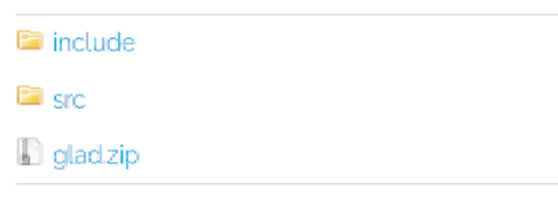
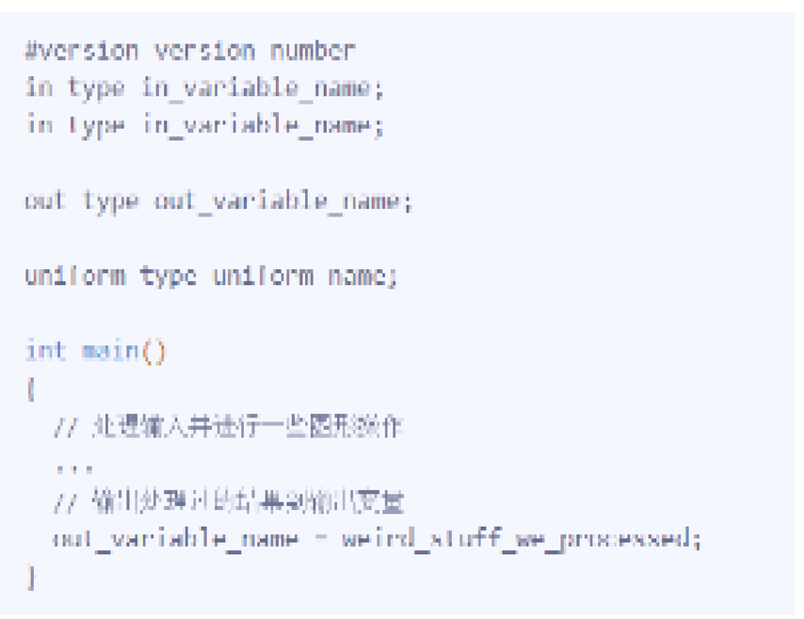
OpenGL

- 学习网址
- 入门
  - 第一节 OpenGL 简介
    - 什么是 OpenGL
      - OpenGL是图形硬件的一种软件接口
      - OpenGL 的设计目标就是作为一种流线型的、独立于硬件的接口，在许多不同的硬件平台上实现。
      - 一般它被认为是一个API(Application Programming Interface,应用程序编程接口)，包含了一系列可以操作图形、图像的函数
    - OpenGL 函数的语法
      - 使用了前缀“gl”，并把组成函数的每个单词的首字母用大写形式表示（例如，glClearColor()）
      - 使用前缀 GL_开头的常量，所有的单词都使用大写形式，并以下划线分隔（例如 GL_COLOR_BUFFER_BIT）
      - glColor3f()这种写法，其中3表示这个函数可以接受3个参数
    - OpenGL 是一个状态机
      - 可以设置它的各种状态（或模式），然后让这些状态一直生效，直到再次修改它们
        - 许多表示模式的状态变量可以用 glEnable()和 glDisable()函数进行启用和禁用
    - OpenGL 渲染管线
      - 纹理装配
        - OpenGL 应用程序可以在几何物体上应用纹理图像，使它们看上去更为逼真
    - 与 OpenGL 相关的函数库
      - GLU
        - GLU 函数都使用前缀 glu
        - 注意：Microsoft Windows 要求在 gl.h 或 glu.h 之前包含 windows.h 头文件，因为Microsoft Windows 版本的 gl.h 和 glu.h 文件内部使用的一些宏是在 windows.h 中定义的。
        - #include <GL/gl.h> #include <GL/glu.h>
      - GLX
        - GLX 函数都使用前缀 glX
        - #include <X11/Xlib.h> #include <GL/glx.h>
      - WGL
        - WGL函数库提供了 Windows 和 OpenGL 之间的接口。所有的 WGL 函数都使用前缀 wgl
        - 对于window 用户，可以使用#include <windows.h> 获取对WGL的访问
      - OpenGL Utility Toolkit（GLUT）
        - 目的是隐藏不同窗口系统 API所带来的复杂性
        - glut 为前缀
        - 如果想使用 GLUT 来实现窗口管理任务，应该包含这段代码：#include <freeglut.h>
    - 动画
      - 绝大多数 OpenGL 实现提供了双缓冲（硬件或软件），即提供了两个完整的颜色缓冲区。当一个缓冲区显示时，另一个缓冲区正在进行绘图。
    - OpenGL 及其废弃机制
      - 在 OpenGL 3.0 中，Khronos OpenGL ARB 工作组制定了一种废弃模式，该模式说明了如何从 API中删除功能。然而，这一修改不只是需要修改核心 OpenGL API，它也影响到如何创建 OpenGL 渲染环境以及可用的 OpenGL 渲染环境的类型。
      - 访问 OpenGL 函数
        - 根据用来开发应用程序的操作系统的不同，可能需要做一些额外的工作来访问某些 OpenGL 函数。
  - 第二节 创建一个窗口
    - 构建GLFW
      - https://www.glfw.org/download.html
      - 去上面那个网址，选择你需要的进行下载，可以选择预编译后的文件（我选的这个），或者可以选择源文件自己去cmake
    - GLAD
      - 因为opengl只提供一些接口，但是具体到特定品牌显卡上进行实验，还是需要程序员自己去编辑代码，这个工作就比较繁琐，因此GLAD的出现，就简便了程序员的工作
      - GLAD提供一个在在线服务，我们可以点击这个链接，将profile选择为core，gl版本可以根据你的需要进行选择（我无脑选择最新）。之后勾选generate a loader。我们就可以看到生成了三个文件。三个文件具体怎么处理，网上有很多教程，自行查看。

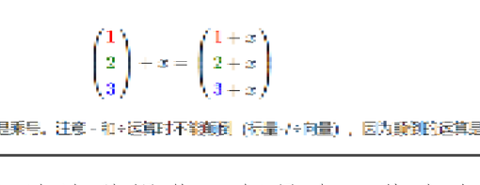


    - 这是一个案例（可以cmake的）
      - https://github.com/Polytonic/Glitter
  - 第三节 Hello Triangle
    - 开始绘制图形之前，我们需要先给OpenGL输入一些顶点数据。OpenGL是一个3D图形库，所以在OpenGL中我们指定的所有坐标都是3D坐标（x、y和z）。
    - 顶点着色器(Vertex Shader)是几个可编程着色器中的一个。如果我们打算做渲染的话，现代OpenGL需要我们至少设置一个顶点和一个片段着色器
    - 编译着色器
    - 片段着色器
      - 片段着色器所做的是计算像素最后的颜色输出
    - 链接顶点属性
    - 元素缓冲对象
  - 第四节 Shader
    - 着色器(Shader)是运行在GPU上的小程序。这些小程序为图形渲染管线的某个特定部分而运行。从基本意义上来说，着色器只是一种把输入转化为输出的程序。着色器也是一种非常独立的程序，因为它们之间不能相互通信；它们之间唯一的沟通只有通过输入和输出。
    - GLSL
      - 着色器是使用一种叫GLSL的类C语言写成的。GLSL是为图形计算量身定制的，它包含一些针对向量和矩阵操作的有用特性。
      - 典型的着色器结构

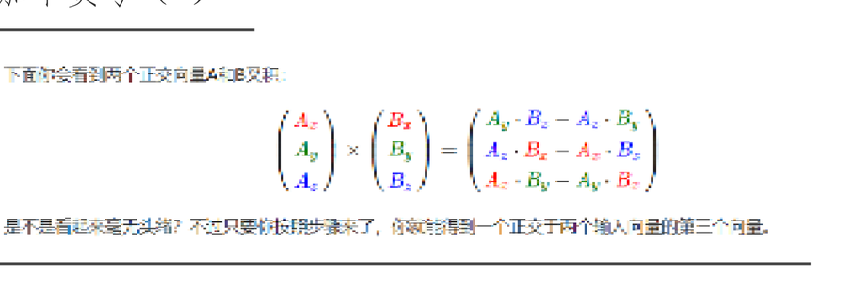


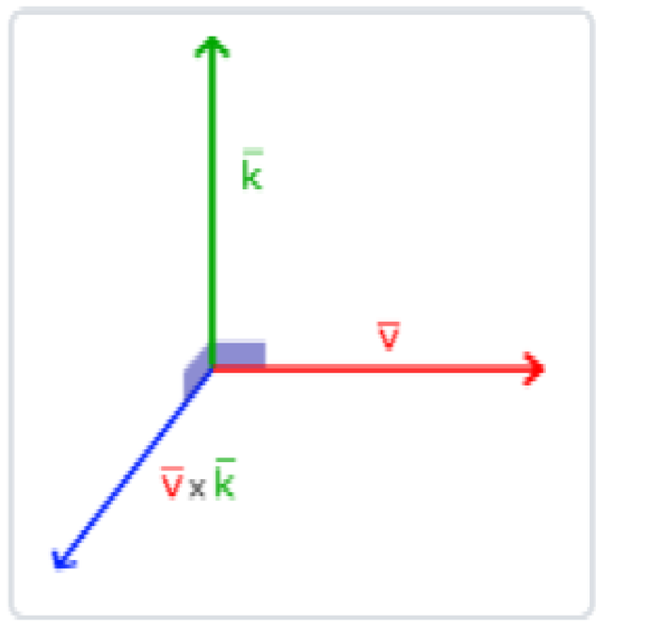
    - 数据类型
      - GLSL中包含C等其它语言大部分的默认基础数据类型：int、float、double、uint和bool。
      - 向量(Vector)

      - 矩阵(Matrix)
    - 自己编写shaderh
      - ps：个人还是不太懂这玩意怎么写
  - 第五节 Textures
    - 纹理就是给2D或者3D添加一些纹理细节。比如给一个空白位置添加瓷砖纹理。
    - 实践：https://github.com/DreamOneYou/OpenglLearn/textures.cpp
  - 第6节 Transformations
    - 通俗讲：使用矩阵对一个物体进行变换（Transformations）
    - 向量
      - 向量与标量运算

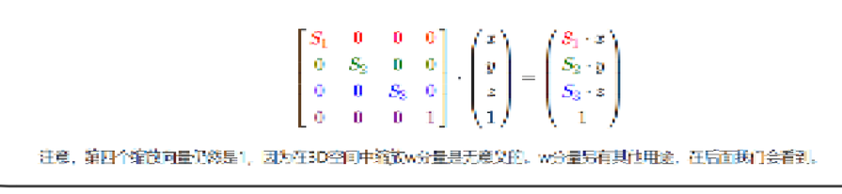


        - 数学上并没有加法这种操作，但是在一些库中有
      - 向量取反
        - 加个负号（-）
      - 向量叉乘

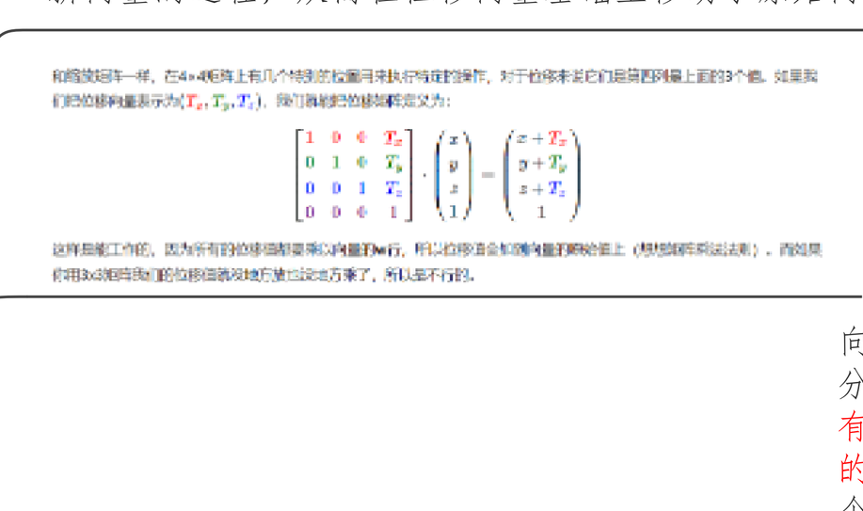




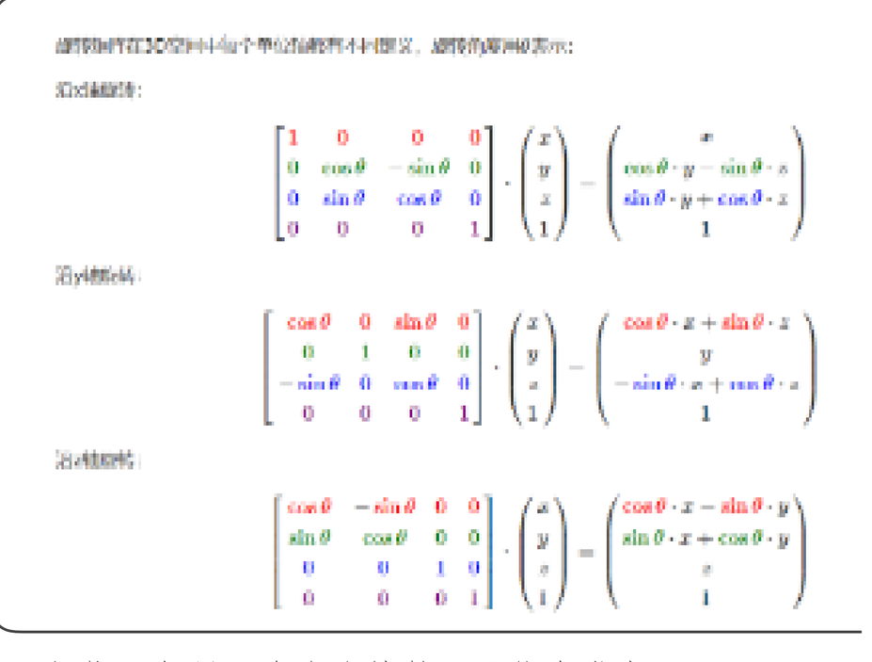
        - 叉乘只在3D空间中有定义，它需要两个不平行向量作为输入，生成一个正交于两个输入向量的第三个向量。如果输入的两个向量也是正交的，那么叉乘之后将会产生3个互相正交的向量。
    - 矩阵
    - 矩阵与向量相乘
      - 缩放
        - 对一个向量进行缩放(Scaling)就是对向量的长度进行缩放，而保持它的方向不变。



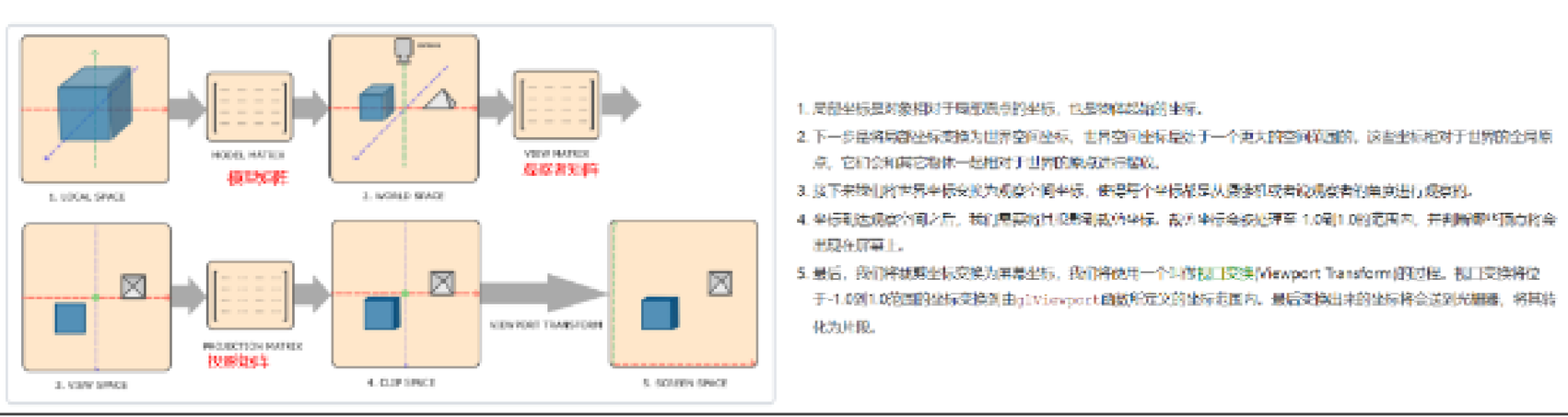
      - 位移
        - 位移(Translation)是在原始向量的基础上加上另一个向量从而获得一个在不同位置的新向量的过程，从而在位移向量基础上移动了原始向量。



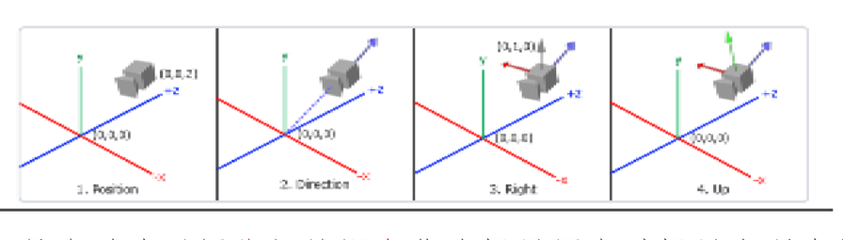
        - 齐次坐标(Homogeneous Coordinates)
          - 向量的w分量也叫齐次坐标。想要从齐次向量得到3D向量，我们可以把x、y和z坐标分别除以w坐标。我们通常不会注意这个问题，因为w分量通常是1.0。使用齐次坐标有几点好处：它允许我们在3D向量上进行位移（如果没有w分量我们是不能位移向量的），而且下一章我们会用w值创建3D视觉效果。如果一个向量的齐次坐标是0，这个坐标就是方向向量(Direction Vector)，因为w坐标是0，这个向量就不能位移（译注：这也就是我们说的不能位移一个方向）。
      - 旋转
        - 2D或3D空间中的旋转用角(Angle)来表示。角可以是角度制或弧度制的，周角是360角度或2PI弧度。其中PI值为3.14159...



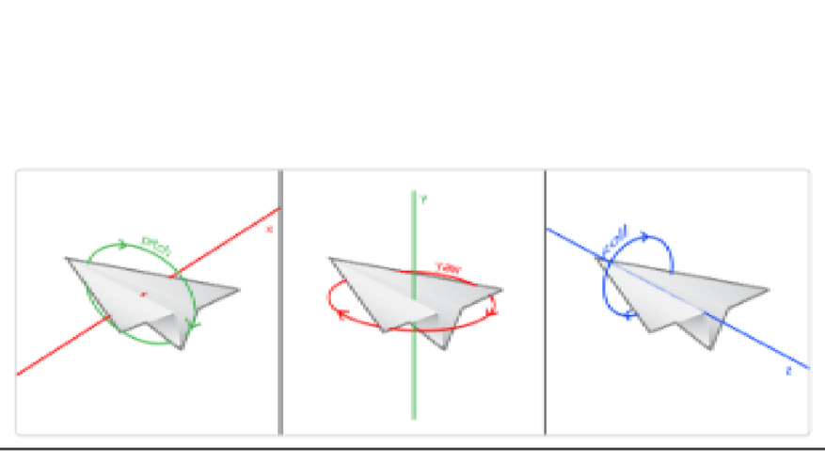
        - 问题：如果三个方向旋转，可能会带来Gimbal Lock
    - 矩阵的组合
      - 使用矩阵进行变换的真正力量在于，根据矩阵之间的乘法，我们可以把多个变换组合到一个矩阵中
      - 变换顺序：建议在组合矩阵时，先进行缩放操作，然后是旋转，最后才是位移，否则它们会（消极地）互相影响。比如，如果你先位移再缩放，位移的向量也会同样被缩放
  - 第7节 Coordinate Systems
    - OpenGL希望在每次顶点着色器运行后，我们可见的所有顶点都为标准化设备坐标(Normalized Device Coordinate, NDC)。也就是说，每个顶点的x，y，z坐标都应该在-1.0到1.0之间，超出这个坐标范围的顶点都将不可见。
    - 5个不同的坐标系统
      - 局部空间(Local Space，或者称为物体空间(Object Space))
      - 世界空间(World Space)
      - 观察空间(View Space，或者称为视觉空间(Eye Space))
      - 裁剪空间(Clip Space)
      - 屏幕空间(Screen Space)
    - 概述



      - 将一个坐标系变换到另一个坐标系，最重要的三个矩阵分别是：模型（model），观察者（view）和投影（projection）
    - 局部空间
      - 指物体所在的坐标空间，即对象最开始所在的地方。甚至有可能你创建的所有模型都以(0,0,0)为初始位置（译注：然而它们会最终出现在世界的不同位置）
    - 世界空间
      - 如果我们将我们所有的物体导入到程序当中，它们有可能会全挤在世界的原点(0,0,0)上，这并不是我们想要的结果。我们想为每一个物体定义一个位置，从而能在更大的世界当中放置它们。世界空间中的坐标正如其名：是指顶点相对于（游戏）世界的坐标。如果你希望将物体分散在世界上摆放（特别是非常真实的那样），这就是你希望物体变换到的空间。物体的坐标将会从局部变换到世界空间；该变换是由模型矩阵(Model Matrix)实现的。
      - 模型矩阵是一种变换矩阵，它能通过对物体进行位移、缩放、旋转来将它置于它本应该在的位置或朝向。你可以将它想像为变换一个房子，你需要先将它缩小（它在局部空间中太大了），并将其位移至郊区的一个小镇，然后在y轴上往左旋转一点以搭配附近的房子。你也可以把上一节将箱子到处摆放在场景中用的那个矩阵大致看作一个模型矩阵；我们将箱子的局部坐标变换到场景/世界中的不同位置。
    - 观察空间
      - 观察空间是将世界空间坐标转化为用户视野前方的坐标而产生的结果。因此观察空间就是从摄像机的视角所观察到的空间。而这通常是由一系列的位移和旋转的组合来完成，平移/旋转场景从而使得特定的对象被变换到摄像机的前方。这些组合在一起的变换通常存储在一个观察矩阵(View Matrix)里，它被用来将世界坐标变换到观察空间。
    - 裁剪空间
      - 在一个顶点着色器运行的最后，OpenGL期望所有的坐标都能落在一个特定的范围内，且任何在这个范围之外的点都应该被裁剪掉(Clipped)。被裁剪掉的坐标就会被忽略，所以剩下的坐标就将变为屏幕上可见的片段。这也就是裁剪空间(Clip Space)名字的由来
      - 为了将顶点坐标从观察变换到裁剪空间，我们需要定义一个投影矩阵(Projection Matrix)，它指定了一个范围的坐标，比如在每个维度上的-1000到1000。投影矩阵接着会将在这个指定的范围内的坐标变换为标准化设备坐标的范围(-1.0, 1.0)。所有在范围外的坐标不会被映射到在-1.0到1.0的范围之间，所以会被裁剪掉。在上面这个投影矩阵所指定的范围内，坐标(1250, 500, 750)将是不可见的，这是由于它的x坐标超出了范围，它被转化为一个大于1.0的标准化设备坐标，所以被裁剪掉了。
      - 观察坐标变换为裁剪坐标的投影矩阵可以为两种不同的形式
        - 正射投影矩阵(Orthographic Projection Matrix)
        - 透视投影矩阵(Perspective Projection Matrix)
    - 组合
      - $V_{clip} = M_{projection} \cdot M_{view} \cdot M_{model} \cdot V_{local}$ ，我们需要从右往左阅读矩阵的乘法
  - 第8节 Camera
    - 摄像机/观察空间



      - 是在讨论以摄像机的视角作为场景原点时场景中所有的顶点坐标：观察矩阵把所有的世界坐标变换为相对于摄像机位置与方向的观察坐标。
      - 要定义一个摄像机，我们需要它在世界空间中的位置、观察的方向、一个指向它右侧的向量以及一个指向它上方的向量
    - 自由移动
    - 移动速度
    - 视角移动
    - 欧拉角
      - 3种欧拉角



        - 俯仰角(Pitch)、偏航角(Yaw)和滚转角(Roll)
        - 俯仰角是描述我们如何往上或往下看的角，可以在第一张图中看到。第二张图展示了偏航角，偏航角表示我们往左和往右看的程度。滚转角代表我们如何翻滚摄像机，通常在太空飞船的摄像机中使用。每个欧拉角都有一个值来表示，把三个角结合起来我们就能够计算3D空间中任何的旋转向量了。
    - 鼠标输入
      - 偏航角和俯仰角是通过鼠标（或手柄）移动获得的，水平的移动影响偏航角，竖直的移动影响俯仰角
      - 原理：储存上一帧鼠标的位置，在当前帧中我们当前计算鼠标位置与上一帧的位置相差多少。如果水平/竖直差别越大那么俯仰角或偏航角就改变越大，也就是摄像机需要移动更多的距离。
    - 缩放
      - 当视野变小时，场景投影出来的空间就会减小，产生放大(Zoom In)了的感觉
  - 第9节 复习