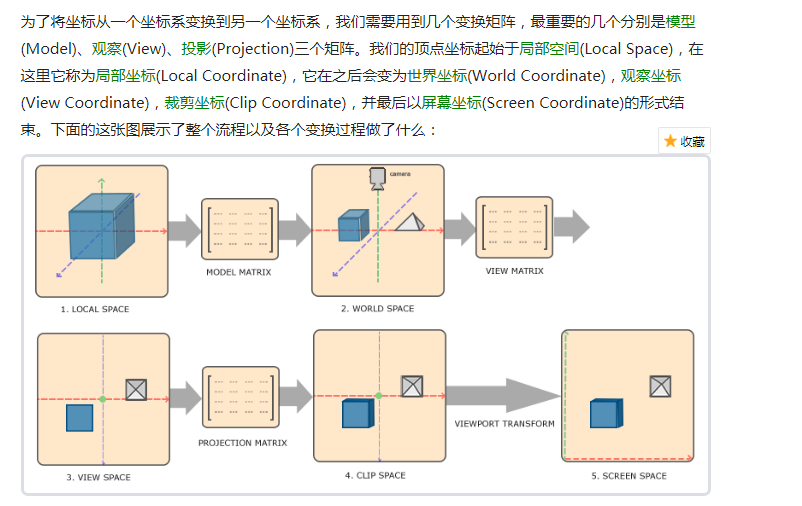


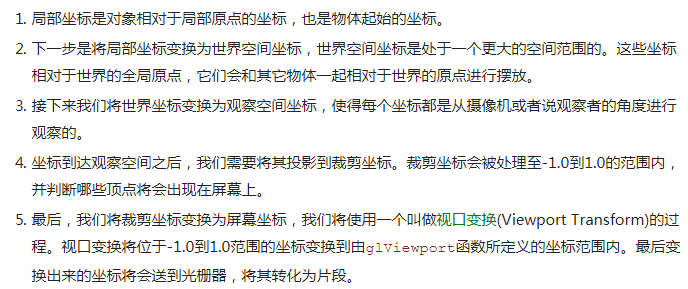
glm::radians将角度转化为弧度

glm::translate函数来完成这个工作的（然后用给定的矩阵乘以位移矩阵就能获得最后需要的矩阵）。

## 坐标系统

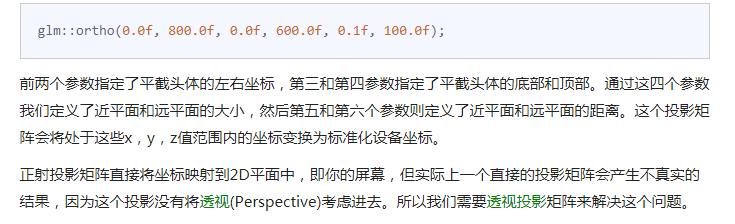






### 正射投影

创建一个正射投影矩阵，我们可以使用GLM的内置函数glm::ortho：

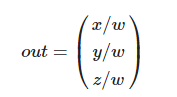


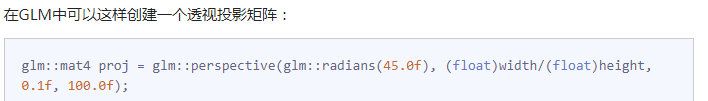
用于二维渲染

透视投影

透视 : 近大远小;

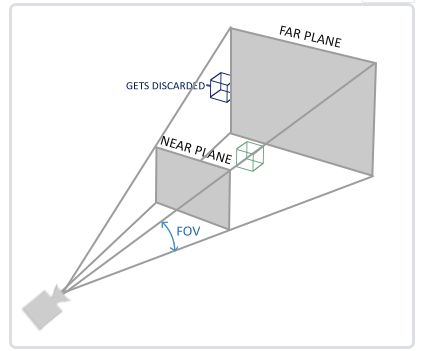
OpenGL要求所有可见的坐标都落在-1.0到1.0范围内，作为顶点着色器最后的输出，因此，一旦坐标在裁剪空间内之后，透视除法就会被应用到裁剪空间坐标上：





同样，glm::perspective所做的其实就是创建了一个定义了可视空间的大平截头体，任何在这个平截头体以外的东西最后都不会出现在裁剪空间体积内，并且将会受到裁剪。

如下图



它的第一个参数定义了fov的值，它表示的是视野(Field of View)，并且设置了观察空间的大小。如果想要一个真实的观察效果，它的值通常设置为45.0f，但想要一个末日风格的结果你可以将其设置一个更大的值。第二个参数设置了宽高比，由视口的宽除以高所得。第三和第四个参数设置了平截头体的近和远平面。我们通常设置近距离为0.1f，而远距离设为100.0f。所有在近平面和远平面内且处于平截头体内的顶点都会被渲染。