逻辑教育
Logic education

Hello 视觉全训班

OpenGL
OpenGL ES
GPUImage
Metal

视觉全训班. 关于OpenGL 下矩阵应用场景

@ CC老师

全力以赴．非同凡”想”

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

3D数学.在图像图形开发中的充当了什么角色?

对于学习OpenGL有一个误区，就是大家认为如果不能精通那些3D图形数学知识，会让我们寸步难行，其实不然。就像我们不需要懂得任何关于汽车结构和内燃机方面的知识也能每天开车。但是，我们最好能对汽车有足够的了解，以便我们意识到什么时候需要更换机油、定期加油、汽车常规保养工作。

同样要成为一名可靠和有能力的OpenGL程序员，至少需要理解这些基础知识，才知道能作什么？以及那些工具适合我们要做的工作。

初学者，经过一段时间的实践，就会渐渐理解矩阵和向量。并且培养出一种更为直观的能力，能够在实践中充分利用所学的内容。

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

3D数学.在图像图形开发中的充当了什么角色?

即使大家现在还没有能力在脑海中默算出2个矩阵的乘法、也要明白矩阵是什么？以及这些矩阵对OpenGL意味着什么?

GLTools 库中有一个组件叫Math3d,其中包含了大量好用的OpenGL 一致的3D数学和数据类型。虽然我们不必亲自进行所有的矩阵和向量的操作，但我然知道它们是什么？以及如何运用它们.

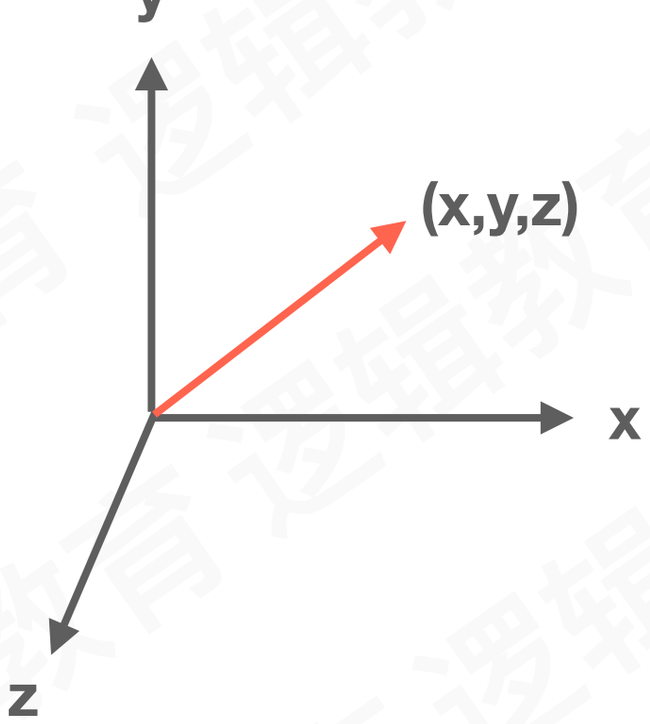
在开发过程我们涉及到的图形变换,就会涉及到矩阵/向量的计算.例如大家在使用CAnimation 实现仿射变换,就使用了OpenGL渲染技术.

课程研发:CC老师
课程授课:CC老师

向量

在 3D 笛卡尔坐标系, 基本上. 一个顶点 就是XYZ 坐标空间上的一个位置. 而在空间中给定的一个位置 恰恰是由一个单独的 XYZ 定义的. 而这这样的 XYZ 就是向量;

@在数学思维中,一个顶点也就是一个向量.



在 X轴上的向量 (1,0,0). 向量长度为1. 我们称为长度为1的向量为单位向量.

向量长度(向量的模)计算公式:

$$\sqrt[2]{x^2+y^2+z^2}$$

如果一个向量不是单位向量, 而我们把它缩放到1. 这个过程叫做标准化. 将一个向量进行标准化就是将它的缩为1; 也叫做单位化向量;

课程研发:CC老师
课程授课:CC老师

OpenGL 如何定义向量 [math3d 库]

math3d库，有2个数据类型，能够表示一个三维或者四维向量。
M3DVector3f可以表示一个三维向量（x,y,z），
M3DVector4f则可以表示一个四维向量（x,y,z,w).
在典型情况下，w坐标设为1.0。x,y,z值通过除以w，来进行缩放。而除以1.0则本质上不改变x,y,z值。

@代码参考

```
//三维向量/四维向量的声明
typedef float M3DVector3f[3];
typedef float M3DVector4f[4];

//声明一个三维向量 M3DVector3f:类型  vVector:变量名
M3DVector3f vVector;

//声明一个四维向量并初始化一个四维向量
M3DVector4f vVertex = {0,0,1,1};

//声明一个三分量顶点数组，例如生成一个三角形
//M3DVector3f vVerts[] = {
                -0.5f,0.0f,0.0f,
                0.5f,0.0f,0.0f,
                0.0f,0.5f,0.0f
};
```

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

向量 点乘

向量可以进行 加法,减法计算. 但是向量里有一个在开发中使用价值非常高的操作,叫做“点乘(dot product)” .点乘只能发生在2个向量之间进行;

2个(三维向量)单元向量 之间进行点乘运算将得到一个标量(不是三维向量,是一个标量).
它表示两个向量之间的夹角;

前提条件: 2个向量必须为单位向量;
动作: 2个三维向量之间进行点乘
结构: 返回一个[-1,1]范围的值. 这个值其实就是 夹角的cos值(余弦值)

如何单位化向量?
(x/|xyz|, y/|xyz|, z/|xyz|);
使用一个非零向量除以它的模(向量的长度), 就可以得到方向相同的单位向量;

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

向量 点乘

@点乘运算返回2个向量之间的夹角

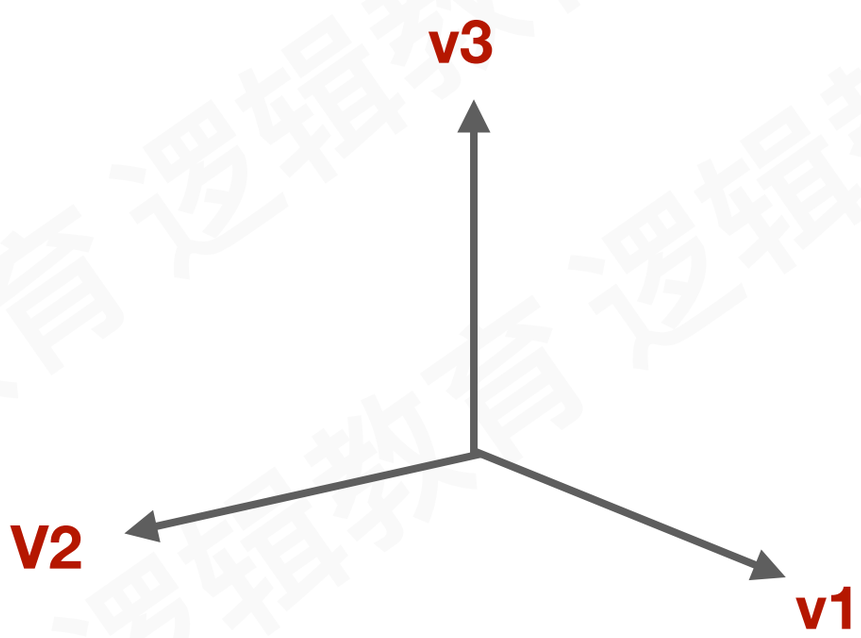
```
math3d 库中提供了关于点乘的API

//1.m3dDotProduct3 函数获得2个向量之间的点乘结果;
float m3dDotProduct3(const M3DVector3f u,const M3DVector3f v);

//2.m3dGetAngleBetweenVector3 即可获取2个向量之间夹角的弧度值;
 float m3dGetAngleBetweenVector3(const M3DVector3f u,const
M3DVector3f v);
```

课程研发:CC老师
课程授课:CC老师

向量 叉乘

向量之间的叉乘(cross product) 也是在业务开发里非常有用的一个计算方式； 2个向量之间叉乘就可以得到另外一个向量,新的向量会与原来2个向量定义的平面垂直. 同时进行叉乘,不必为单位向量;

前提： 2个普通向量
动作： 向量与向量叉乘
结果： 向量(垂直于原来2个向量定义的平面的向量)



叉乘运算结果返回一个新的向量, 这个向量与原来2个向量垂直

课程研发:CC老师
课程授课:CC老师

向量 叉乘

math3d 库中提供了关于叉乘的API

```
//1.m3dCrossProduct3 函数获得2个向量之间的叉乘结果得到一个新的向量
void m3dCrossProduct3(M3DVector3f result,const M3DVector3f  u ,const
M3DVector3f v);
```

课程研发:CC老师
课程授课:CC老师

矩阵 Matrix

假设，在空间有一个点.使用 xyz 描述它的位置. 此时让其围绕任意位置旋转一定角度后. 我们需要知道这个点的新的位置. 此时需要通过矩阵进行计算；

为什么？
因为新的位置的x 不单纯与原来的x还和旋转的参数有关. 甚至于y和z坐标有关；

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \quad \begin{bmatrix} 0 & 42 \\ 1.5 & 0.877 \\ 2 & 14 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

三个矩阵

矩阵只有一行或者一列都是合理的. 只有一行或者一列数字可以称为向量，也可以称为矩阵；

矩阵 Matrix

@代码参考

```
//三维矩阵/四维矩阵的声明
typedef float M3DMatrix33f[9];
typedef float M3DMatrix44f[16];
```

在其他编程标准中，许多矩阵库定义一个矩阵时,使用二维数组；OpenGL的约定里,更多倾向使用 一维数组；这样做的原因是：OpenGL 使用的是 Column-Major(以列为主)矩阵排序的约定;

这个在数学叫 转置矩阵

A0	A1	A2	A3
A4	A5	A6	A7
A8	A9	A10	A11
A12	A13	A14	A15

行优先矩阵排序

A0	A4	A8	A12
A1	A5	A9	A13
A2	A6	A10	A14
A3	A7	A11	A15

列优先矩阵排序

课程研发:CC老师
课程授课:CC老师

矩阵 Matrix

X轴方向　Y轴方向　Z轴方向　交换、位置

$$\begin{bmatrix} X_x & Y_x & Z_X & T_x \\ X_y & Y_y & Z_y & T_y \\ X_z & Y_z & Z_z & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

奥秘之处,在于这 16 个值表示空间中一个特定的位置;

这4列中,每一列都是有4个元素组成的向量;

如果将一个对象所有的顶点向量 乘以这个矩阵,就能让整个对象变换到空间中给定的位置和方向;

一个4*4矩阵是如何在3D空间中表示一个位置和方向的

列向量进行了特别的标注：矩阵的最后一行都为0，只有最后一个元素为1

课程研发:CC老师
课程授课:CC老师

单元矩阵 Matrix

```
//单元矩阵初始化方式①
GLFloat m[] = {
            1,0,0,0, //X Column
            0,1,0,0, //Y Column
            0,0,1,0, //Z Column
            0,0,0,1  // Translation
        }
```

```
// 单元矩阵初始化方式 ②
M3DMatrix44f m = {
            1,0,0,0, //X Column
            0,1,0,0, //Y Column
            0,0,1,0, //Z Column
            0,0,0,1  // Translation
        }
```

```
//单元矩阵初始化方式③
void m3dLoadIdentity44f(M3DMatrix44f m);
```

课程研发:CC老师
课程授课:CC老师

单元矩阵 Matrix

将一个向量 × 单元矩阵 ,就相当于一个向量×1. 不会发生任何改变;

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 8 \\ 4 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 8 \\ 4 \\ 2 \\ 1 \end{bmatrix}$$

列矩阵-单元矩阵 4 × 4　　4 × 1 矩阵　　4 × 1 矩阵

$$\begin{bmatrix} 8 \\ 4 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \text{无定义(不符合矩阵规则)}$$

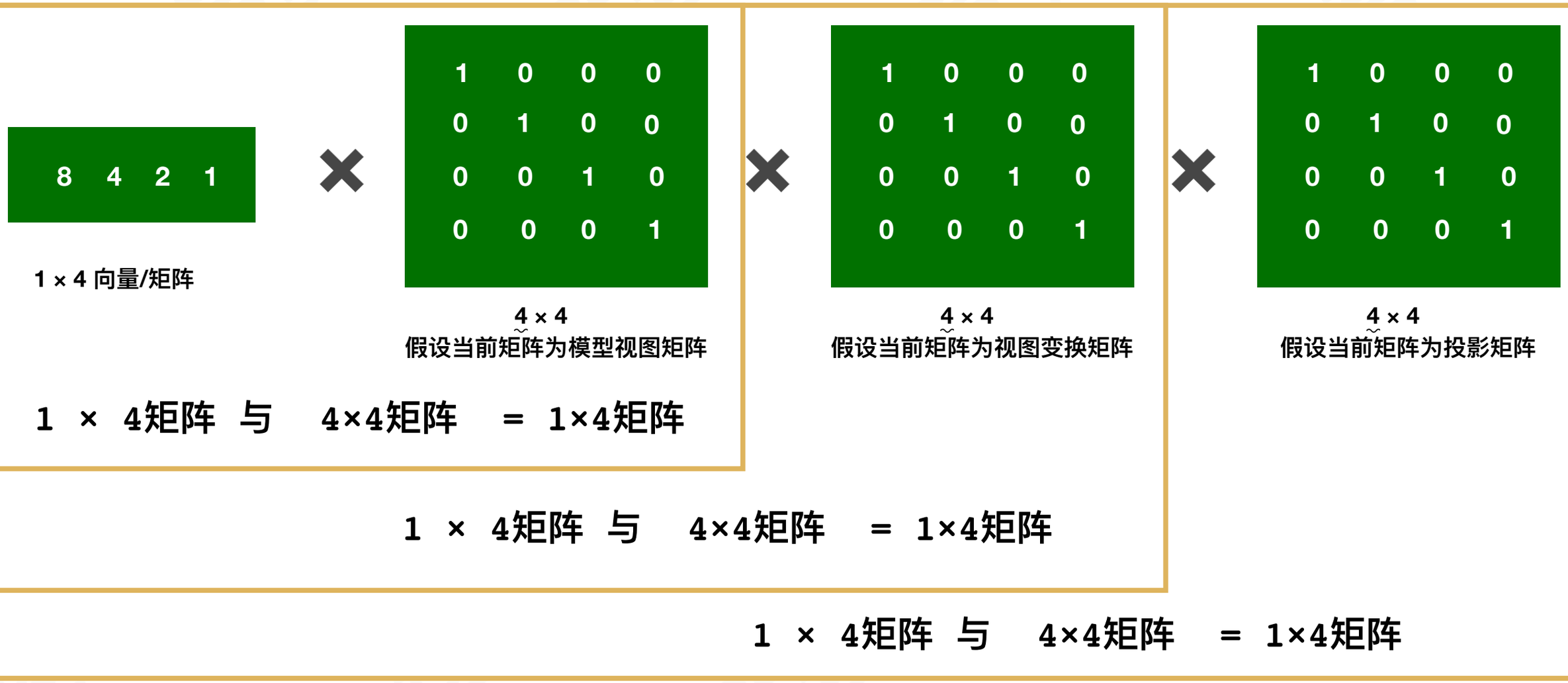
4 × 1 矩阵　　4 × 4 列矩阵-单元矩阵

课程研发:CC老师
课程授课:CC老师

单元矩阵 Matrix [线性代数角度]

在线性代数数学的维度,为了便于书写. 所以坐标计算. 都是从左往右顺序,进行计算. 如下列公式:

变换后顶点向量 = V_local * M_model * M_view * M_pro
变换后顶点向量 = 顶点 × 模型矩阵 × 观察矩阵 × 投影矩阵;

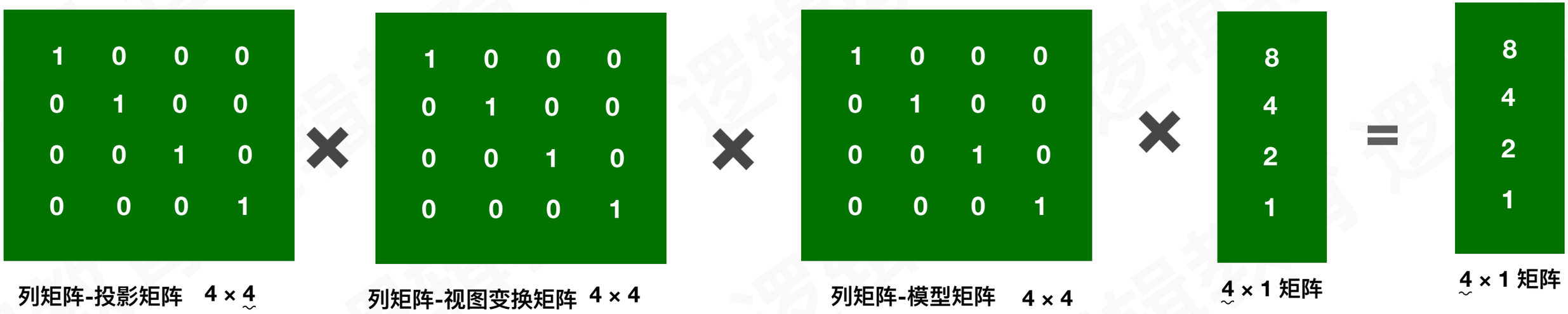


逻辑教育
Logic education

单元矩阵 Matrix [OpenGL角度]

在OpenGL 的维度. 如下列公式:

变换顶点向量 = M_pro * M_view * M_model * V_local
变换顶点向量 = 投影矩阵 × 视图变换矩阵 × 模型矩阵 × 顶点



列矩阵-投影矩阵 4 × 4

列矩阵-视图变换矩阵 4 × 4

列矩阵-模型矩阵 4 × 4

4 × 1 矩阵

4 × 1 矩阵

课程研发:CC老师
课程授课:CC老师

矩阵左乘

```
71
72          inline void MultMatrix(const M3DMatrix44f mMatrix) {
73              M3DMatrix44f mTemp;
74              m3dCopyMatrix44(mTemp, pStack[stackPointer]);
75              m3dMatrixMultiply44(pStack[stackPointer], mTemp, mMatrix);
76              }
77
78          inline void MultMatrix(GLFrame& frame) {
79              M3DMatrix44f m;
80              frame.GetMatrix(m);
```

1. 从栈顶获取栈顶矩阵 复制到 mTemp
2. 将栈顶矩阵 mTemp 左乘 mMatrix
3. 将结果放回栈顶空间里;

代码段

```
    //投影矩阵: projectionMatrix
    modelViewMatix.PushMatrix();
    M3DMatrix44f pm;
    projectionMatrix.GetMatrix(pm);
    modelViewMatix.MultMatrix(pm);


    //观察者(视图变换)矩阵:viewMatrix
    modelViewMatix.PushMatrix();
    M3DMatrix44f mCamera;
    cameraFrame.GetCameraMatrix(mCamera);
    modelViewMatix.MultMatrix(mCamera);


    //模型变换矩阵: modelMartix
    M3DMatrix44f mObjectFrame;
    viewFrame.GetMatrix(mObjectFrame);
    modelViewMatix.MultMatrix(mObjectFrame);
```

代码段 — 顶点着色器(glsl)

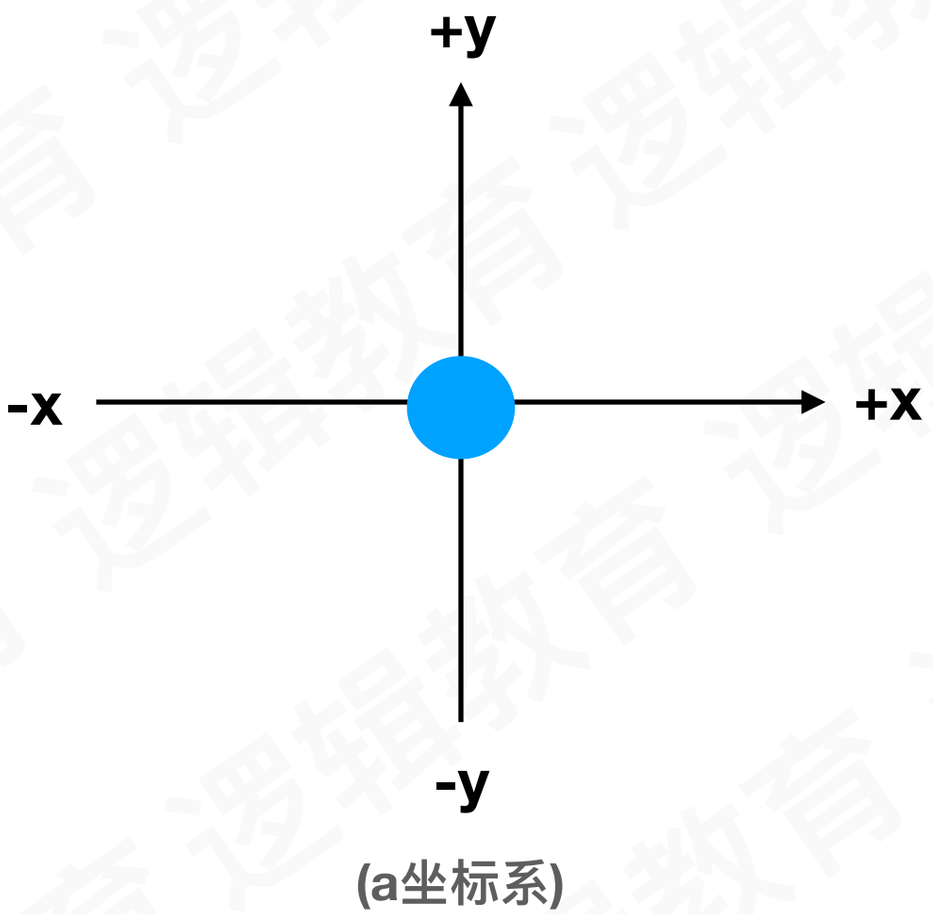
```glsl
attribute vec4 position;
attribute vec4 positionColor;

uniform mat4 projectionMatrix;
uniform mat4 modelViewMatrix;

varying lowp vec4 varyColor;

void main()
{
    varyColor = positionColor;

    vec4 vPos;

    //4*4 * 4*4 * 4*1
    vPos = projectionMatrix * modelViewMatrix * position;

    gl_Position = vPos;
}

```

理解 在OpenGL 里的变化

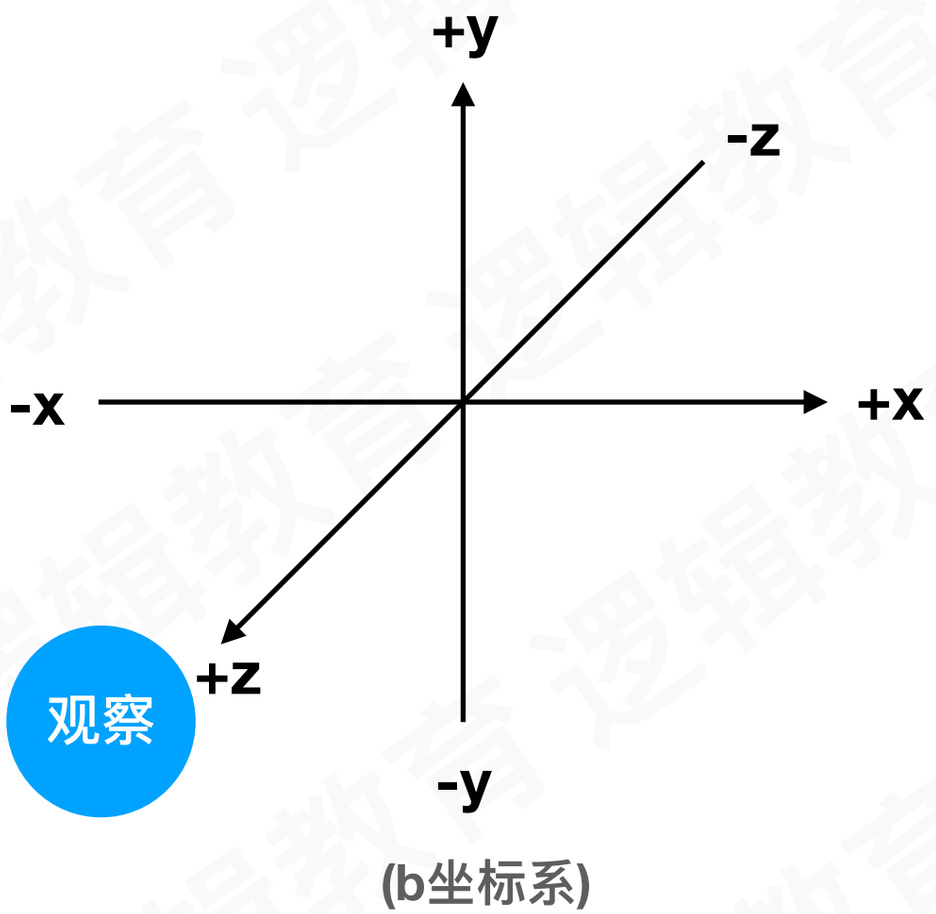
变换	解释
视图变换	指定观察者位置
模型变换	在场景中移动物体
模型视图	描述视图/模型变换的二元性(2种看到模型转换的方式)
投影	改变视景体大小和设置它的投影方式
视口	伪变化,对窗口上最终输出进行缩放

课程研发:CC老师
课程授课:CC老师

2个视角观察视觉坐标



(a坐标系)



(b坐标系)

观察(observer)

逻辑教育
Logic education

视图变换

视图变换是应用到场景中的第一种变换，它用来确定场景中的有利位置,在默认情况下，透视投影中位于原点(0,0,0),并沿着 z 轴负方向进行观察 (向显示器内部"看过去").

当观察者点位于原点(0,0,0) 时,就像在透视投影中一样.

视图变换将观察者放在你希望的任何位置.并允许在任何方向上观察场景，确定视图变换就像在场景中放置观察者并让它指向某一个方向;

从大局上考虑，在应用任何其他模型变换之前，必须先应用视图变换. 这样做是因为，对于视觉坐标系而言，视图变换移动了当前的工作的坐标系；后续的变化都会基于新调整的坐标系进行.

截自 <OpenGL 超级宝典 第5版 > 第96页

课程研发:CC老师
课程授课:CC老师

视图变换

(a) (b)

当我们利用 OpenGL 进行 3D 绘制时，就会使用笛卡尔坐标系。如果不进行任何变换，那么使用的坐标系将与刚刚描述的视觉坐标系相同。

4.3.2 视图变换

视图变换是应用到场景中的第一种变换。它用来确定场景中的有利位置。在默认情况下，透视投影中的观察点位于原点（0,0,0），并沿着 z 轴的负方向进行观察（向显示器内部“看进去”）。观察点相对于视觉坐标系进行移动，来提供特定的有利位置。当观察点位于原点（0,0,0）时，就像在透视投影中一样，绘制在 z 坐标为正的位置的对象则位于观察者背后。

然而在正投影中，观察者被认为是在 z 轴正方向无穷远的位置，能够看到视景体中的任何东西。

视图变换允许我们把观察点放在所希望的任何位置，并允许在任何方向上观察场景。确定视图变换就像在场景中放置照相机并让它指向某个方向。

从大局上考虑，在应用任何其他模型变换之前，必须先应用视图变换。这样做是因为，对于视觉坐标系而言，视图变换移动了当前的工作坐标系。所有后续变换随后都会基于新调整的坐标系进行。然后，在实际开始考虑如何进行这些变换时，就会更容易地看到这些变换是如何实现的了。

4.3.3 模型变换

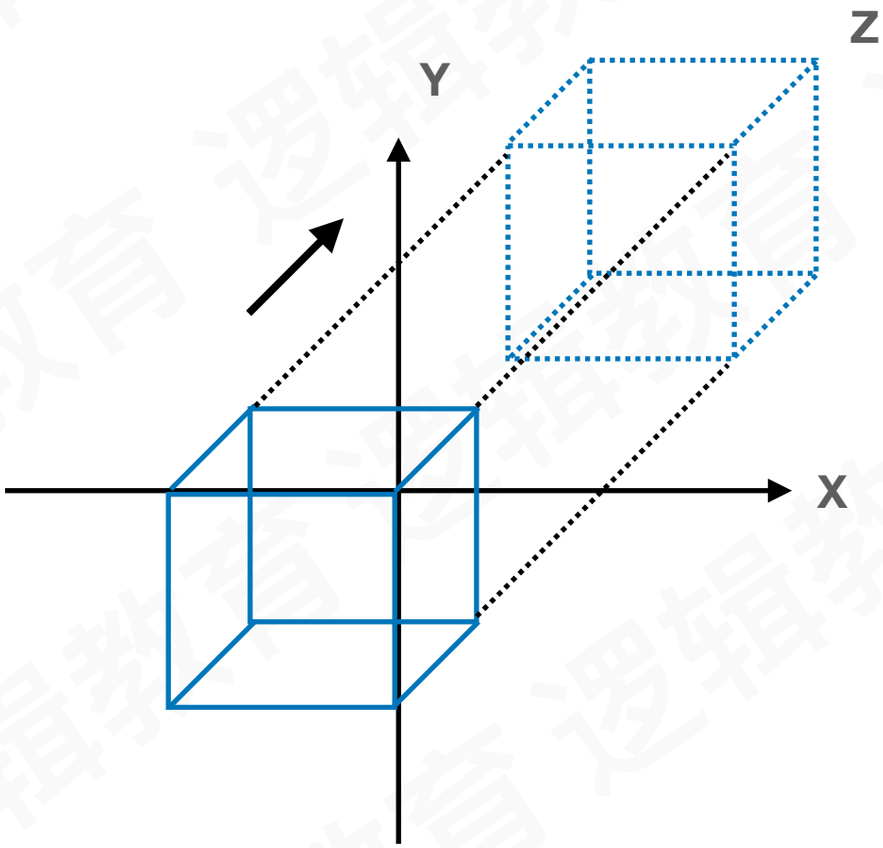
截自 <OpenGL 超级宝典 第5版 > 第96页

模型变换用于操纵模型和其中的特定对象。这些变换将对象移动
转和缩放。

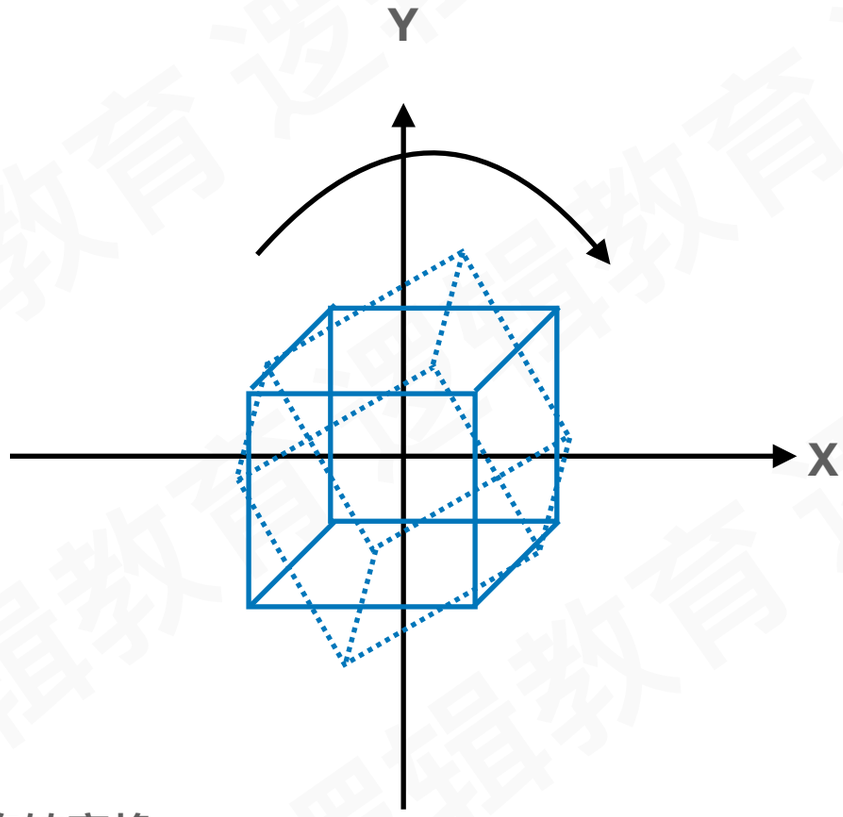
课程研发:CC老师
课程授课:CC老师

模型变换

模型变换：用于操纵模型与其中某特定变换． 这些变换将对象移动到需要的位置． 通过旋转,缩放,平移．



平移
图中对象沿着给定的轴进行移动

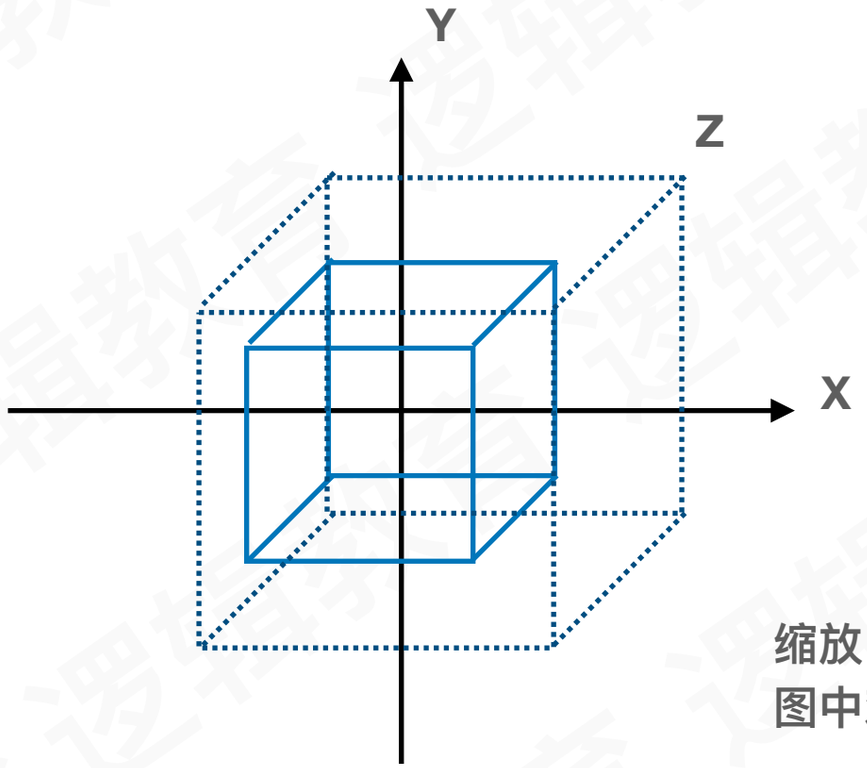


旋转变换
图中对象围绕一条坐标轴进行旋转

课程研发:CC老师
课程授课:CC老师

模型变换

模型变换：用于操纵模型与其中某特定变换． 这些变换将对象移动到需要的位置． 通过旋转,缩放,平移．

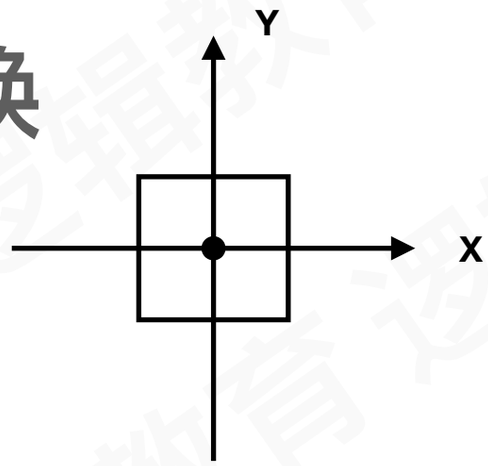


缩放
图中对象的大小进行了指定数量的放大或者缩小

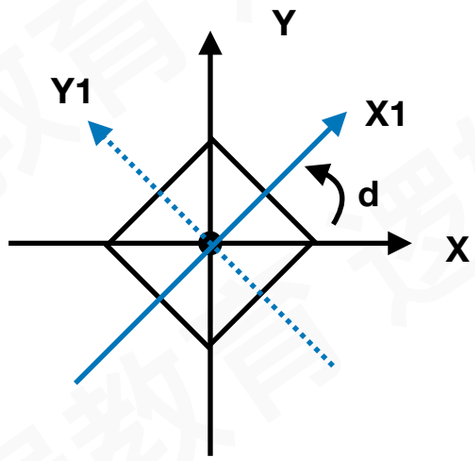
课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

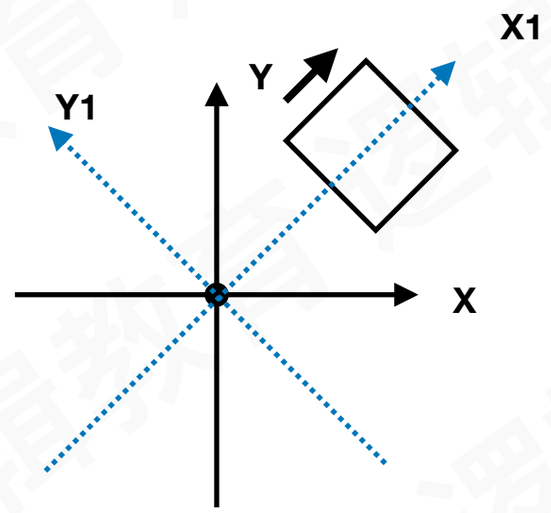
模型变换



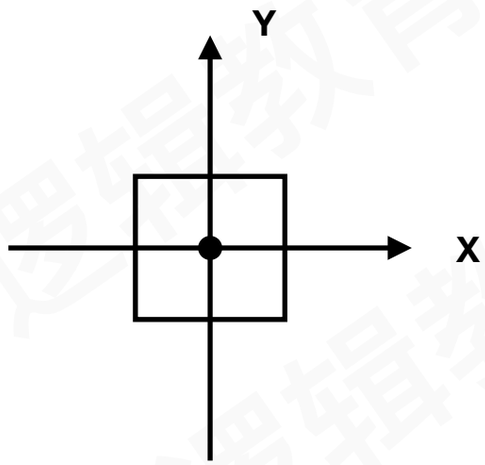
初始正方形



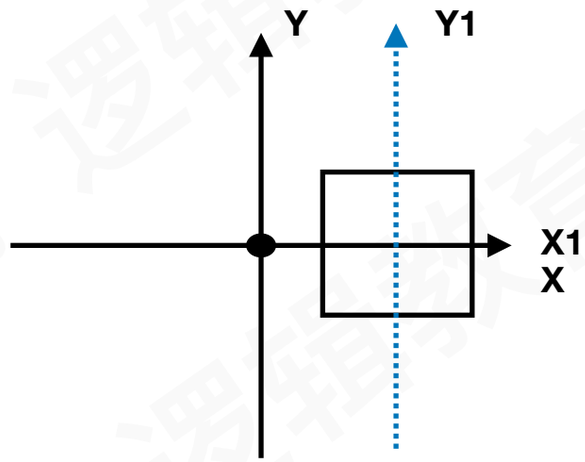
围绕z轴旋转得到新的x轴



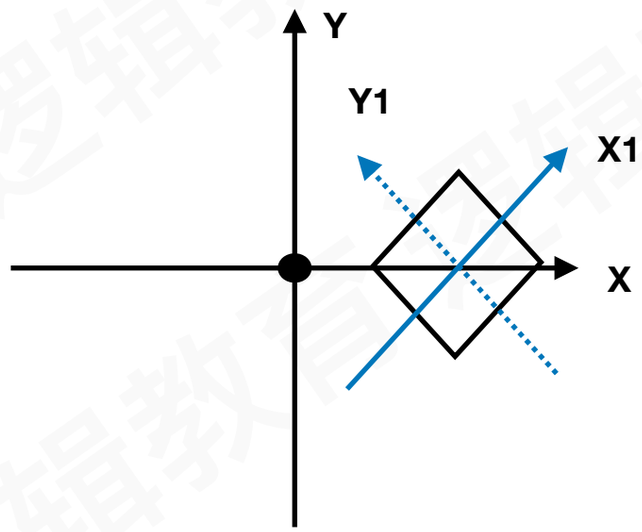
现在沿着X轴平移
其实是沿着X1轴平移



初始正方形



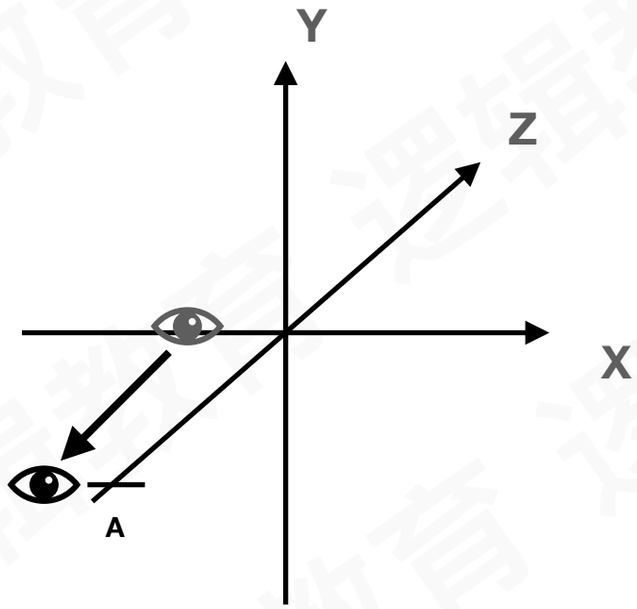
沿着X轴平移



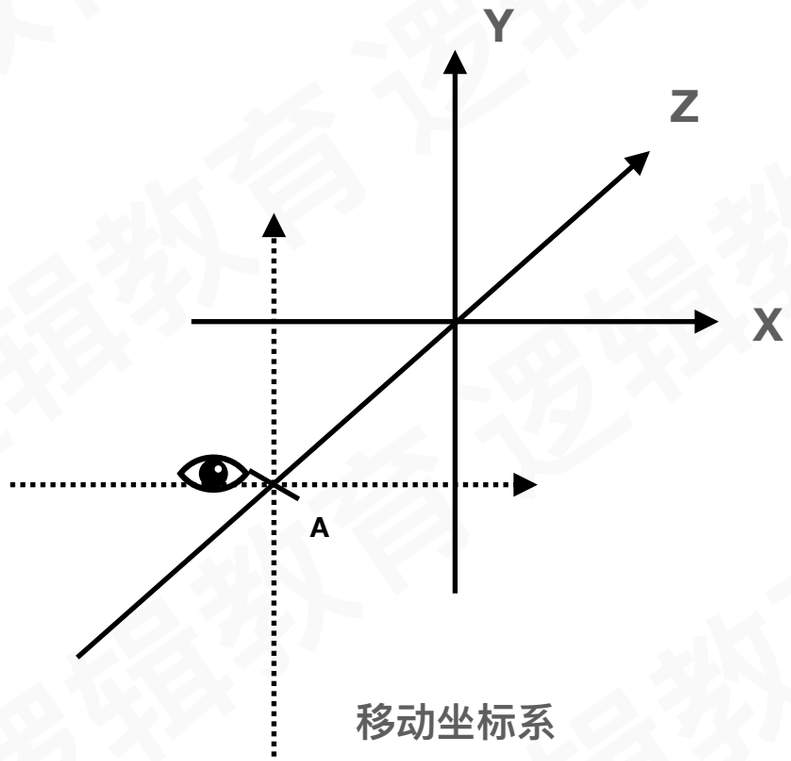
现在将平移后的坐标系进行旋转

课程研发:CC老师
课程授课:CC老师

两种看待模型变换的方式



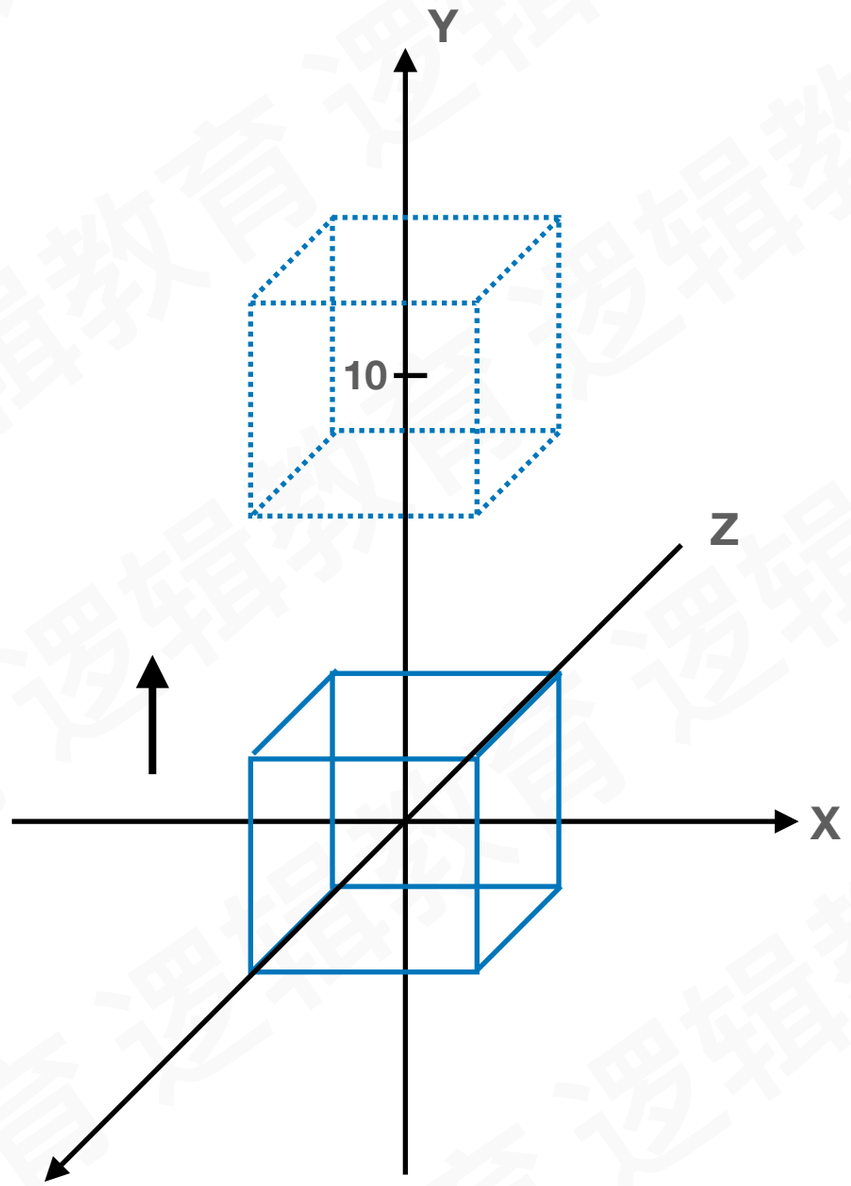
移动观察者



移动坐标系

课程研发:CC老师
课程授课:CC老师

模型变换-平移

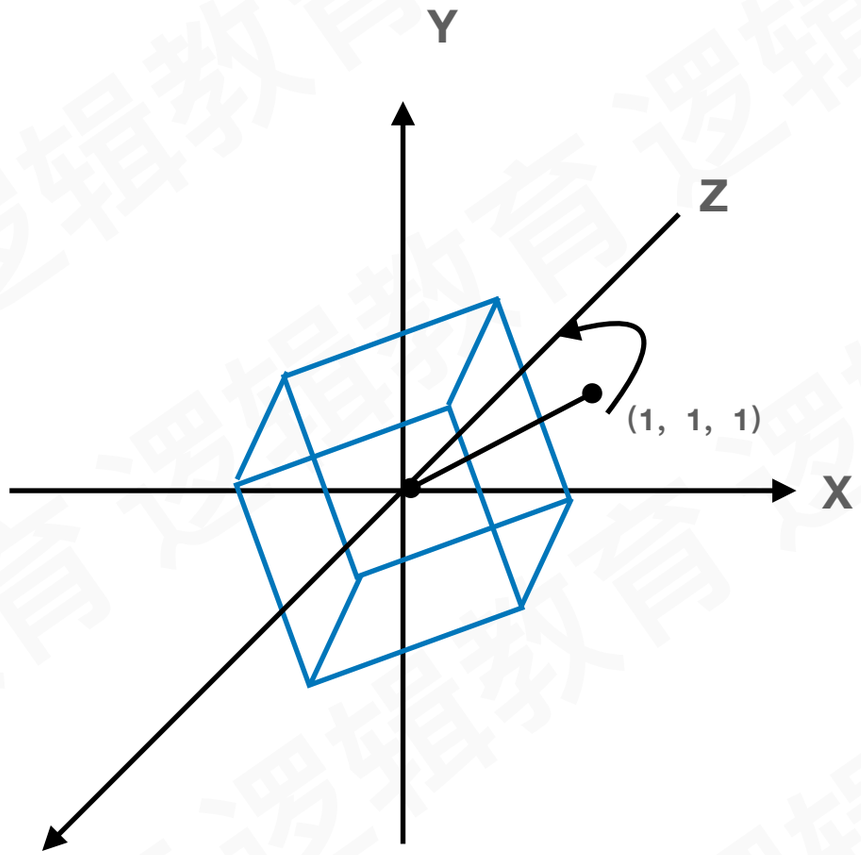


平移
图中对象沿着给定的Y轴进行移动10个单位

```
void m3dTranslationMatrix44(M3DMatrix44f m, floata x, float y, float z);
```

课程研发:CC老师
课程授课:CC老师

模型变换-旋转

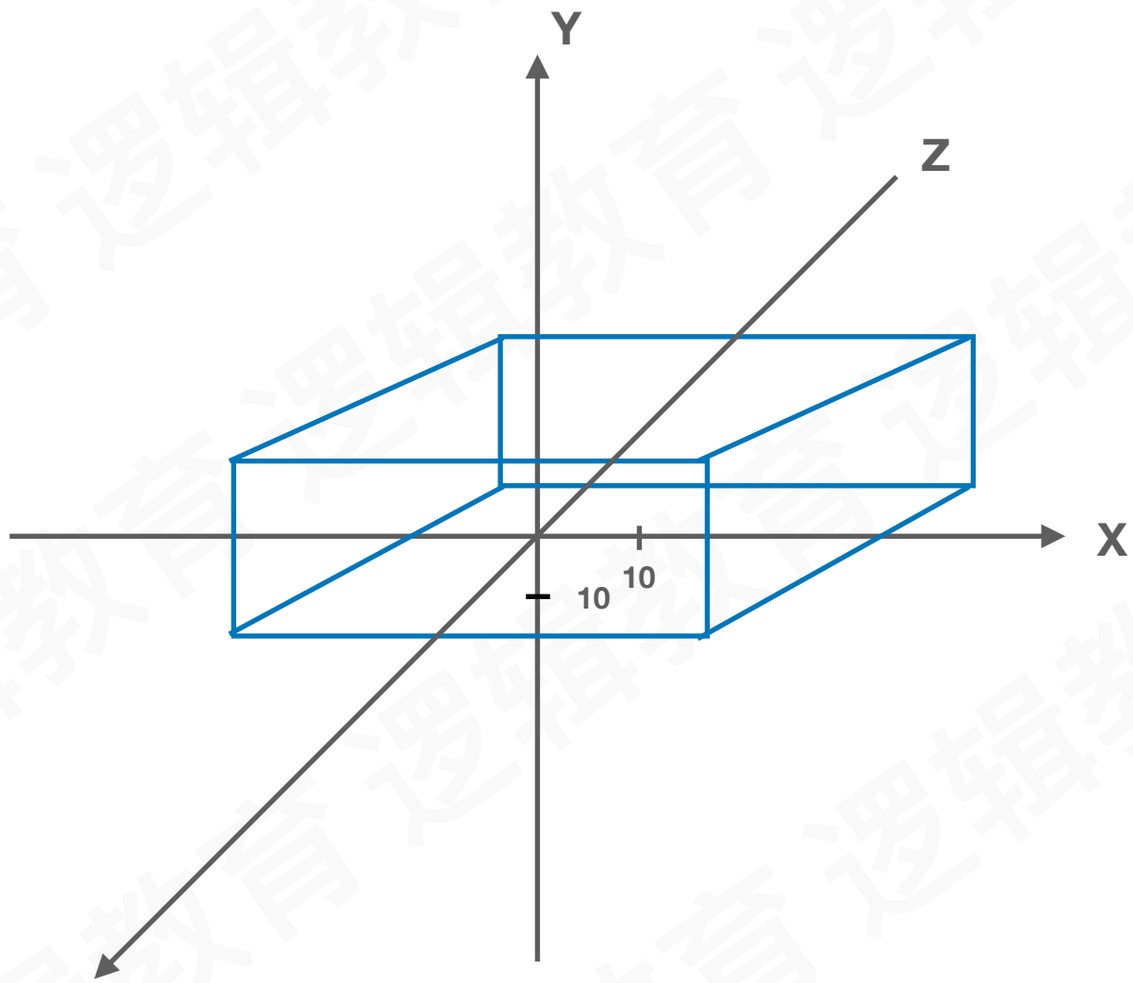


一个正方体围绕任意轴旋转

```
m3dRotationMatrix44(m3dDegToRad(45.0), floata x, float y, float z);
```

课程研发:CC老师
课程授课:CC老师

模型变换-缩放



立方体不一致缩放

```
void m3dScaleMatrix44(M3DMatrix44f m, floata xScale, float yScale, float zScale);
```

课程研发:CC老师
课程授课:CC老师

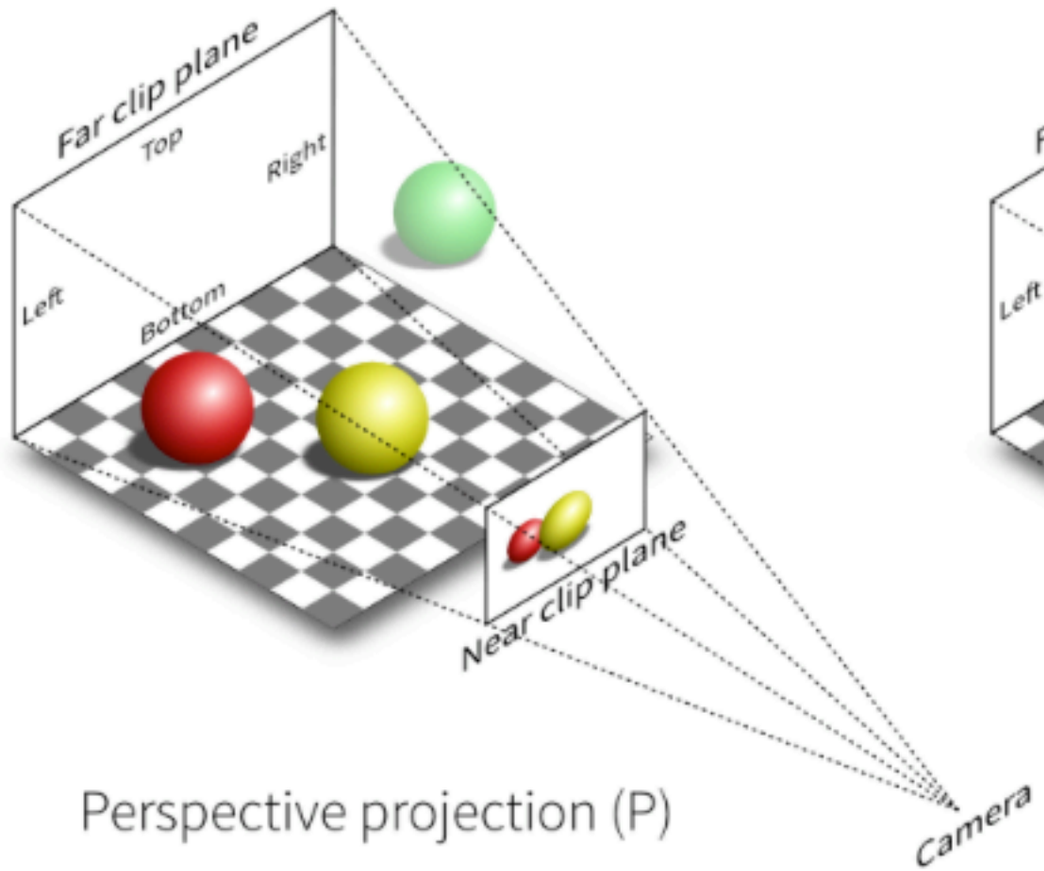
逻辑教育
Logic education

模型变换-综合变换

void m3dMatrixMultiply44(M3DMatrix44f product, const M3DMatrix44f a, const M3DMatrix44f b);

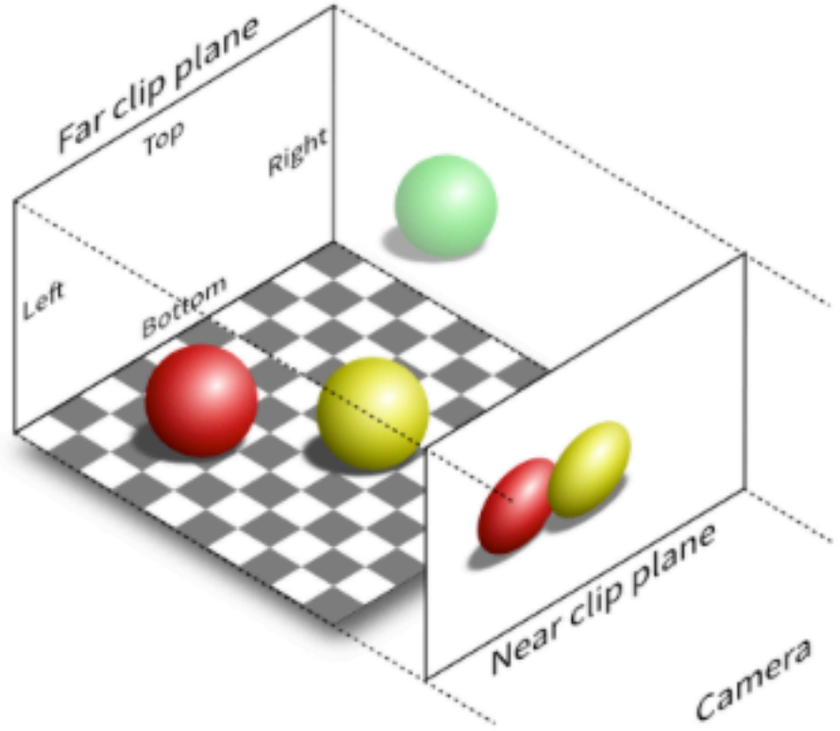
课程研发:CC老师
课程授课:CC老师

投影变换



Perspective projection (P)

透视投影



Orthographic projection (O)

正投影

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

案例演示

1. 使用矩阵的方式来实现正方形移动

2. 了解OpenGL 模型

课程研发:CC老师
课程授课:CC老师

矩阵堆栈 使用

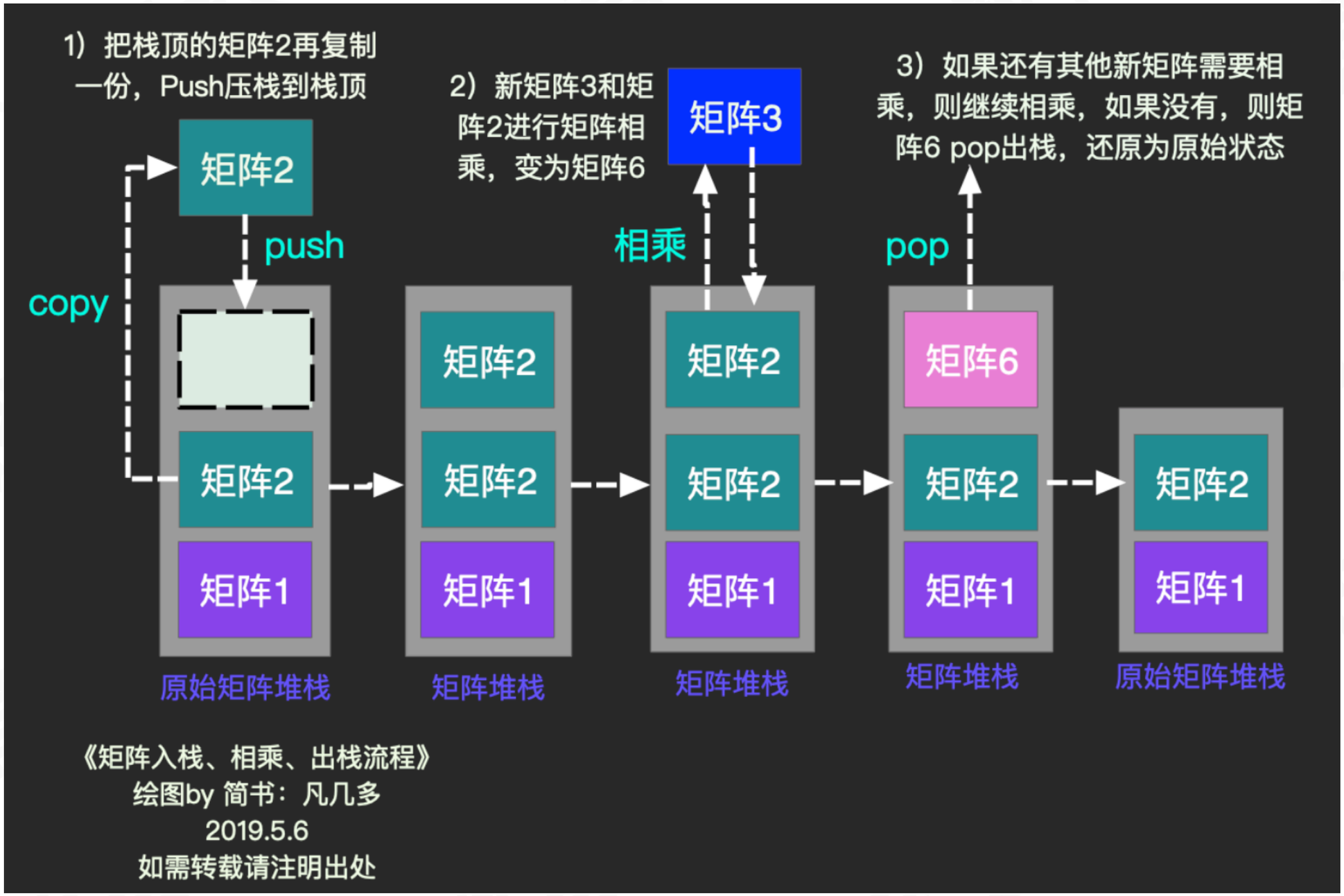
```
//类型
GLMatrixStack::GLMatrixStack(int iStackDepth = 64);

//在堆栈顶部载入一个单元矩阵
void GLMatrixStack::LoadIdentity(void);

//在堆栈顶部载入任何矩阵
//参数: 4*4矩阵
void GLMatrixStack::LoadMatrix(const M3DMatrix44f m);

//矩阵乘以矩阵堆栈顶部矩阵，相乘结果存储到堆栈的顶部
void GLMatrixStack::MultMatrix(const M3DMatrix44f);

//获取矩阵堆栈顶部的值 GetMatrix 函数
//为了适应GLShaderMananger的使用，或者获取顶部矩阵的副本

const M3DMatrix44f & GLMatrixStack::GetMatrix(void);
void GLMatrixStack::GetMatrix(M3DMatrix44f mMatrix);
```

课程研发:CC老师
课程授课:CC老师

矩阵堆栈 使用

```
//将当前矩阵压入堆栈(栈顶矩阵copy 一份到栈顶)
void GLMatrixStack::PushMatrix(void);

//将M3DMatrix44f 矩阵对象压入当前矩阵堆栈
void PushMatrix(const M3DMatrix44f mMatrix);

//将GLFame 对象压入矩阵对象
void PushMatrix(GLFame &frame);

//出栈（出栈指的是移除顶部的矩阵对象）
void GLMatrixStack::PopMatrix(void);
```

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

仿射变换

```
//Rotate 函数angle参数是传递的度数，而不是弧度
void MatrixStack::Rotate(GLfloat angle,GLfloat x,GLfloat y,GLfloat z);

void MatrixStack::Translate(GLfloat x,GLfloat y,GLfloat z);

void MatrixStack::Scale(GLfloat x,GLfloat y,GLfloat z);
```

课程研发:CC老师
课程授课:CC老师

2.1 使用照相机(摄像机) 和 角色帧 进行移动

```
class GLFrame
    {
    protected:
        M3DVector3f vOrigin;   // Where am I?
        M3DVector3f vForward;  // Where am I going?
        M3DVector3f vUp;       // Which way is up?

    }
```

2.3 GLFrame

```
//将堆栈的顶部压入任何矩阵
void GLMatrixStack::LoadMatrix(GLFrame &frame);
//矩阵乘以矩阵堆栈顶部的矩阵。相乘结果存储在堆栈的顶部
void GLMatrixStack::MultMatrix(GLFrame &frame);

//将当前的矩阵压栈
void GLMatrixStack::PushMatrix(GLFrame &frame);
```

课程研发:CC老师
课程授课:CC老师

2.4 照相机管理

```
//GLFrame函数，这个函数用来检索条件适合的观察者矩阵
void GetCameraMatrix(M3DMatrix44f m,bool bRotationOnly = flase);
```

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

see you next time ~

@ CC老师

全力以赴．非同凡"想"

课程研发:CC老师
课程授课:CC老师