逻辑教育
Logic education

Hello 视觉全训班

视觉全训班.OpenGL 图形专有名词与坐标解析

@ 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

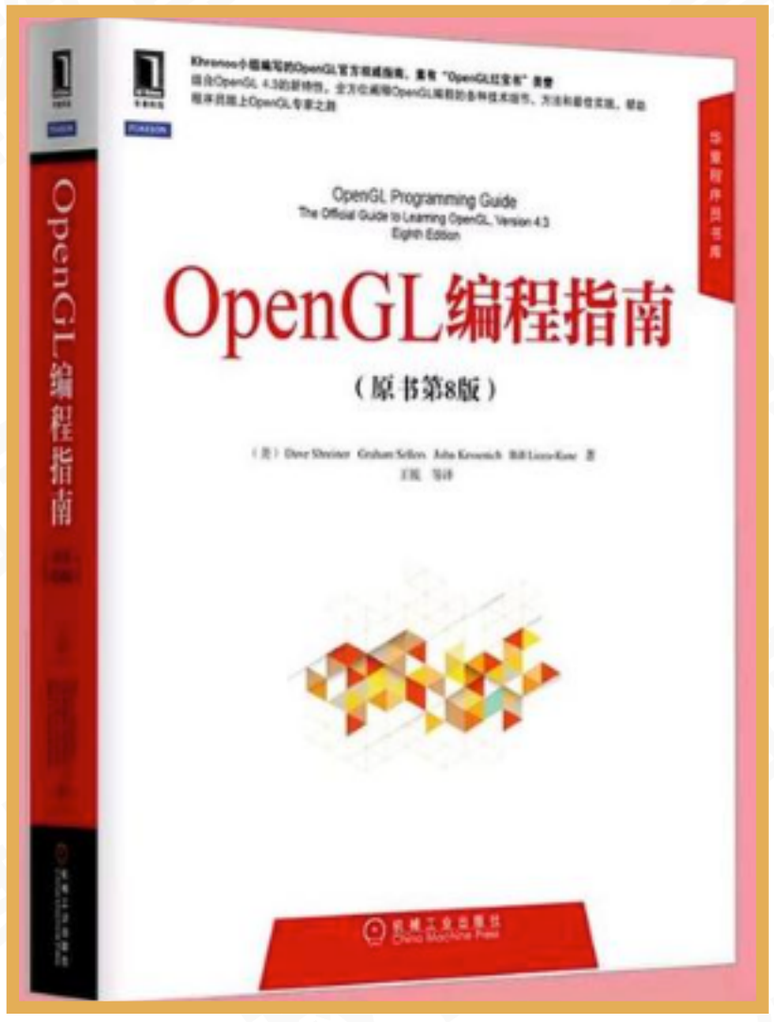
逻辑教育
Logic education

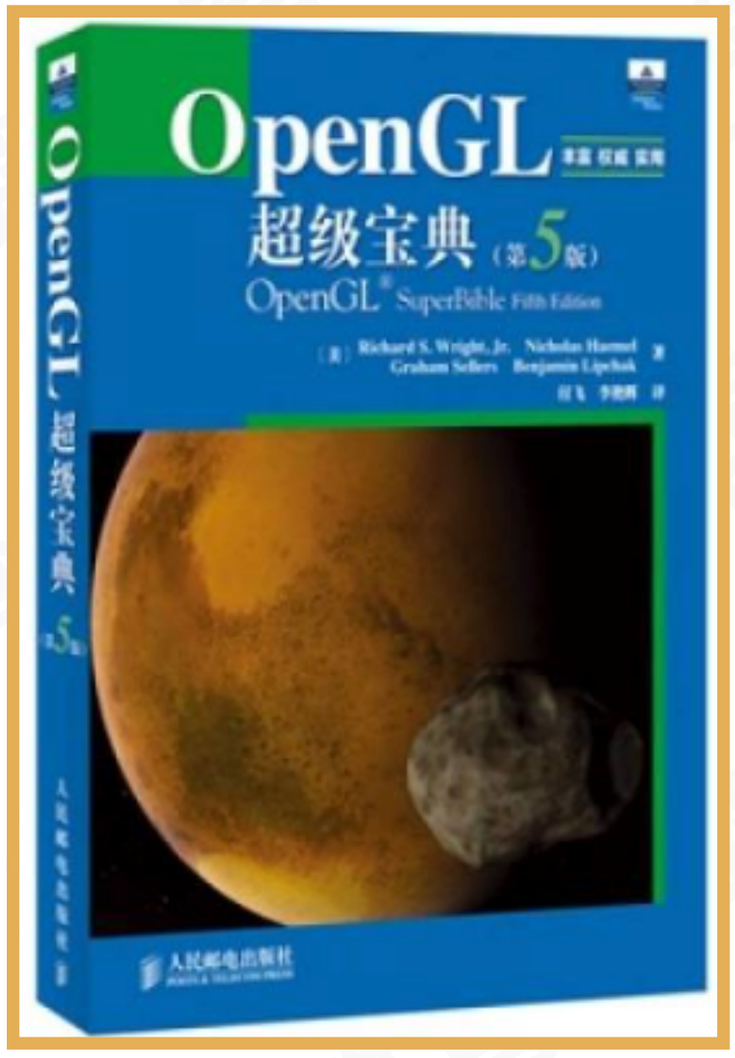
课堂目标:

1. 图形API简介
2. 图形API解决那些问题?
3. OpenGL 中专业名词解析
4. OpenGL坐标系解析
5. 图形/图片从文件渲染到屏幕过程解析
6. 案例01: 在存储着色器的情况下渲染图形,并通过键盘移动该图形.

课程研发:CC老师
课程授课:CC老师

OpenGL 两大”圣经”





课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

图形API简介

- OpenGL （Open Graphics Library）是一个跨编程语言、跨平台的编程图形程序接口，它将计算机的资源抽象称为一个个OpenGL的对象，对这些资源的操作抽象为一个个的OpenGL指令
- OpenGL ES （OpenGL for Embedded Systems）是 OpenGL 三维图形 API 的子集，针对手机、PDA和游戏主机等嵌入式设备而设计，去除了许多不必要和性能较低的API接口。
- DirectX 是由很多API组成的，DirectX并不是一个单纯的图形API. 最重要的是DirectX是属于Windows上一个多媒体处理框架.并不支持Windows以外的平台,所以不是跨平台框架. 按照性质分类，可以分为四大部分，显示部分、声音部分、输入部分和网络部分.
- Metal : *Metal: Apple*为游戏开发者推出了新的平台技术 *Metal*，该技术能够为 *3D* 图像提高 *10* 倍的渲染性能.*Metal* 是*Apple*为了解决*3D*渲染而推出的框架

课程研发:CC老师
课程授课:CC老师

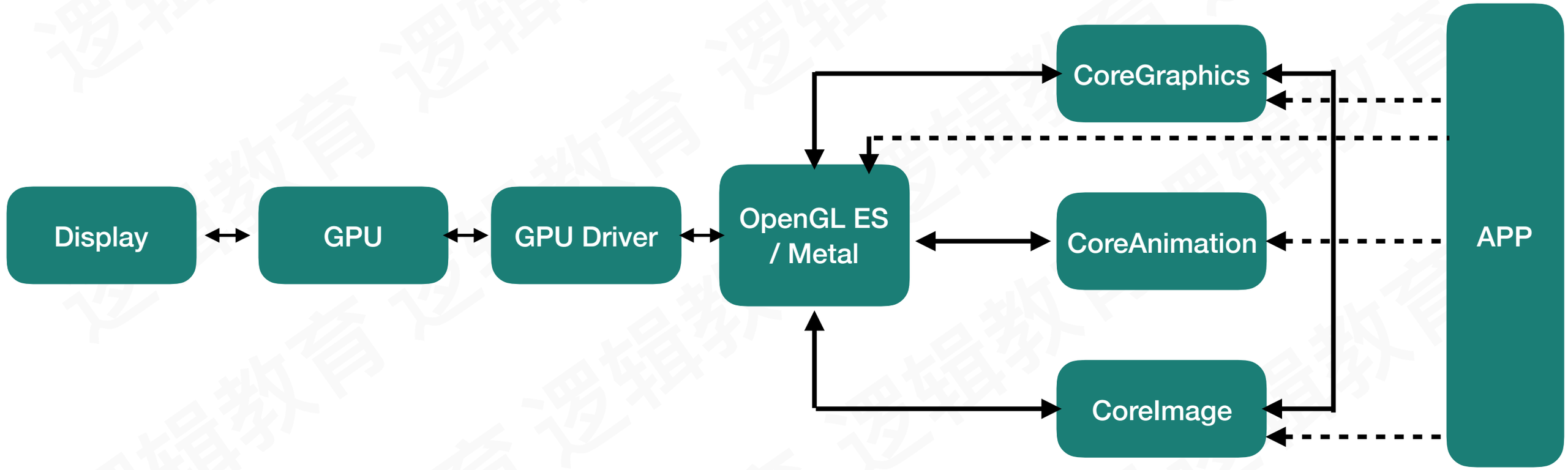
逻辑教育
Logic education

课堂练习:

- 下面说明中正确的是 ( )
  - A. 开发者可以在Mac 程序中使用OpenGL 来实现图形渲染
  - B. 开发者只能使用Metal 在Apple 平台下实现图形渲染
  - C. 开发者们能使用DirectX 在Apple 平台下实现图形渲染
  - D. OpenGL 和 OpenGL ES 并没有差别

- 下面说明中正确的是 ()
  - A. 2018年后Apple 开发者不能在 Apple 平台下使用OpenGL /OpenGL ES
  - B. 苹果底层渲染是由Metal 来实现的
  - C. OpenGL ES 只是比OpenGL 减少了一部分子集
  - D. 有了Metal , iOS 开发者没有必要学习OpenGL ES

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

Metal & OpenGL ES

Deprecated != Unserviceable

课程研发:CC老师
课程授课:CC老师

图形API目的是解决什么问题

简单来说就是实现图形的底层渲染

A. 比如在游戏开发中,对于游戏场景/游戏人物的渲染

B. 比如在音视频开发中,对于视频解码后的数据渲染

C. 比如在地图引擎,对于地图上的数据渲染

D. 比如在动画中,实现动画的绘制

E. 比如在视频处理中,对于视频加上滤镜效果

OpenGL /OpenGL ES/ Metal 在任何项目中解决问题的本质

就是利用GPU芯片来高效渲染图形图像.

图形API 是iOS开发者唯一接近GPU的方式.

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

学习要求:

OpenGL 阶段:

- 熟悉图形图像API中的专有名词
- 熟悉图形图像常用处理手段,比如深度测试等.
- 熟悉图形渲染流程
- 熟悉纹理坐标 使用

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

学习要求:

OpenGL ES 阶级:

- 熟练掌握 GLSL 语法
- 熟悉 GLKit 框架
- OpenGL ES 渲染流程
- 能通过案例灵活运用API

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

学习要求:

Metal 阶级:

- 熟悉Metal shading language
- 熟悉Metal kit
- 熟悉Metal 渲染流程
- 能通过案例灵活运用API

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

课堂练习:

- 下面说明中正确的是( )

  A. 音视频开发有必要学习OpenGL ES

  B. ARKit 中关于3D图形渲染不能用OpenGL ES 渲染

  C. Cocos2D 框架底层中需要依赖于OpenGL ES

  D. Metal 能够替代OpenGL ES

课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

- OpenGL 上下文〔context〕
  - 在应用程序调用任何OpenGL的指令之前，需要安排首先创建一个OpenGL的上下文。这个上下文是一个非常庞大的状态机，保存了OpenGL中的各种状态，这也是OpenGL指令执行的基础
  - OpenGL的函数不管在哪个语言中，都是类似C语言一样的面向过程的函数，本质上都是对OpenGL上下文这个庞大的状态机中的某个状态或者对象进行操作，当然你得首先把这个对象设置为当前对象。因此，通过对OpenGL指令的封装，是可以将OpenGL的相关调用封装成为一个面向对象的图形API的
  - 由于OpenGL上下文是一个巨大的状态机，切换上下文往往会产生较大的开销，但是不同的绘制模块，可能需要使用完全独立的状态管理。因此，可以在应用程序中分别创建多个不同的上下文，在不同线程中使用不同的上下文，上下文之间共享纹理、缓冲区等资源。这样的方案，会比反复切换上下文，或者大量修改渲染状态，更加合理高效的.

课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

- OpenGL 状态机
  - 状态机是理论上的一种机器.这个非常难以理解.所以我们把这个状态机这么理解.状态机描述了一个对象在其生命周期内所经历的各种状态，状态间的转变，发生转变的动因，条件及转变中所执行的活动。或者说，状态机是一种行为，说明对象在其生命周期中响应事件所经历的状态序列以及对那些状态事件的响应。因此具有以下特点：
    - 有记忆功能，能记住其当前的状态
    - 可以接收输入，根据输入的内容和自己的原先状态，修改自己当前状态，并且可以有对应输出
    - 当进入特殊状态（停机状态）的时候，变不再接收输入，停止工作；

课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

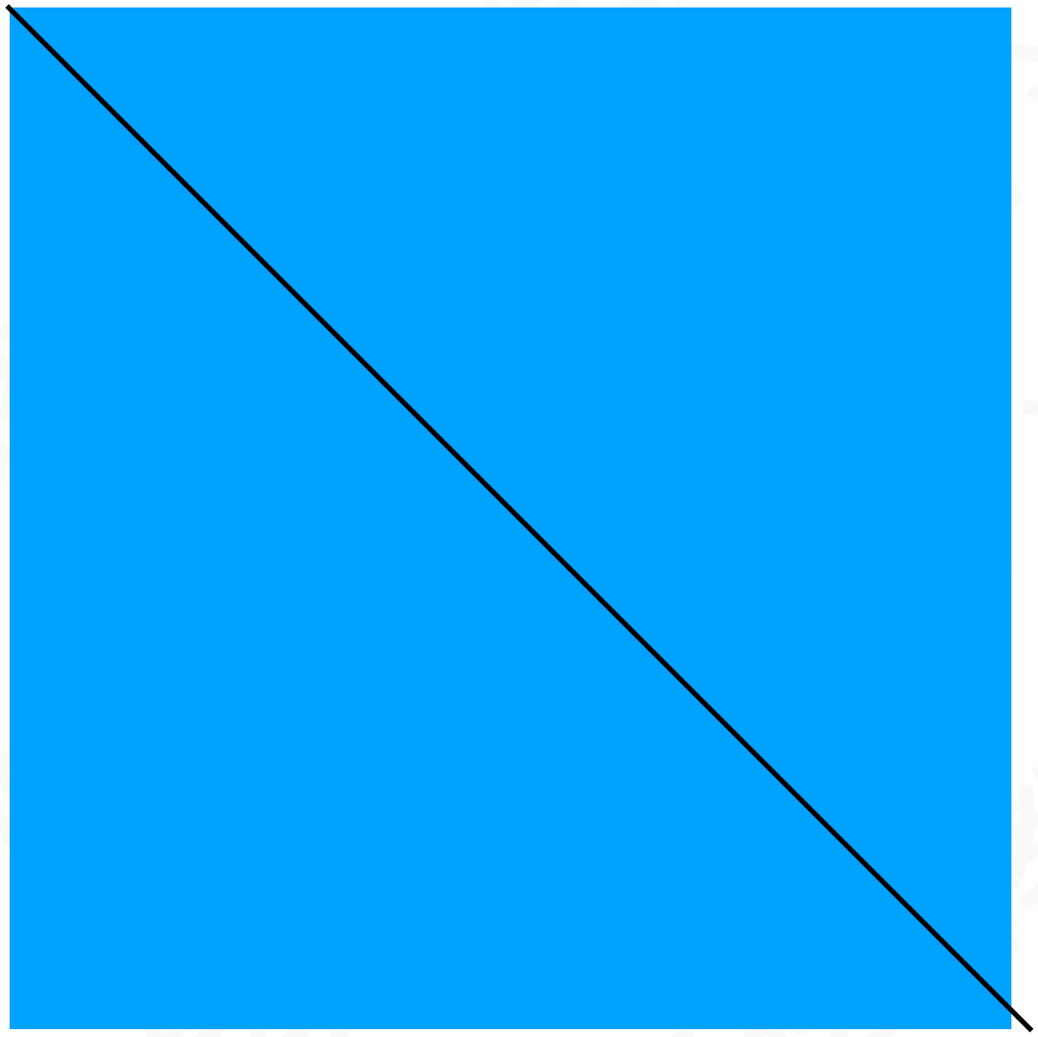
- OpenGL 状态机
  - 类推到OpenGL 中来,可以这么理解：
    - OpenGL可以记录自己的状态（如当前所使用的颜色、是否开启了混合功能等）
    - OpenGL可以接收输入（当调用OpenGL函数的时候，实际上可以看成OpenGL在接收我们的输入），如我们调用glColor3f，则OpenGL接收到这个输入后会修改自己的“当前颜色”这个状态；
    - OpenGL可以进入停止状态，不再接收输入。在程序退出前，OpenGL总会先停止工作的；

课程研发:CC老师
课程授课:CC老师

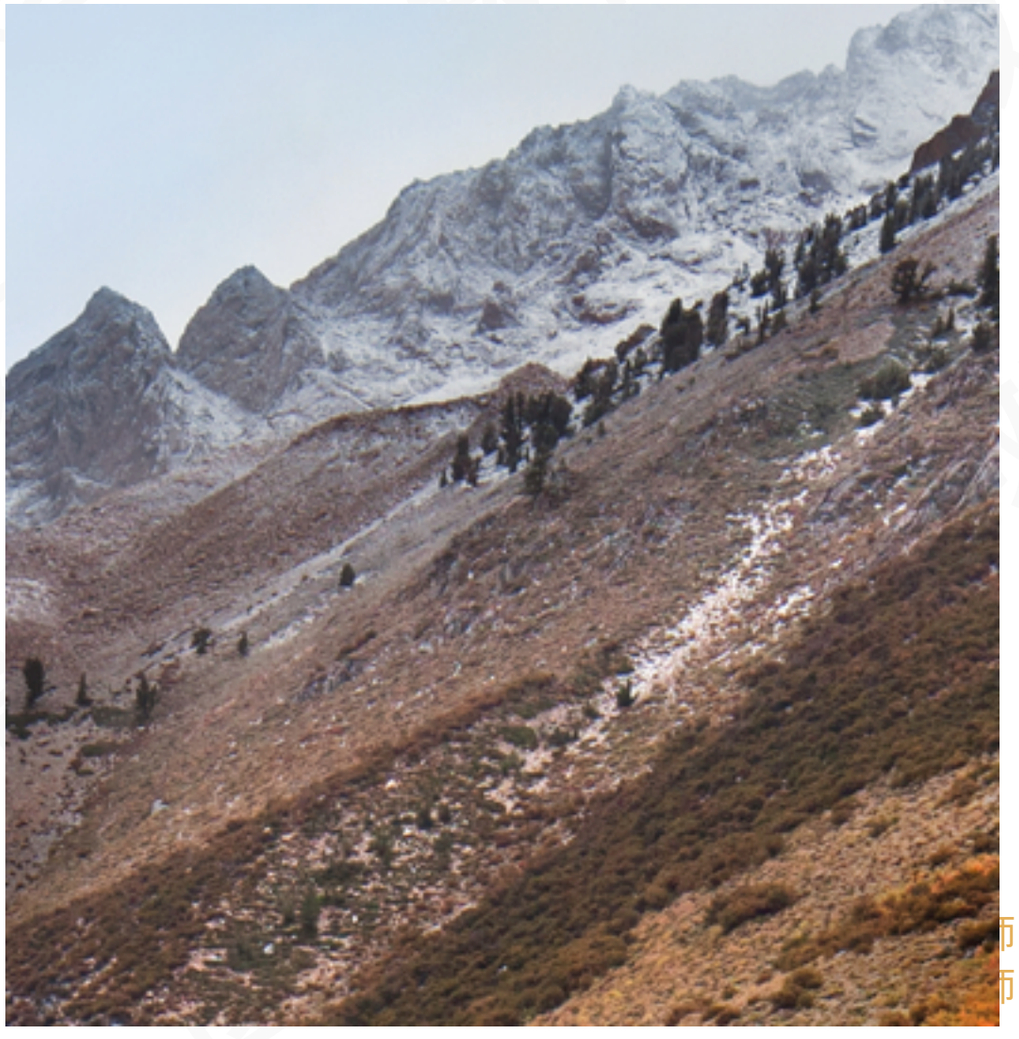
OpenGL 专业名词解析

- 渲染:将图形/图像数据转换成2D空间图像操作叫做渲染(Rendering).
- 顶点数组(VertexArray) 和 顶点缓冲区 （ VertexBuffer)
  - 画图一般是先画好图像的骨架，然后再往骨架里面填充颜色，这对于OpenGL也是一样的。顶点数据就是要画的图像的骨架，和现实中不同的是，OpenGL中的图像都是由图元组成。在OpenGL ES中，有3种类型的图元：点、线、三角形。那这些顶点数据最终是存储在哪里的呢？开发者可以选择设定函数指针，在调用绘制方法的时候，直接由内存传入顶点数据，也就是说这部分数据之前是存储在内存当中的，被称为顶点数组。而性能更高的做法是，提前分配一块显存，将顶点数据预先传入到显存当中。这部分的显存，就被称为顶点缓冲区
  - 顶点指的是我们在绘制一个图形时,它的顶点位置数据.而这个数据可以直接存储在数组中或者将其缓存到GPU内存中

课程研发:CC老师
课程授课:CC老师

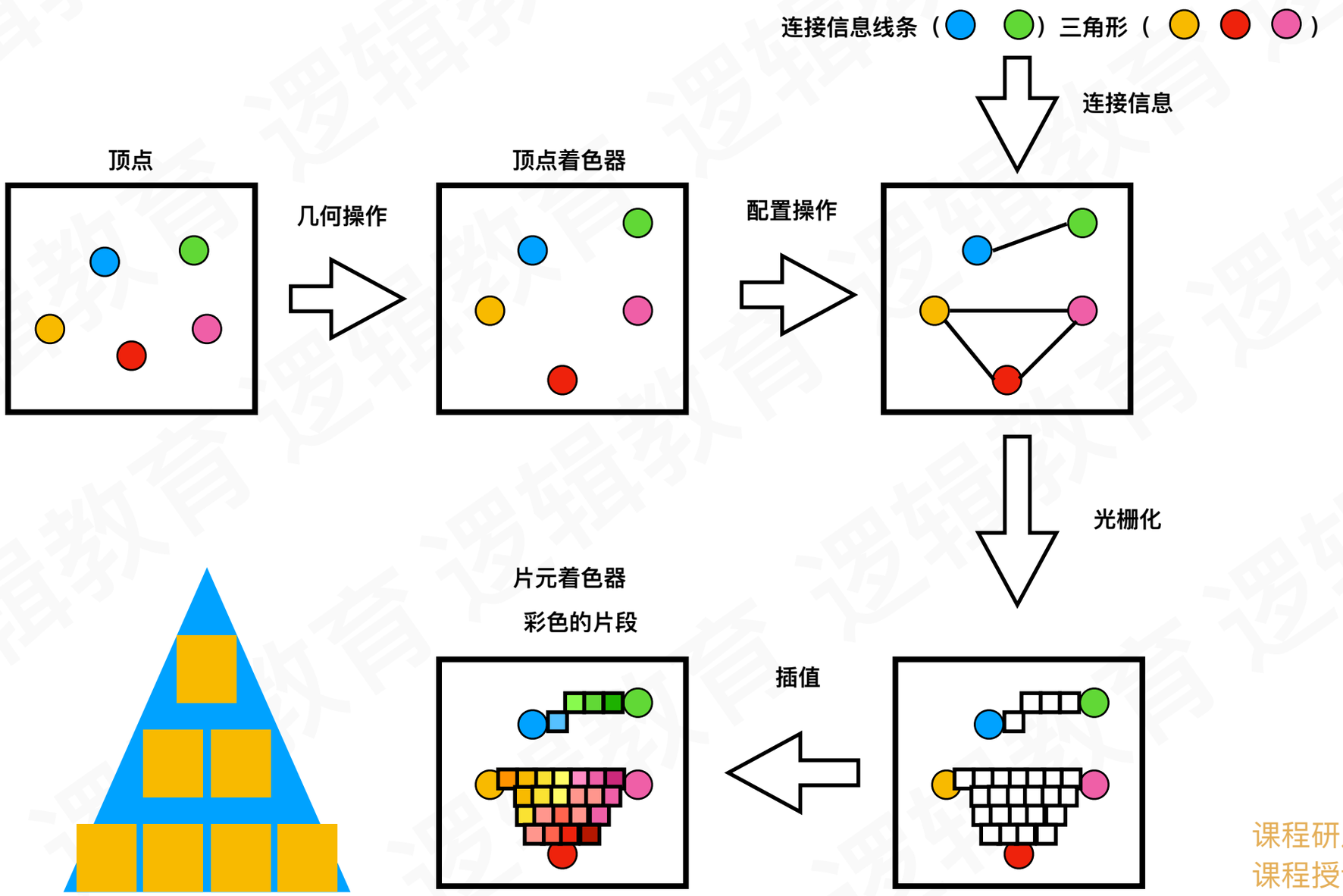


png->位图



着色器渲染过程

在渲染过程中，必须存储2种着色器，分别是顶点着色器、片元着色器。顶点着色器是第一个着色器、片元着色器是最后一个。顶点着色器中处理顶点、片元着色器处理像素点颜色。



课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

- 着色器程序Shader
  - 就全面的将固定渲染管线架构变为了可编程渲染管线。因此，OpenGL在实际调用绘制函数之前，还需要指定一个由shader编译成的着色器程序。常见的着色器主要有顶点着色器（VertexShader），片段着色器（FragmentShader）/像素着色器（PixelShader），几何着色器（GeometryShader），曲面细分着色器（TessellationShader）。片段着色器和像素着色器只是在OpenGL和DX中的不同叫法而已。可惜的是，直到OpenGLES 3.0，依然只支持了顶点着色器和片段着色器这两个最基础的着色器。
  - OpenGL在处理shader时，和其他编译器一样。通过编译、链接等步骤，生成了着色器程序（glProgram），着色器程序同时包含了顶点着色器和片段着色器的运算逻辑。在OpenGL进行绘制的时候，首先由顶点着色器对传入的顶点数据进行运算。再通过图元装配，将顶点转换为图元。然后进行光栅化，将图元这种矢量图形，转换为栅格化数据。最后，将栅格化数据传入片段着色器中进行运算。片段着色器会对栅格化数据中的每一个像素进行运算，并决定像素的颜色

课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

- 管线:在OpenGL 下渲染图形,就会有经历一个一个节点.而这样的操作可以理解管线.大家可以想象成流水线.每个任务类似流水线般执行.任务之间有先后顺序. 管线是一个抽象的概念，之所以称之为管线是因为显卡在处理数据的时候是按照一个固定的顺序来的，而且严格按照这个顺序。就像水从一根管子的一端流到另一端，这个顺序是不能打破的

- 固定管线/存储着色器
  - 在早期的OpenGL 版本,它封装了很多种着色器程序块内置的一段包含了光照、坐标变换、裁剪等等诸多功能的固定shader程序来完成,来帮助开发者来完成图形的渲染. 而开发者只需要传入相应的参数,就能快速完成图形的渲染. 类似于iOS开发会封装很多API,而我们只需要调用,就可以实现功能.不需要关注底层实现原理
  - 但是由于OpenGL 的使用场景非常丰富,固定管线或存储着色器无法完成每一个业务.这时将相关部分开放成可编程

课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

- 顶点着色器 VertexShader
  - 一般用来处理图形每个顶点变换(旋转/平移/投影等)
  - 顶点着色器是OpenGL中用于计算顶点属性的程序。顶点着色器是逐顶点运算的程序，也就是说每个顶点数据都会执行一次顶点着色器，当然这是并行的，并且顶点着色器运算过程中无法访问其他顶点的数据
  - 一般来说典型的需要计算的顶点属性主要包括顶点坐标变换、逐顶点光照运算等等。顶点坐标由自身坐标系转换到归一化坐标系的运算，就是在这里发生的。

课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

- 片元着色器程序 FragmentShader
  - 一般用来处理图形中每个像素点颜色计算和填充
  - 片段着色器是OpenGL中用于计算片段（像素）颜色的程序。片段着色器是逐像素运算的程序，也就是说每个像素都会执行一次片段着色器，当然也是并行的

课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

- GLSL（OpenGL Shading Language）
  - OpenGL着色语言（OpenGL Shading Language）是用来在OpenGL中着色编程的语言，也即开发人员写的短小的自定义程序，他们是在图形卡的GPU（Graphic Processor Unit图形处理单元）上执行的，代替了固定的渲染管线的一部分，使渲染管线中不同层次具有可编程性。比如：视图转换、投影转换等。GLSL（GL Shading Language）的着色器代码分成2个部分：Vertex Shader（顶点着色器）和Fragment（片元着色器）

课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

- 光栅化Rasterization
  - 是把顶点数据转换为片元的过程，具有将图转化为一个个栅格组成的图象的作用，特点是每个元素对应帧缓冲区中的一像素。
  - 光栅化就是把顶点数据转换为片元的过程。片元中的每一个元素对应于帧缓冲区中的一个像素。
  - 光栅化其实是一种将几何图元变为二维图像的过程。该过程包含了两部分的工作。第一部分工作：决定窗口坐标中的哪些整型栅格区域被基本图元占用；第二部分工作：分配一个颜色值和一个深度值到各个区域。光栅化过程产生的是片元
  - 把物体的数学描述以及与物体相关的颜色信息转换为屏幕上用于对应位置的像素及用于填充像素的颜色，这个过程称为光栅化，这是一个将模拟信号转化为离散信号的过程

课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

- 纹理
  - 纹理可以理解为图片.大家在渲染图形时需要在其编码填充图片,为了使得场景更加逼真.而这里使用的图片,就是常说的纹理.但是在OpenGL,我们更加习惯叫纹理,而不是图片.

课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

- 混合（Blending)
  - 在测试阶段之后，如果像素依然没有被剔除，那么像素的颜色将会和帧缓冲区中颜色附着上的颜色进行混合，混合的算法可以通过OpenGL的函数进行指定。但是OpenGL提供的混合算法是有限的，如果需要更加复杂的混合算法，一般可以通过像素着色器进行实现，当然性能会比原生的混合算法差一些.

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

x 变换矩阵(旋转,缩放,平移)

线性代数了解一波~
3D 数学(数学原理)

课程研发:CC老师
课程授课:CC老师

OpenGL 专业名词解析

- 变换矩阵(Transformation)
  - 例如图形想发生平移,缩放,旋转变换.就需要使用变换矩阵.
- 投影矩阵Projection
  - 用于将3D坐标转换为二维屏幕坐标,实际线条也将在二维坐标下进行绘制

课程研发:CC老师
课程授课:CC老师

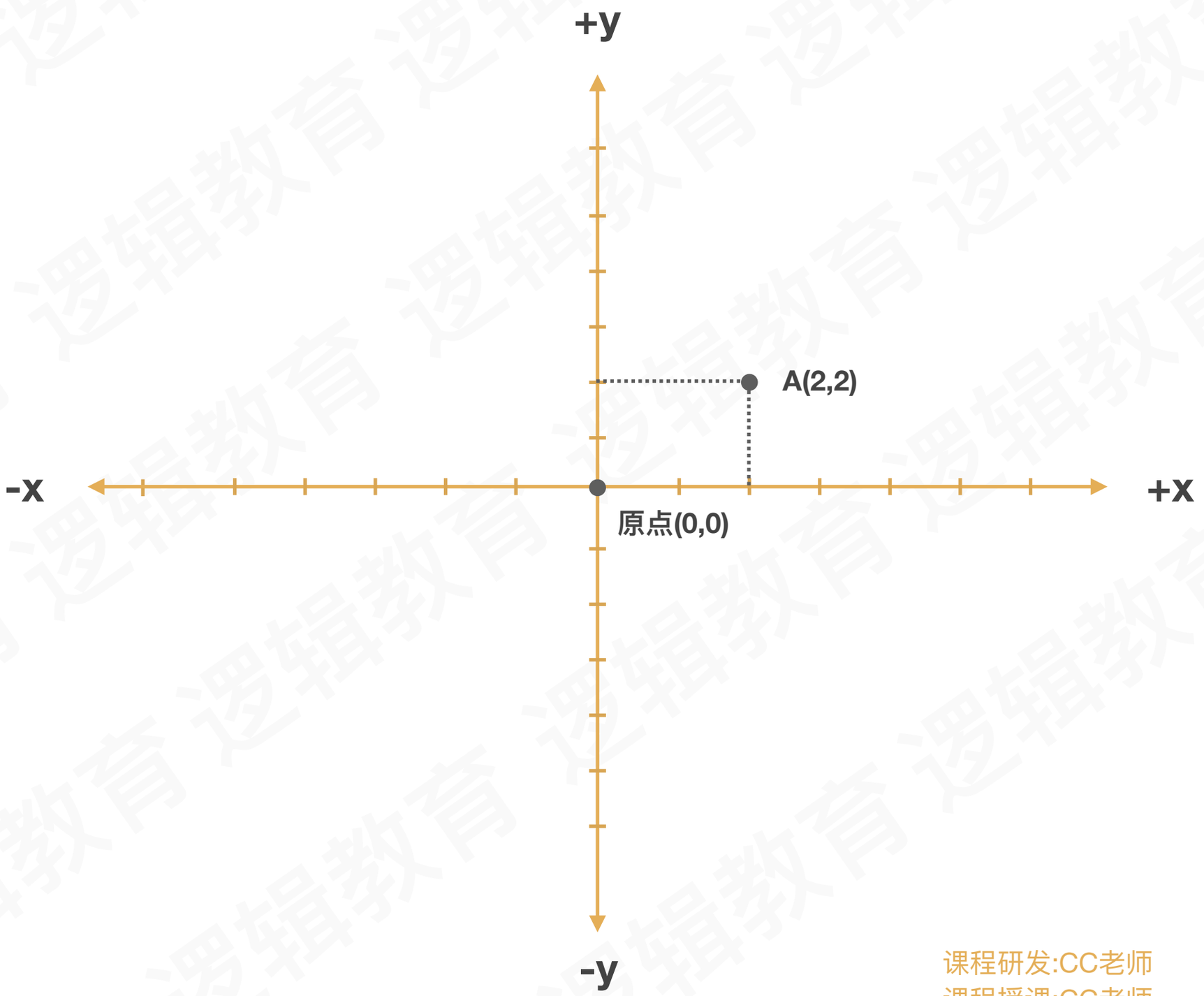
OpenGL 专业名词解析

- 渲染上屏/交换缓冲区(SwapBuffer)
  - 渲染缓冲区一般映射的是系统的资源比如窗口。如果将图像直接渲染到窗口对应的渲染缓冲区，则可以将图像显示到屏幕上。
  - 但是，值得注意的是，如果每个窗口只有一个缓冲区，那么在绘制过程中屏幕进行了刷新，窗口可能显示出不完整的图像
  - 为了解决这个问题，常规的OpenGL程序至少都会有两个缓冲区。显示在屏幕上的称为屏幕缓冲区，没有显示的称为离屏缓冲区。在一个缓冲区渲染完成之后，通过将屏幕缓冲区和离屏缓冲区交换，实现图像在屏幕上的显示。
  - 由于显示器的刷新一般是逐行进行的，因此为了防止交换缓冲区的时候屏幕上下区域的图像分属于两个不同的帧，因此交换一般会等待显示器刷新完成的信号，在显示器两次刷新的间隔中进行交换，这个信号就被称为垂直同步信号，这个技术被称为垂直同步
  - 使用了双缓冲区和垂直同步技术之后，由于总是要等待缓冲区交换之后再进行下一帧的渲染，使得帧率无法完全达到硬件允许的最高水平。为了解决这个问题，引入了三缓冲区技术，在等待垂直同步时，来回交替渲染两个离屏的缓冲区，而垂直同步发生时，屏幕缓冲区和最近渲染完成的离屏缓冲区交换，实现充分利用硬件性能的目的

课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

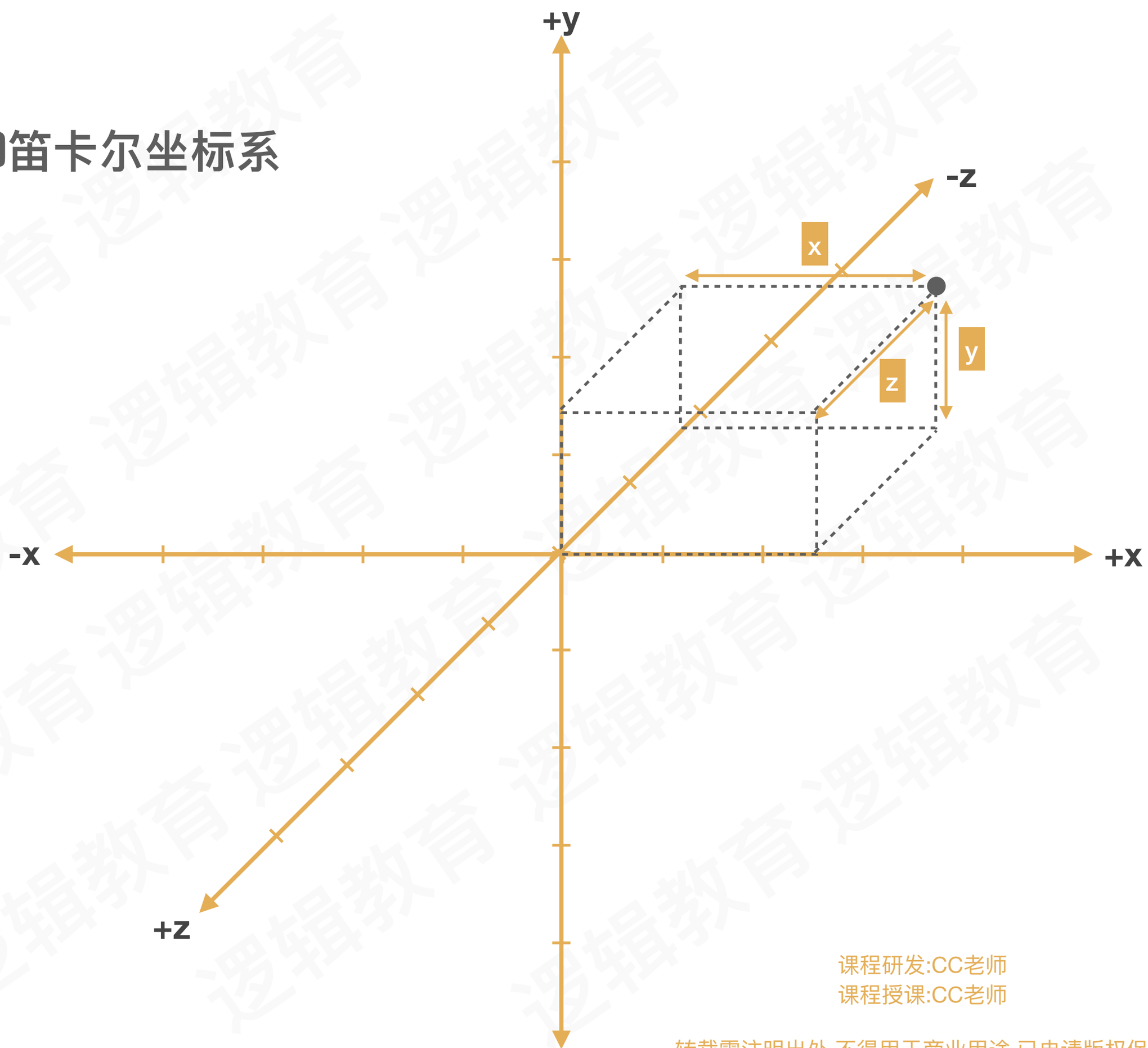
2D笛卡尔坐标系



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

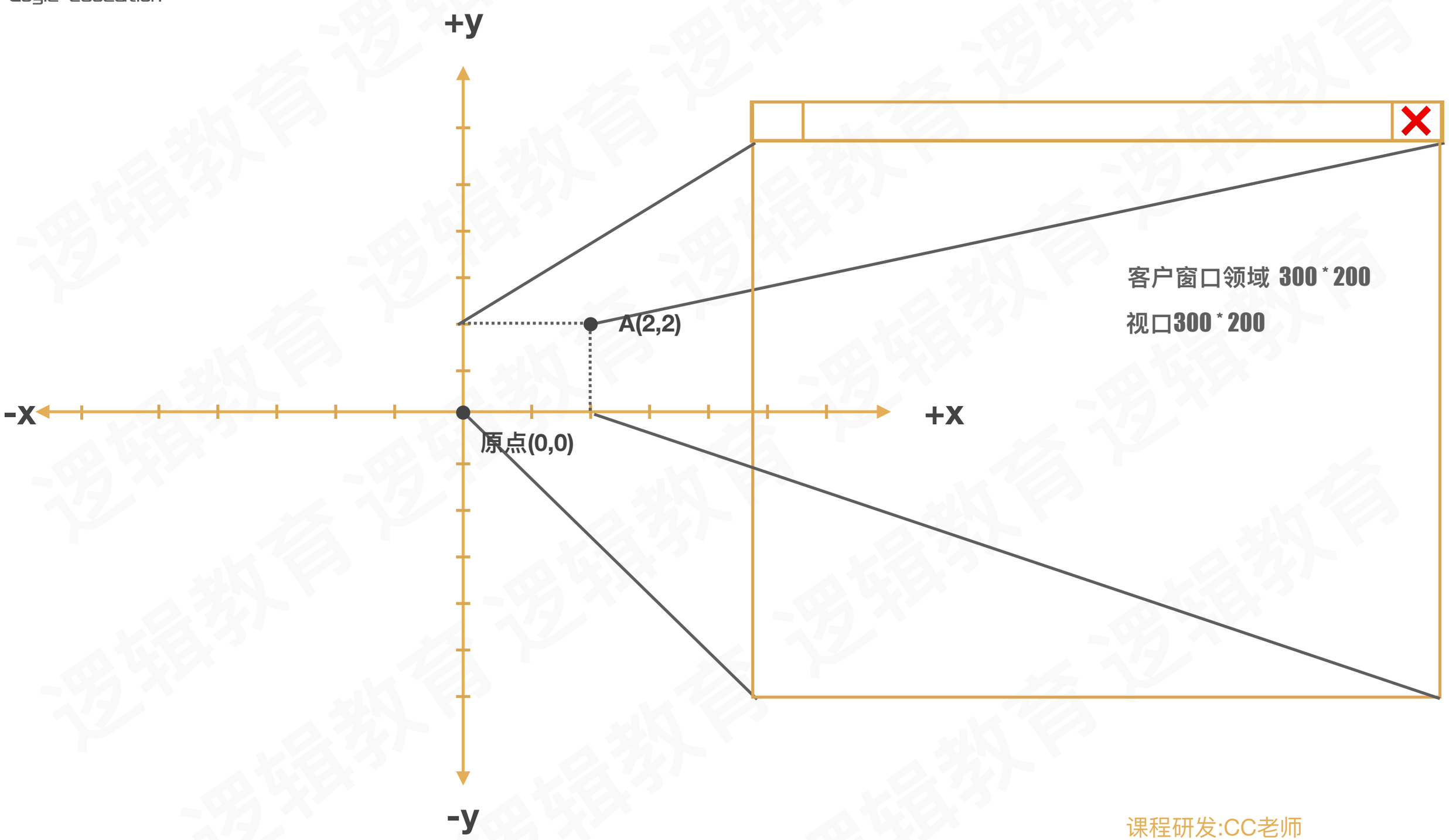
3D笛卡尔坐标系



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

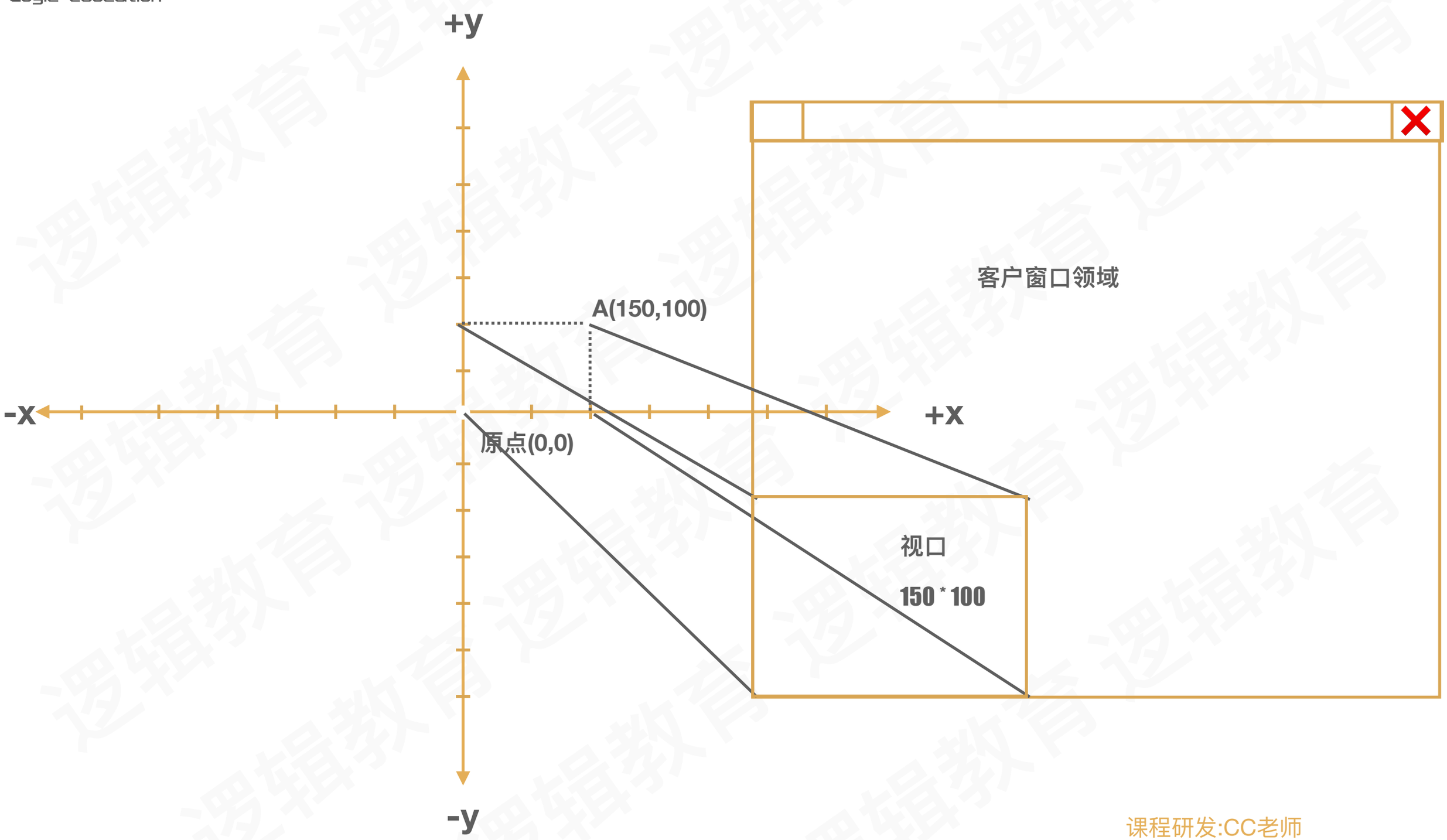
视口



课程研发:CC老师
课程授课:CC老师

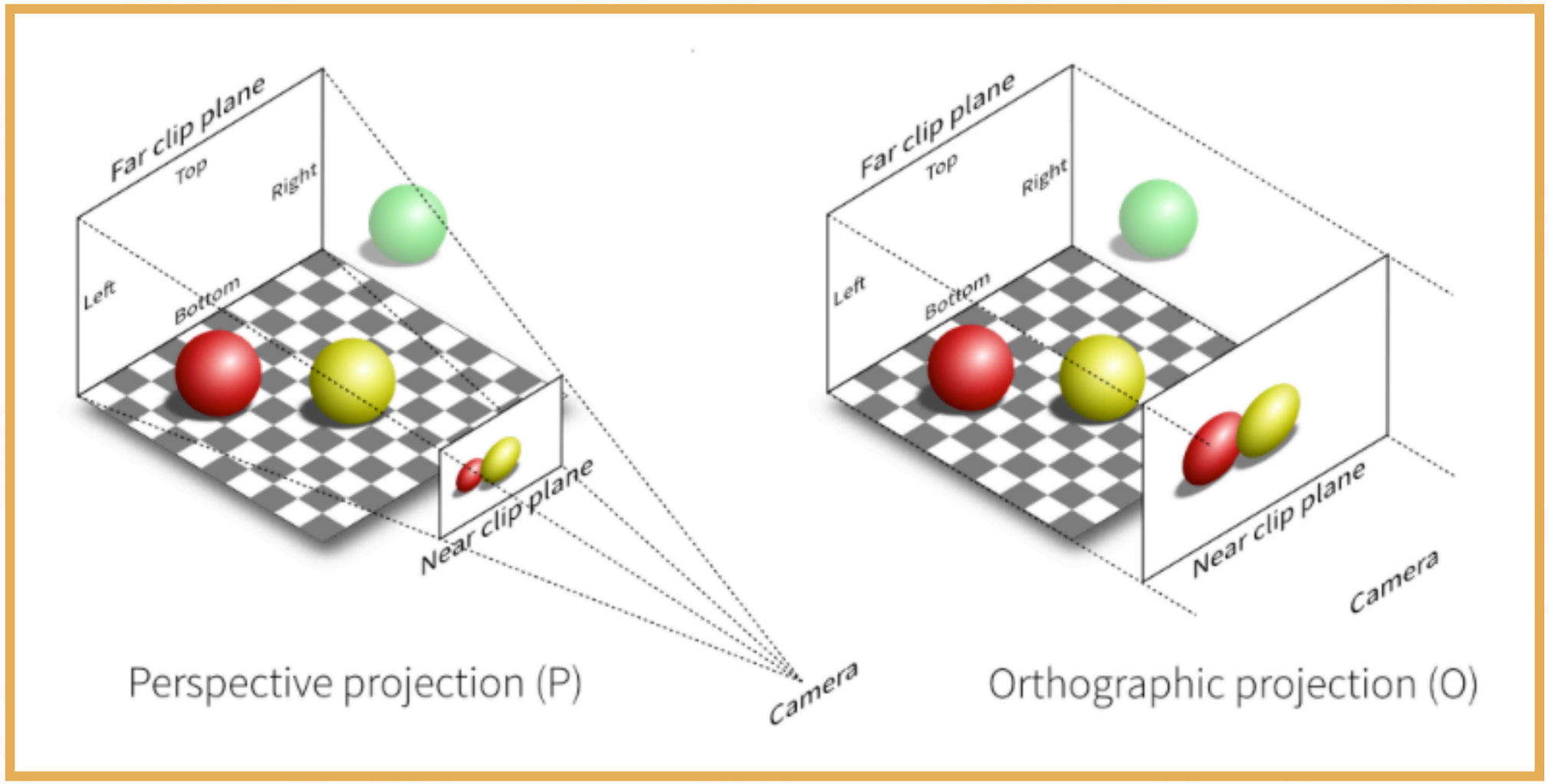
逻辑教育
Logic education

视口



课程研发:CC老师
课程授课:CC老师

OpenGL 投影方式

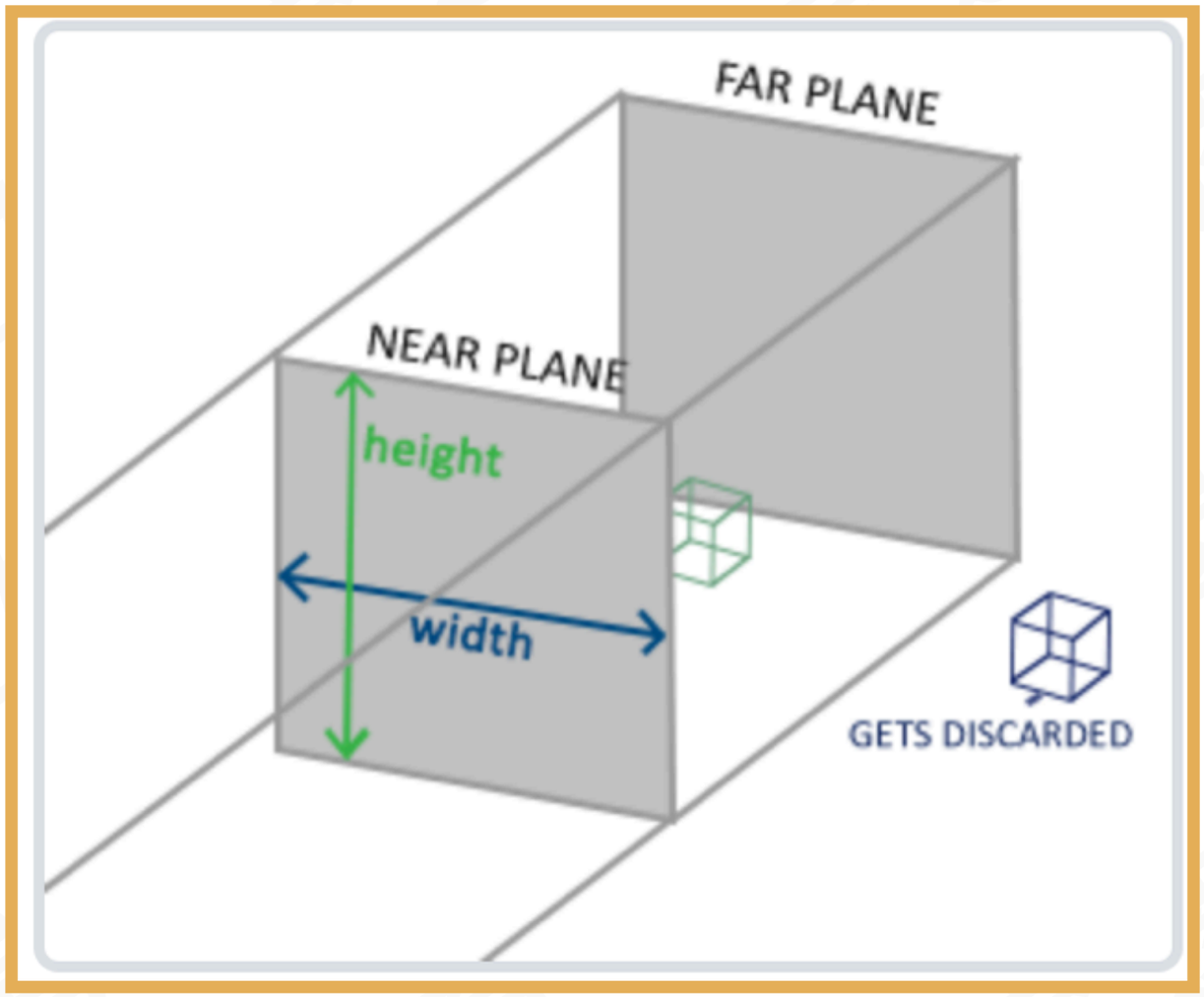


透视投影

正投影

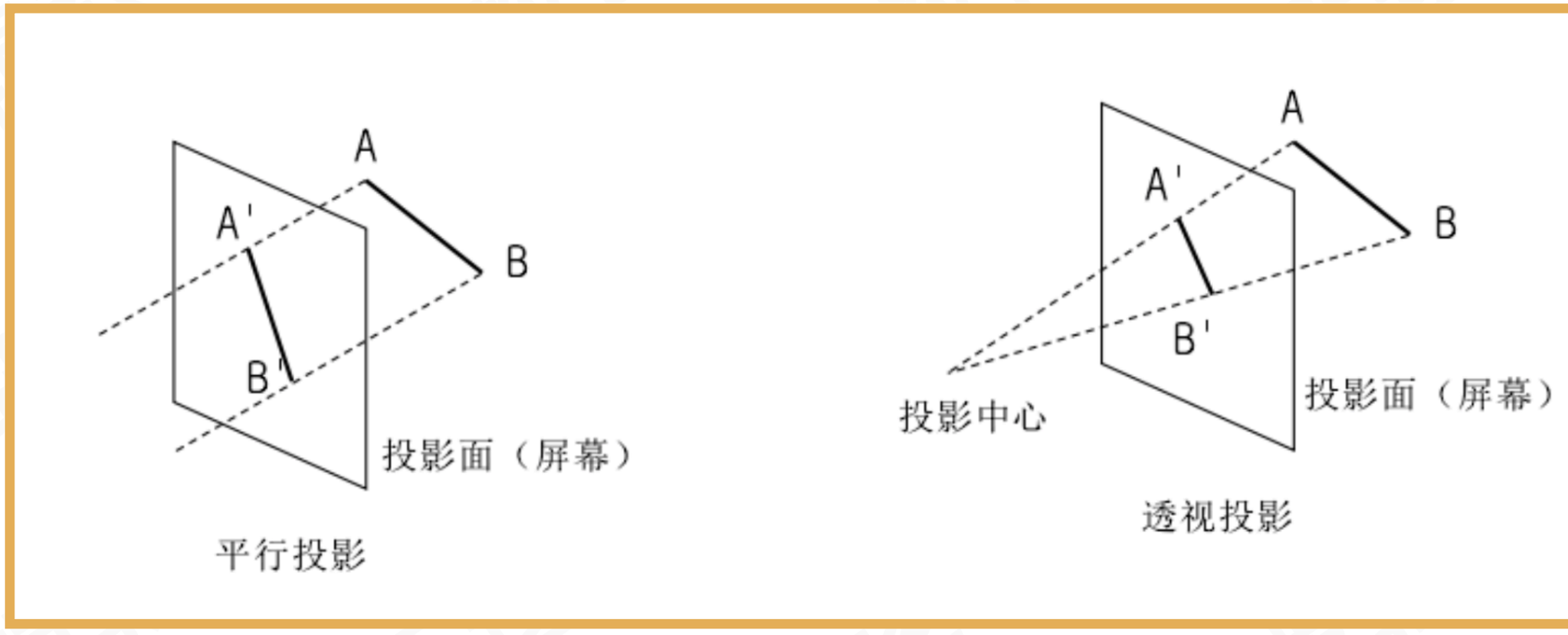
课程研发:CC老师
课程授课:CC老师

OpenGL 正投影



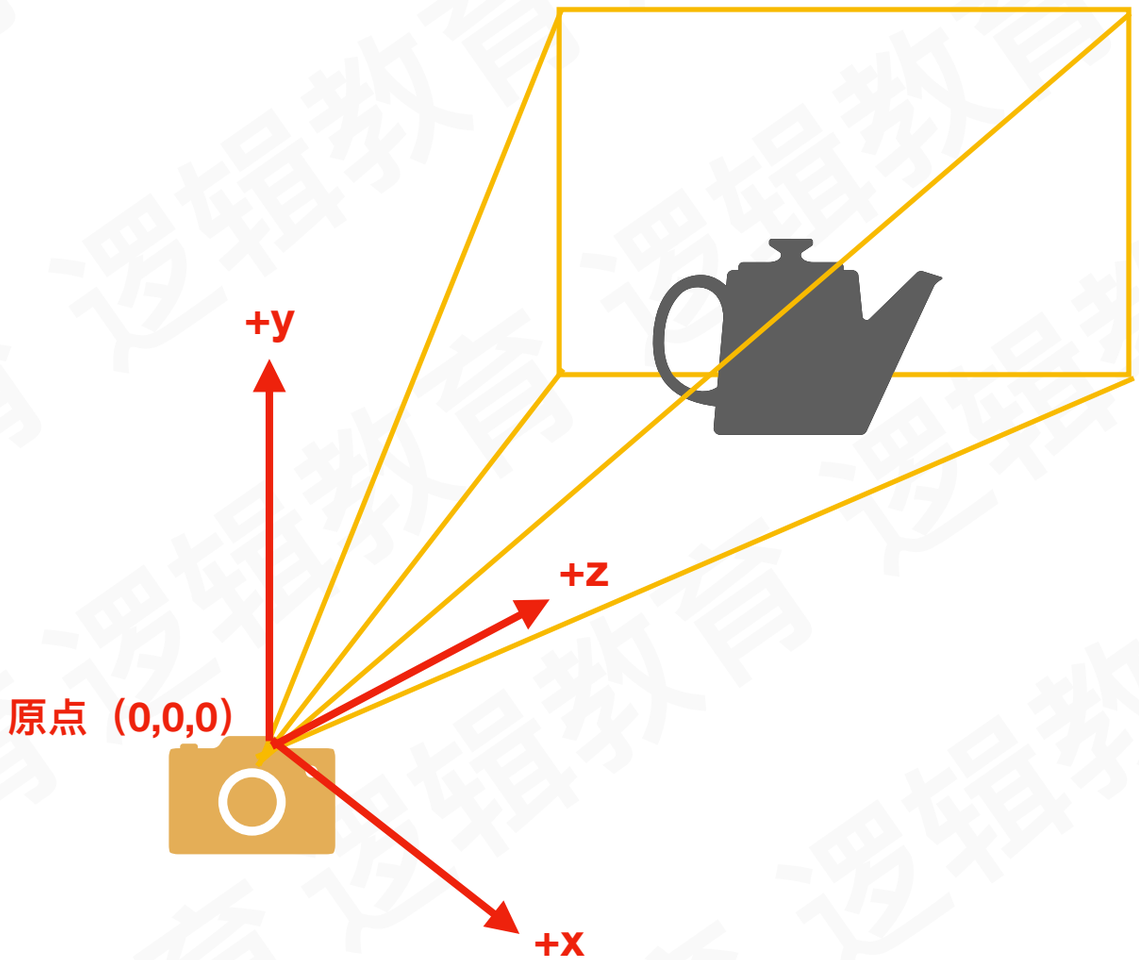
课程研发:CC老师
课程授课:CC老师

OpenGL 投影方式



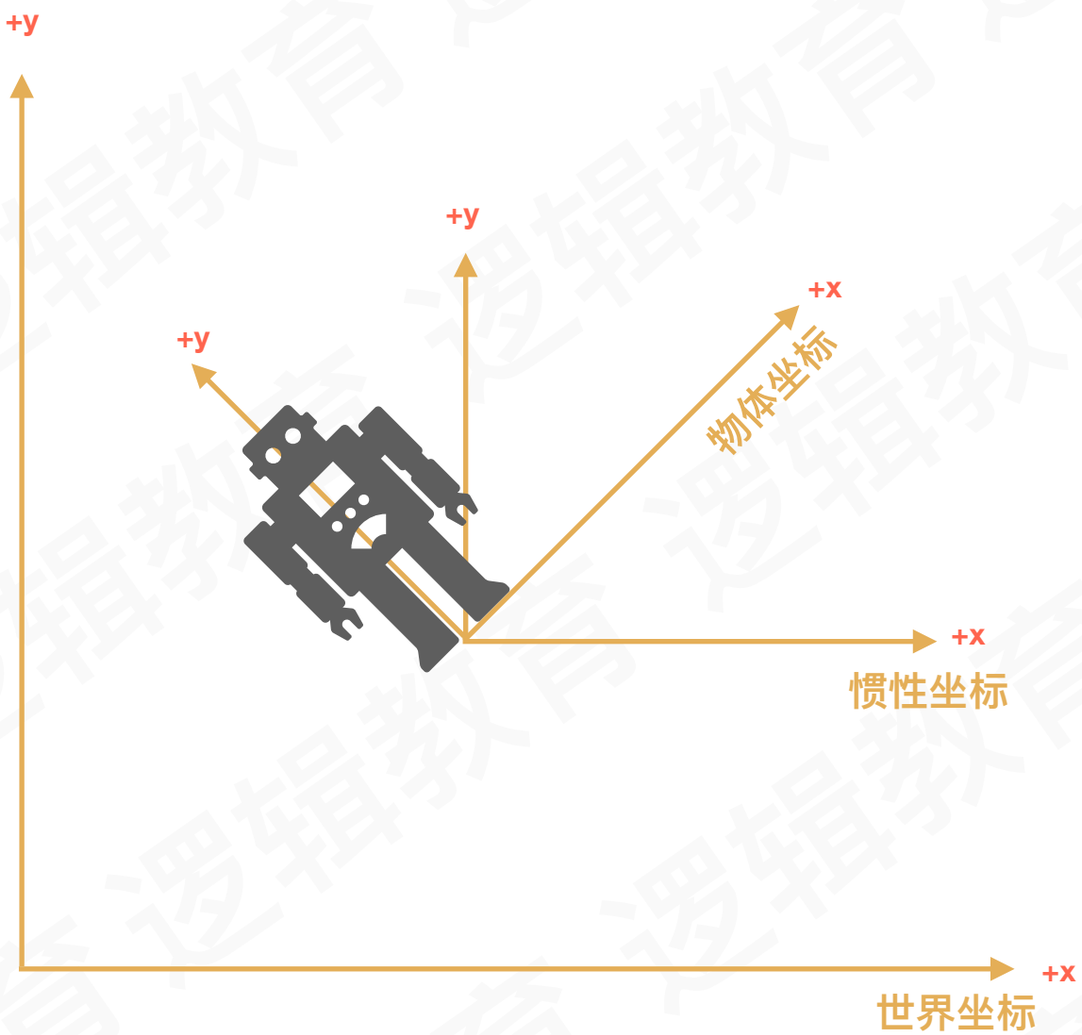
课程研发:CC老师
课程授课:CC老师

OpenGL 摄像机坐标系



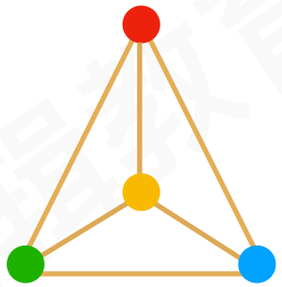
课程研发:CC老师
课程授课:CC老师

OpenGL 坐标系(世界坐标系,惯性坐标系,物体坐标系)



课程研发:CC老师
课程授课:CC老师

OpenGL 坐标变换全局图



Object Space

物体坐标/对象坐标

Modeling transformation

模型变换

World Space

世界坐标

Viewing transformation

视变换

Eye Space

观察者坐标/摄像机坐标

Projection transformation

投影变换

Clip Space

裁剪坐标

Perspective divide

透视除法

NDC Space

规范化设备坐标

ViewPort mapping

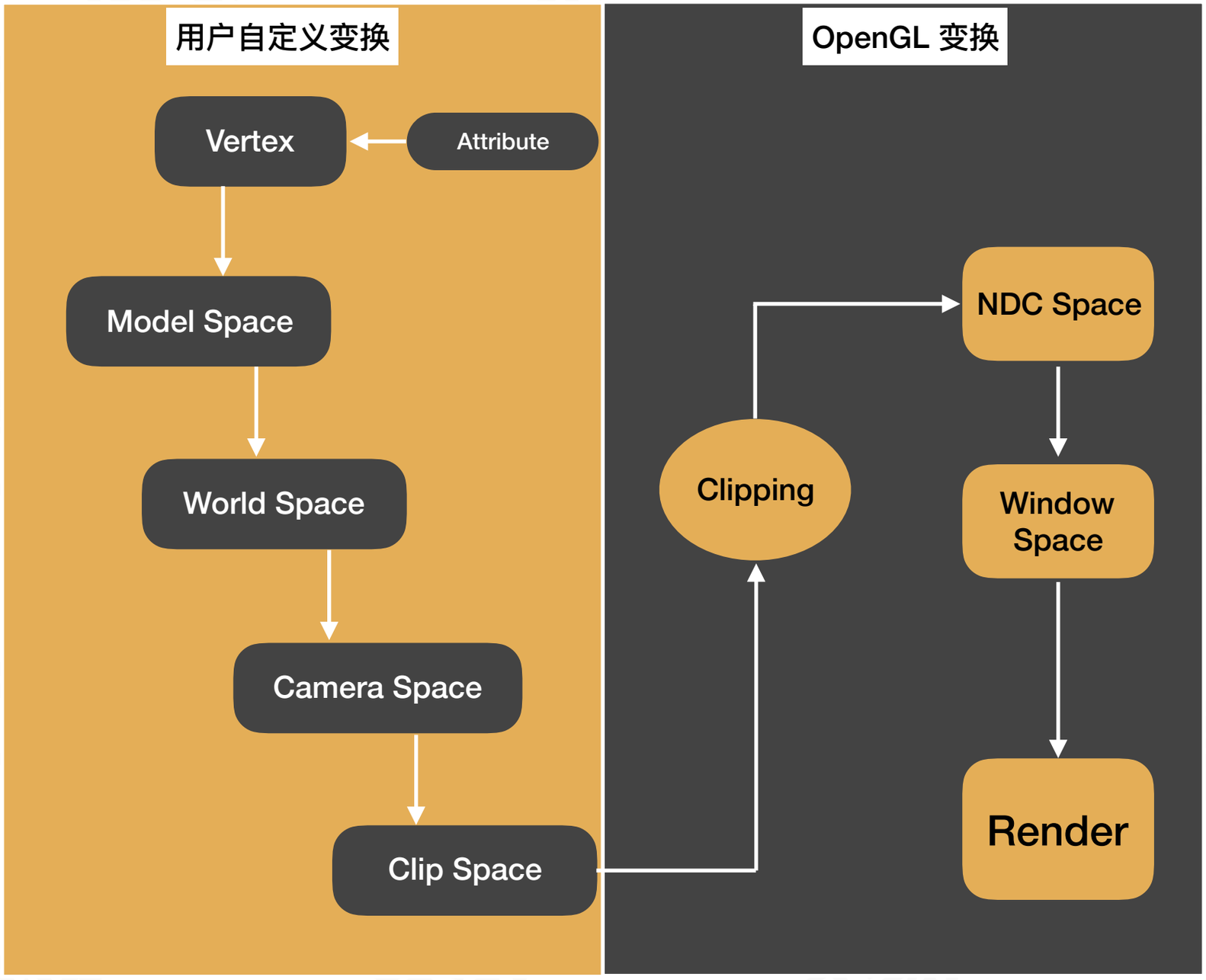
视口变换

Screen space

屏幕坐标

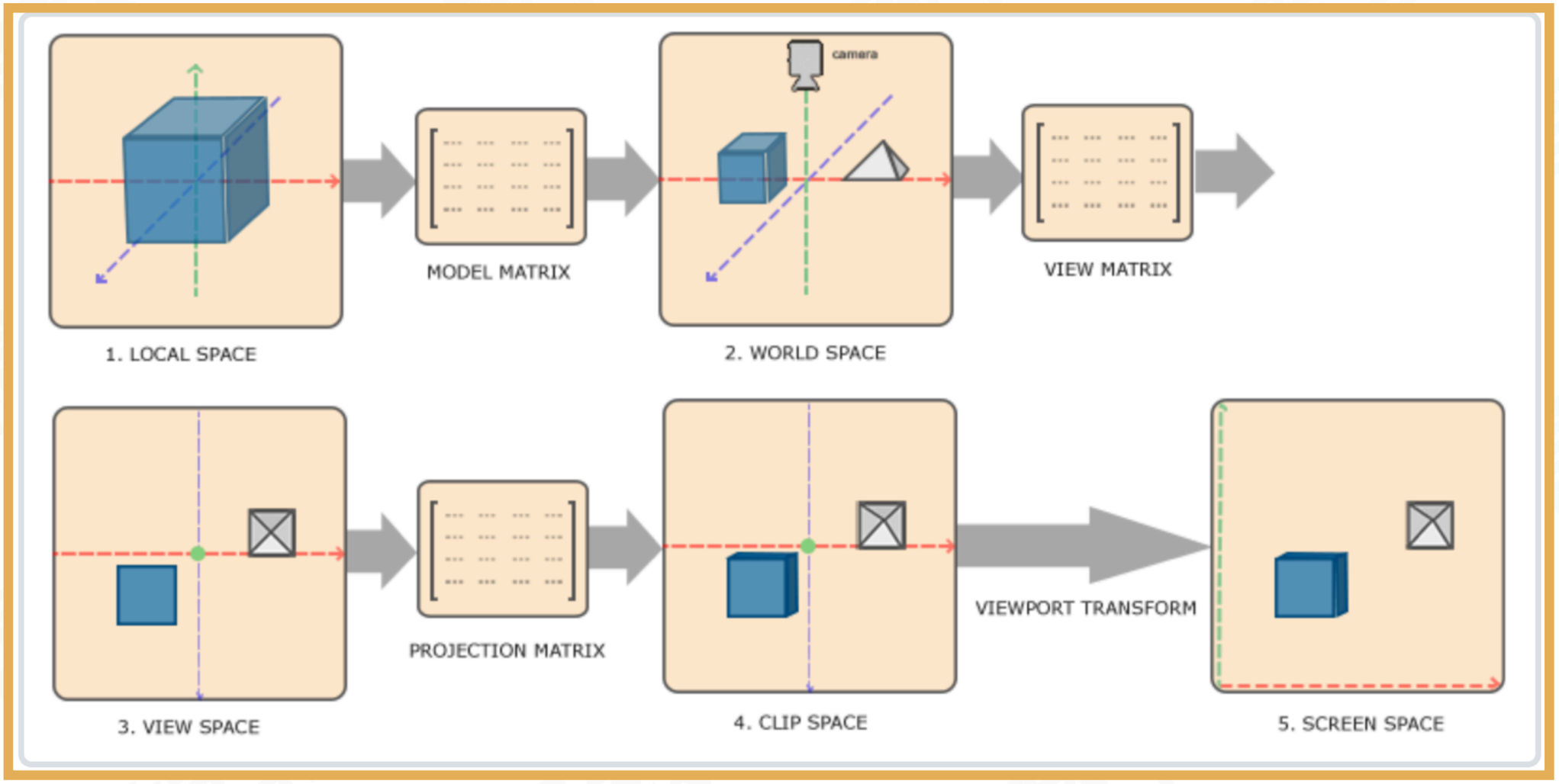
课程研发:CC老师
课程授课:CC老师

User-Defined Transformations & OpenGL Transformations



课程研发:CC老师
课程授课:CC老师

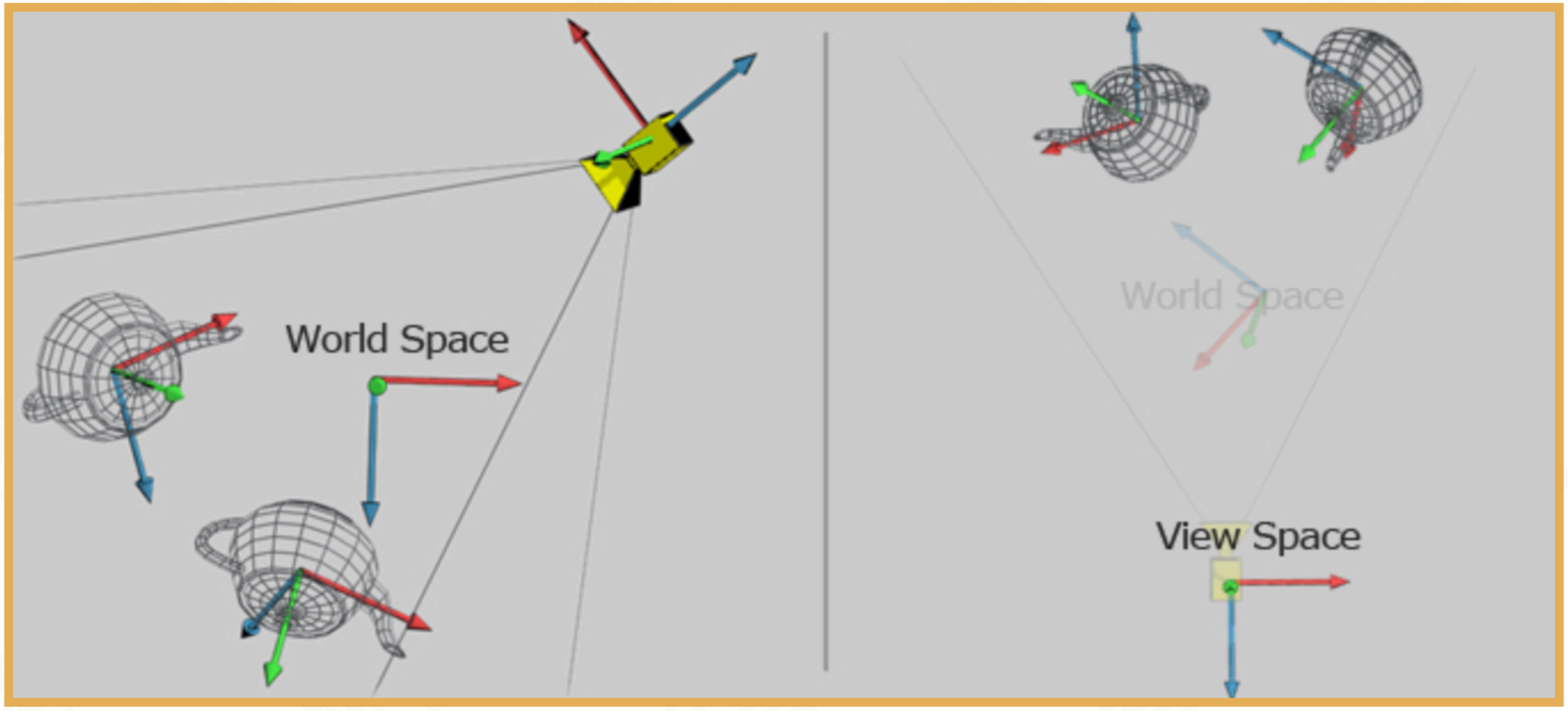
坐标转换计算



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

视变换

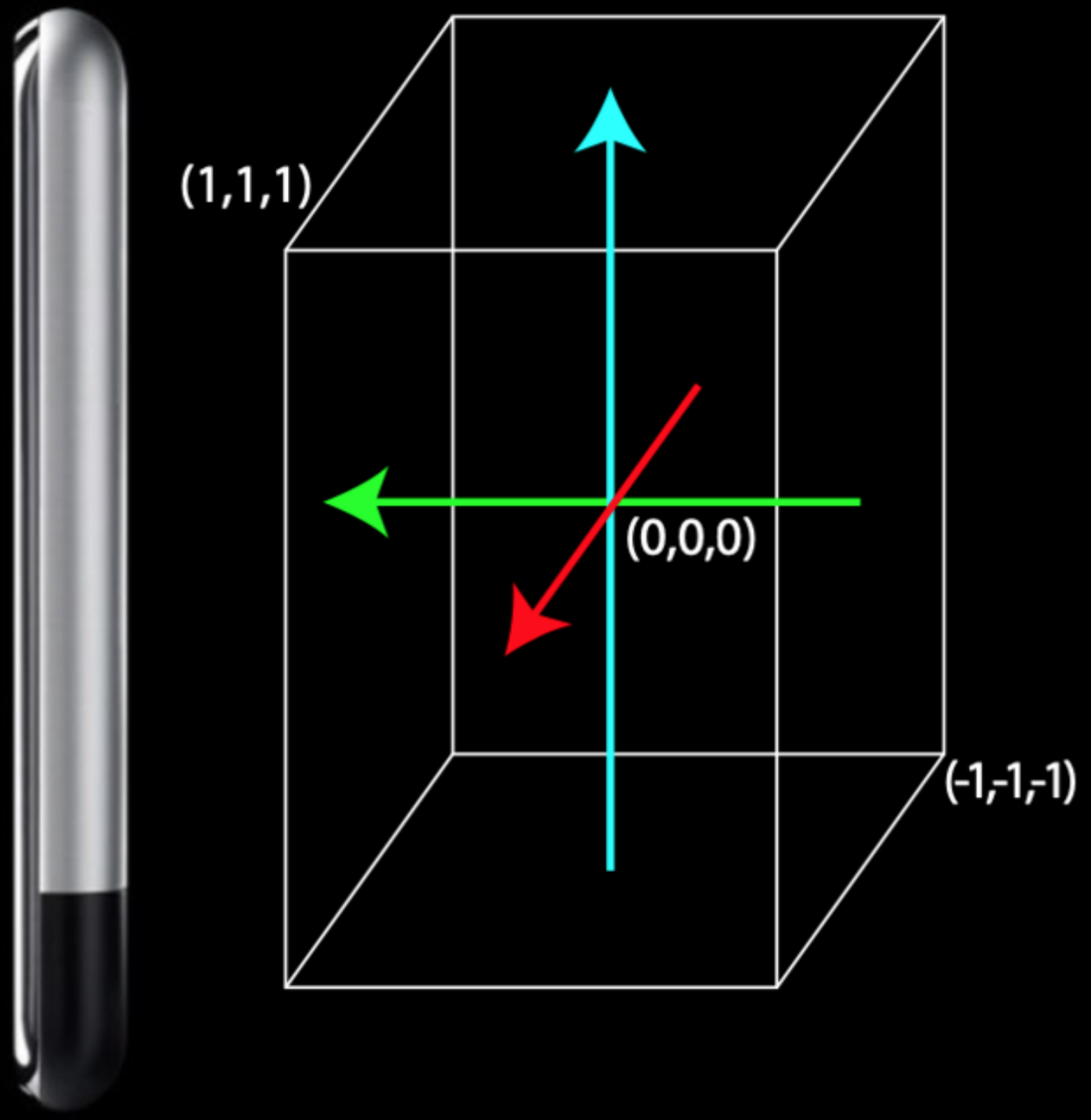


课程研发:CC老师
课程授课:CC老师

Window Coordinates

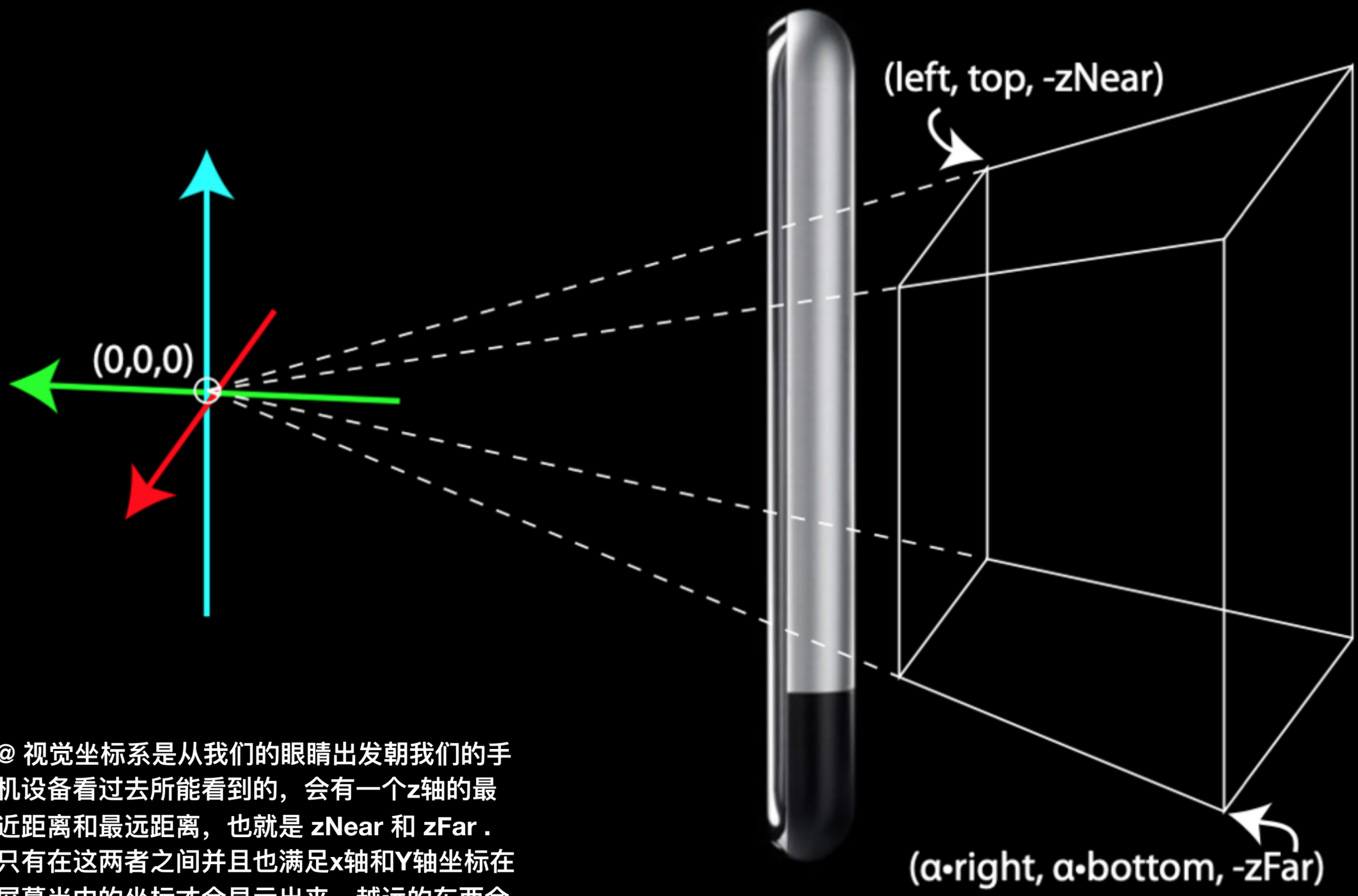
@ 视窗坐标也就是我们手机窗口对应的坐标系统，以左上角为原点，右下角对应我们手机的最大像素值的集合，如下图是一个像素为320*480的手机，那他右下角的坐标就是(320,480)

Normalized Device Coordinates



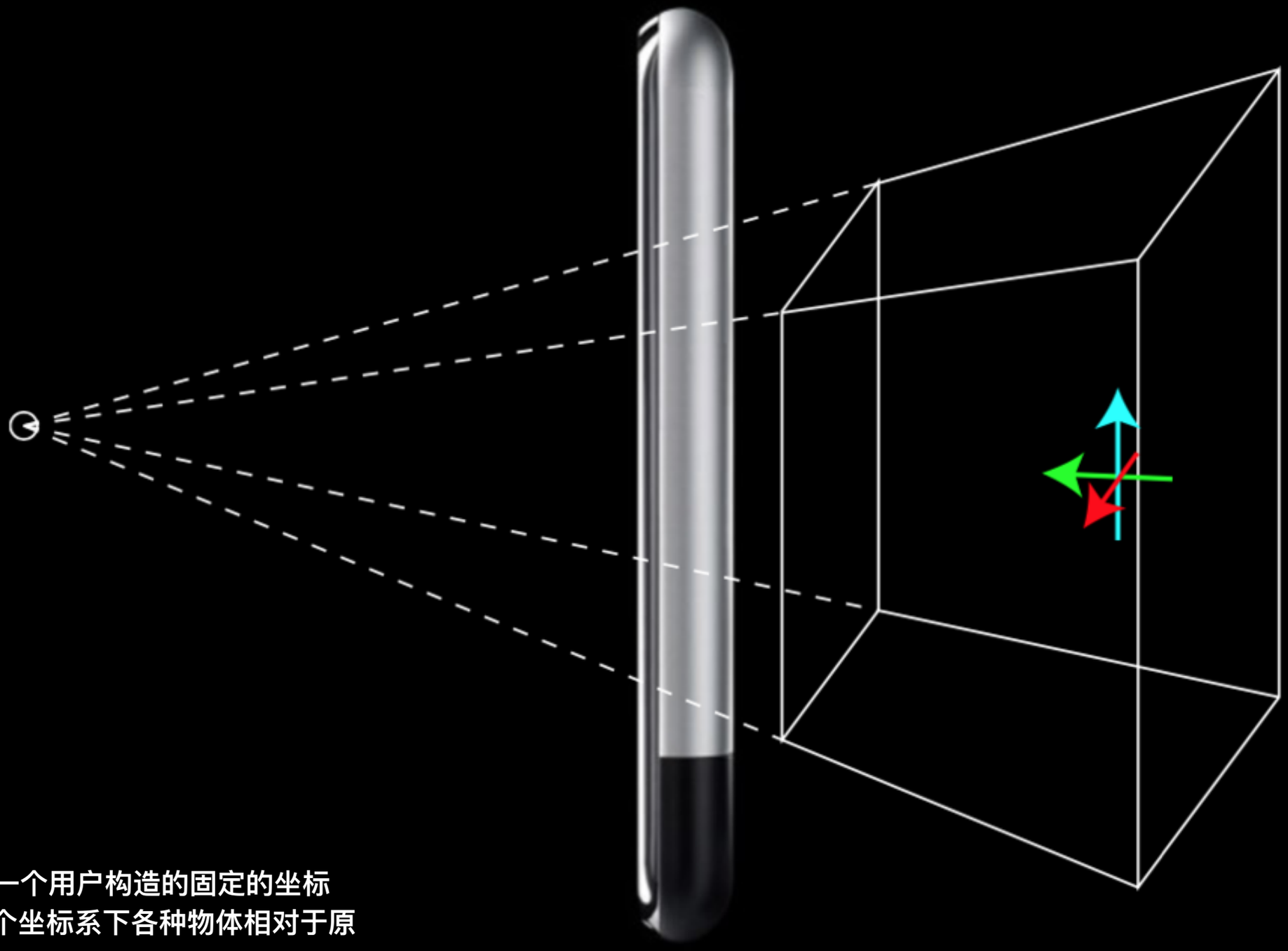
@ 规格化设备坐标 是以屏幕中心为原点, X轴朝由, Y轴朝上, 所以左下角的坐标为 (-1,-1), 右上角坐标为 (1,1). 在规格化设备坐标系需要考虑Z轴. 需要将一个平面的思维转化为一个立体. 原点是 (0,0,0) 也就是立方体的中心.

Eye Coordinates



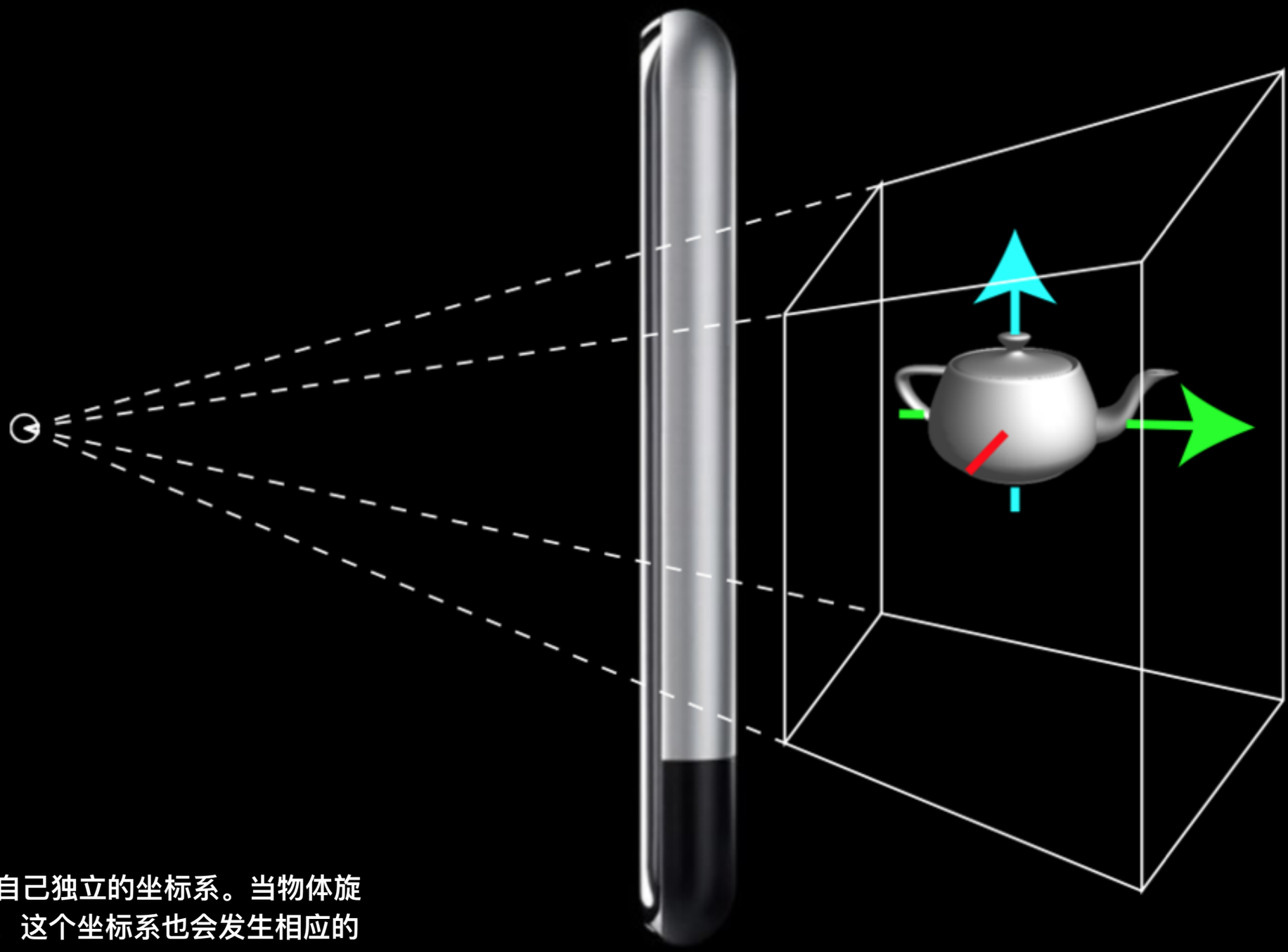
@ 视觉坐标系是从我们的眼睛出发朝我们的手机设备看过去所能看到的，会有一个z轴的最近距离和最远距离，也就是 zNear 和 zFar . 只有在这两者之间并且也满足x轴和Y轴坐标在屏幕当中的坐标才会显示出来，越远的东西会显示得越小，产生透视的效果。

World Coordinates



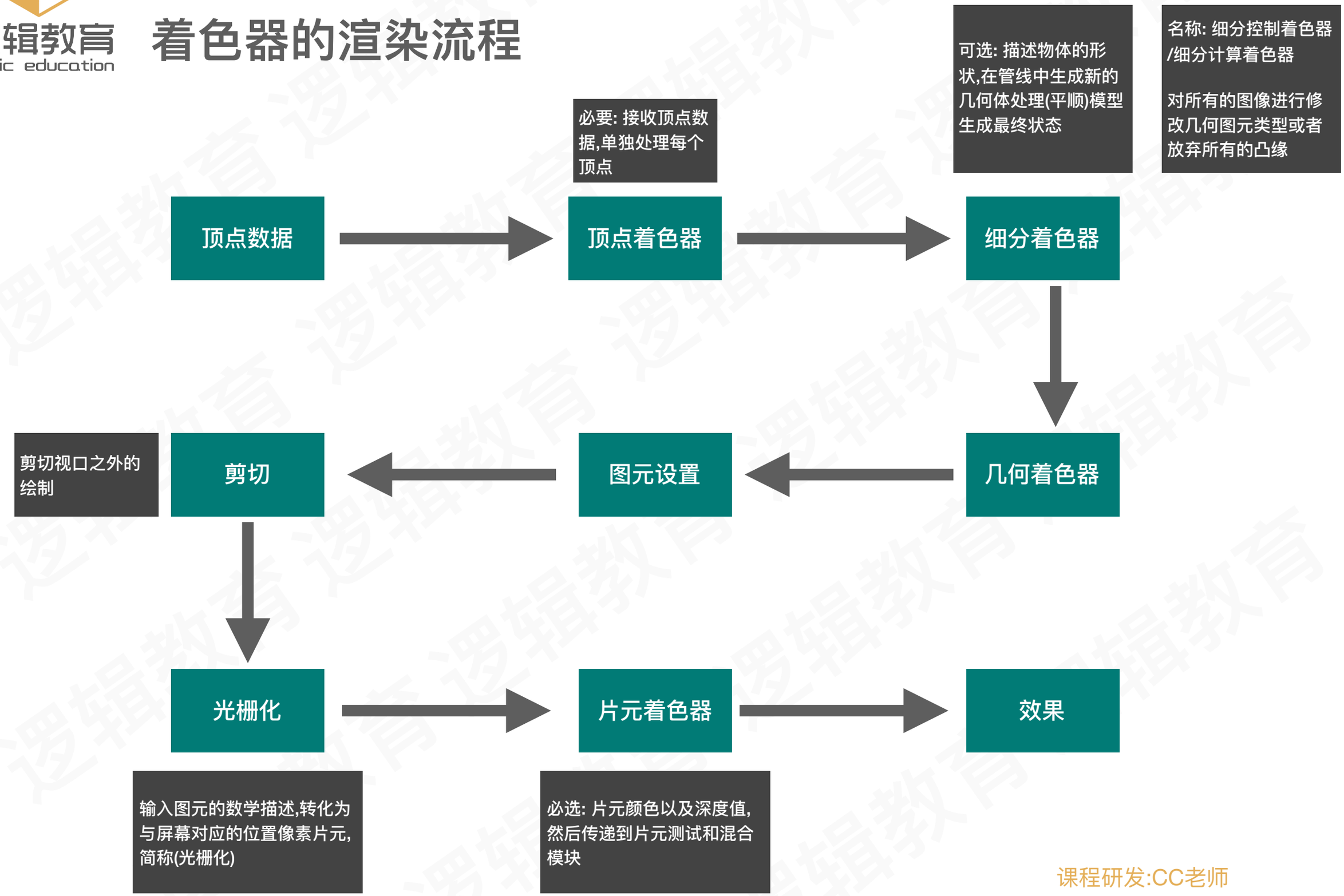
@ 世界坐标就是一个用户构造的固定的坐标系，方便描述这个坐标系下各种物体相对于原点的位置。

Object Coordinates



@ 每个物体都有自己独立的坐标系。当物体旋转和移动的时候，这个坐标系也会发生相应的变化。

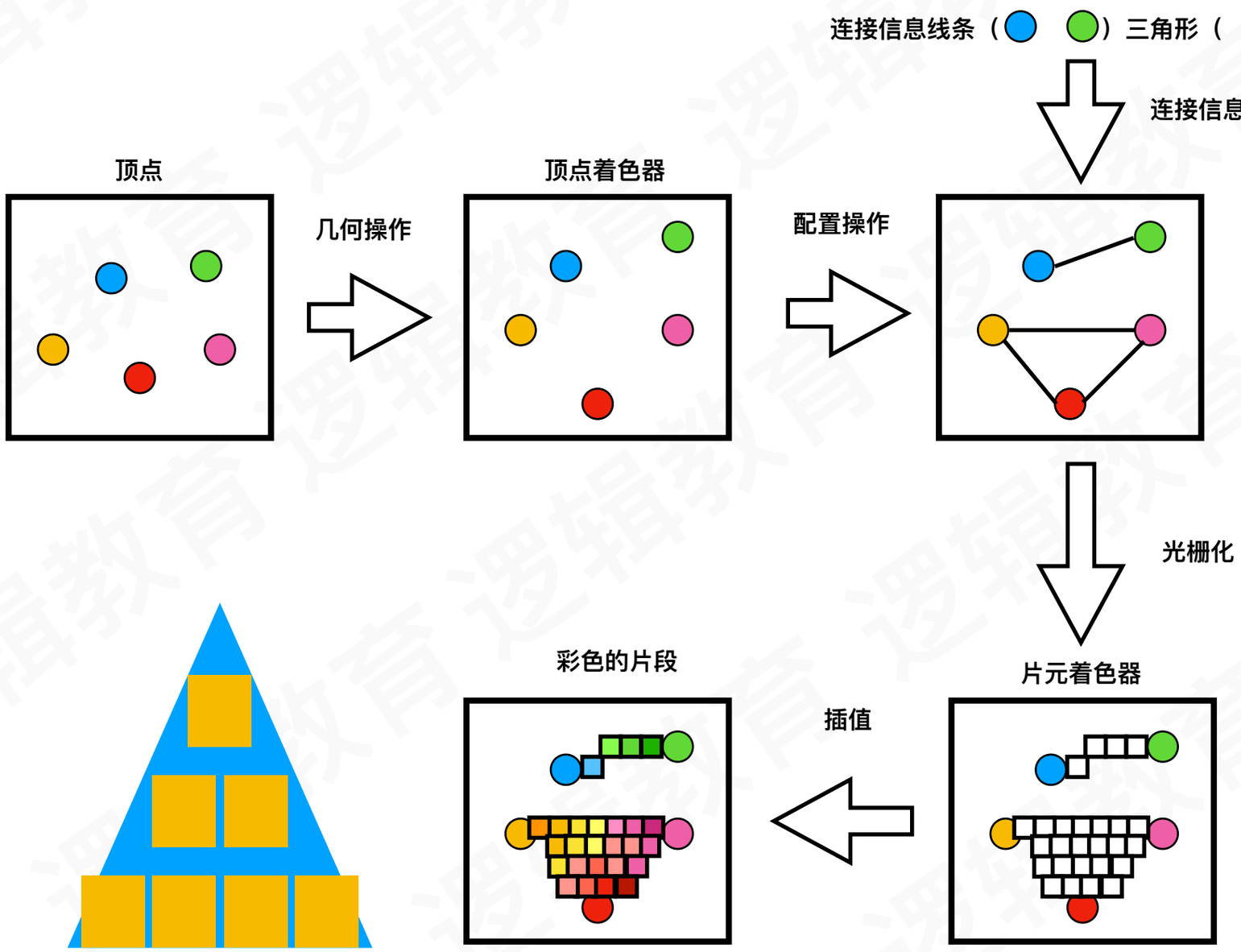
着色器的渲染流程



课程研发:CC老师
课程授课:CC老师

着色器渲染过程

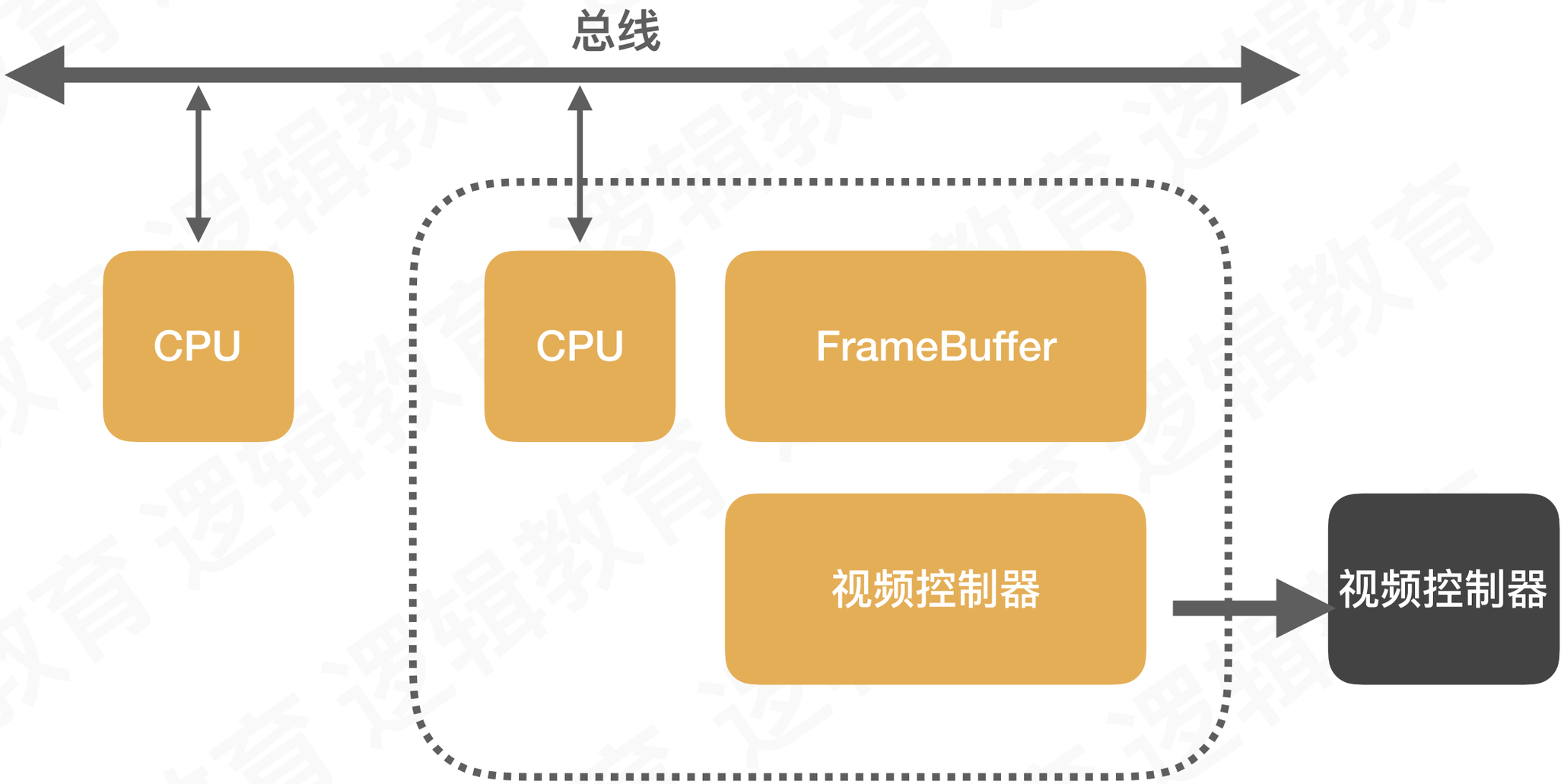
在渲染过程中，必须存储2种着色器，分别是顶点着色器、片元着色器。顶点着色器是第一个着色器、片元着色器是最后一个。顶点着色器中处理顶点、片元着色器处理像素点颜色。



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

图片渲染流程



课程研发:CC老师
课程授课:CC老师

逻辑教育
Logic education

YYImage解压图片

```
CGImageRef YYCGImageCreateDecodedCopy(CGImageRef imageRef, BOOL decodeForDisplay) {
    ...

    if (decodeForDisplay) { // decode with redraw (may lose some precision)
        CGImageAlphaInfo alphaInfo = CGImageGetAlphaInfo(imageRef) & kCGBitmapAlphaInfoMask;

        BOOL hasAlpha = NO;
        if (alphaInfo == kCGImageAlphaPremultipliedLast ||
            alphaInfo == kCGImageAlphaPremultipliedFirst ||
            alphaInfo == kCGImageAlphaLast ||
            alphaInfo == kCGImageAlphaFirst) {
            hasAlpha = YES;
        }

        // BGRA8888 (premultiplied) or BGRX8888
        // same as UIGraphicsBeginImageContext() and -[UIView drawRect:]
        CGBitmapInfo bitmapInfo = kCGBitmapByteOrder32Host;
        bitmapInfo |= hasAlpha ? kCGImageAlphaPremultipliedFirst : kCGImageAlphaNoneSkipFirst;

        CGContextRef context = CGBitmapContextCreate(NULL, width, height, 8, 0,
YYCGColorSpaceGetDeviceRGB(), bitmapInfo);
        if (!context) return NULL;

        CGContextDrawImage(context, CGRectMake(0, 0, width, height), imageRef); // decode
        CGImageRef newImage = CGBitmapContextCreateImage(context);
        CFRelease(context);

        return newImage;
    } else {
        ...
    }
```

逻辑教育
Logic education

使用固定存储着色器渲染三角形/正方形,并能通过键盘移动

案例1: 001—三角形渲染

课程研发:CC老师
课程授课:CC老师

使用固定存储着色器渲染三角形/正方形,并能通过键盘移动

案例2: 002—图形移动

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

逻辑教育
Logic education

Hello Coder

学习,是一件开心的事

知识,是一个值得分享的东西

献给,我可爱的开发者们.

课程研发:CC老师
课程授课:CC老师