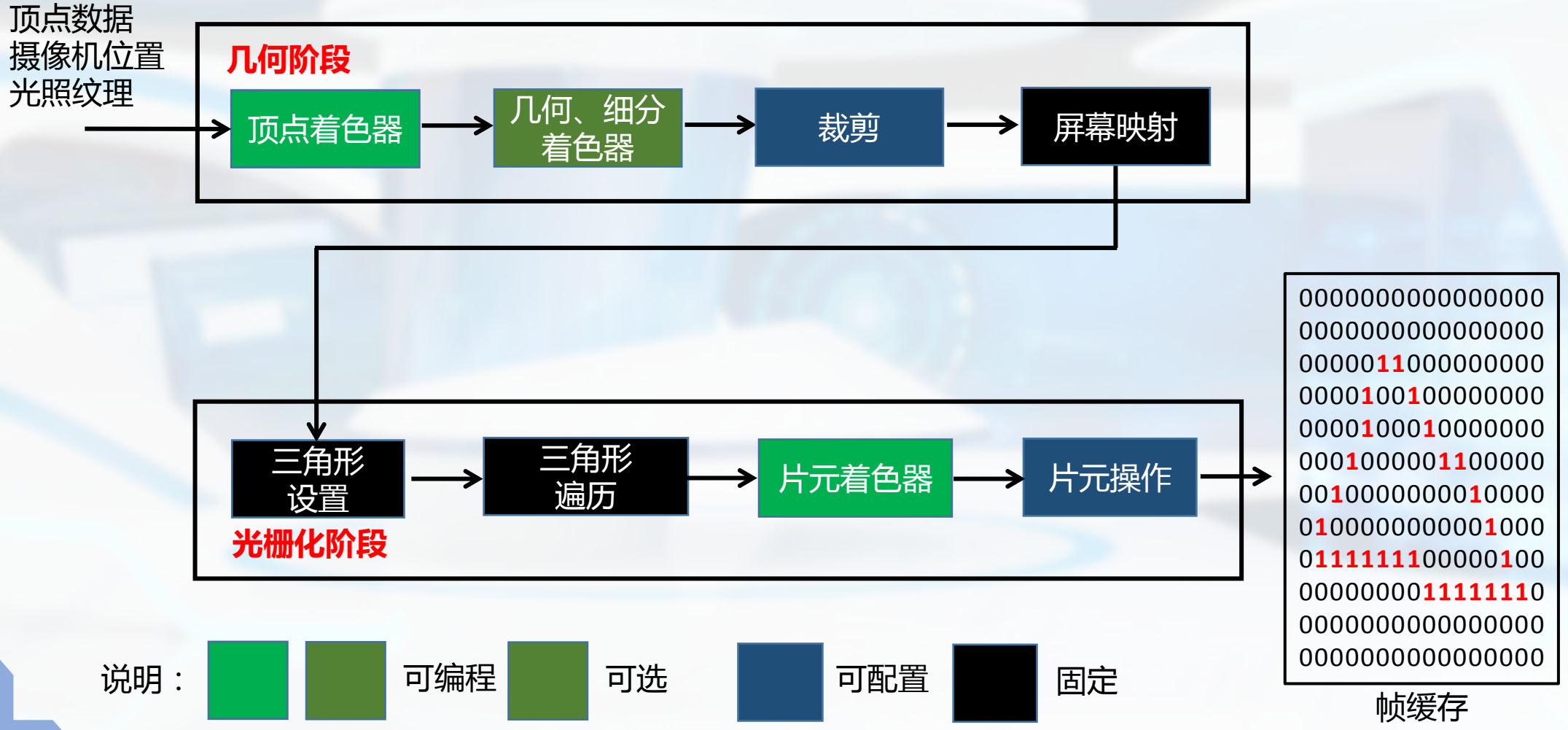
着色器编程

华中科技大学软件学院　万琳

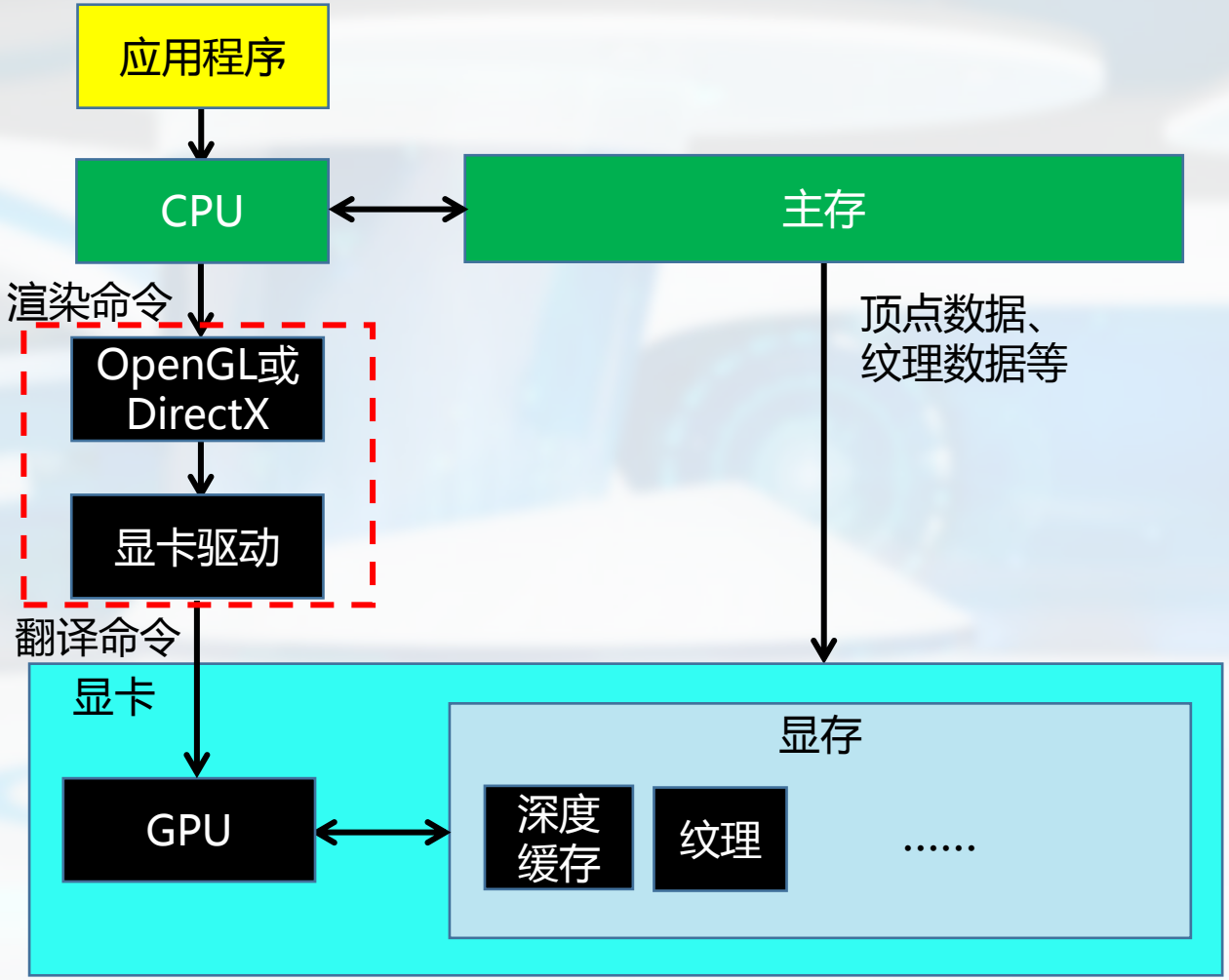
提纲

1. 着色器语言
2. GLSL
3. EBO、VBO和VAO
4. 例程

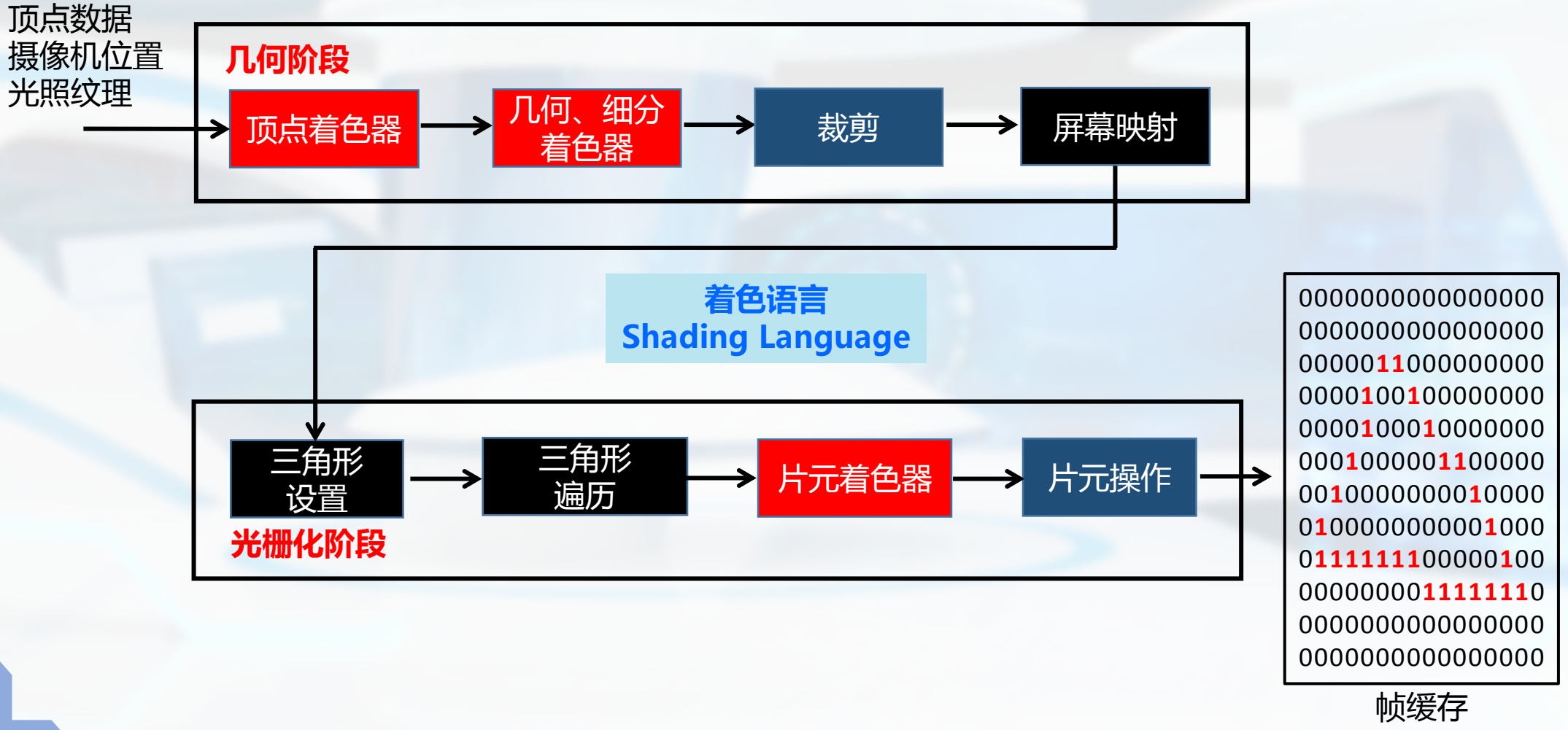
1 着色器语言


顶点数据
摄像机位置
光照纹理
几何阶段
顶点着色器
几何、细分着色器
裁剪
屏幕映射
三角形设置
三角形遍历
片元着色器
片元操作
光栅化阶段
0000000000000000
0000000000000000
0000011000000000
0000100100000000
0000100010000000
0001000001100000
0010000000010000
0100000000001000
0111111100000100
0000000011111110
0000000000000000
0000000000000000
帧缓存
说明：
可编程
可选
可配置
固定

1 着色器语言


应用程序
CPU
主存
渲染命令
OpenGL或
DirectX
显卡驱动
顶点数据、
纹理数据等
翻译命令
显卡
显存
GPU
深度
缓存
纹理
……

1 着色器语言


顶点数据
摄像机位置
光照纹理
几何阶段
顶点着色器
几何、细分着色器
裁剪
屏幕映射
着色语言
Shading Language
三角形设置
三角形遍历
片元着色器
片元操作
光栅化阶段
0000000000000000
0000000000000000
0000011000000000
0000100100000000
0000100010000000
0001000001100000
0010000000010000
0100000000001000
0111111100000100
0000000011111110
0000000000000000
0000000000000000
帧缓存

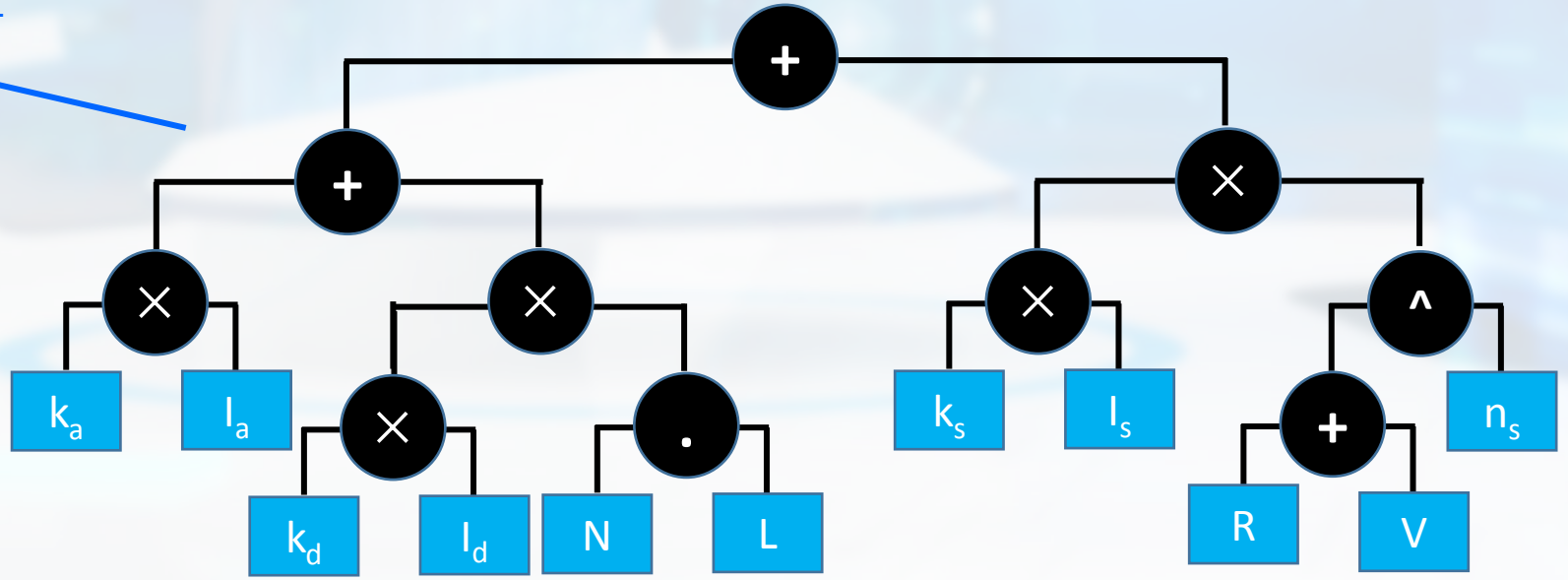
1 着色器语言

Phong光照公式的表达式树

Phong光照明模型的综合表述：由物体表面上一点P反射到视点的光强I为环境光的反射光强、理想漫反射光强和镜面反射光强的总和。

$$I = I_a K_a + I_p K_d (L \cdot N) + I_p K_s (R \cdot V)^n$$

Cook着色树



1 着色器语言

Pixar的Renderman

2 GLSL

OpenGL的着色器语言GLSL，也就是OpenGL Shading Language

2 GLSL

OpenGL的着色器语言GLSL，也就是OpenGL Shading Language

- ◆顶点着色器Vertex Shader
- ◆几何着色器Geometry Shader
- ◆曲面细分着色器Tessellation Shader
- ◆片元着色器Fragment Shader

2 GLSL

在OpenGL中使用着色器的流程：

step1 创建着色器对象

step2 源码关联到着色器对象

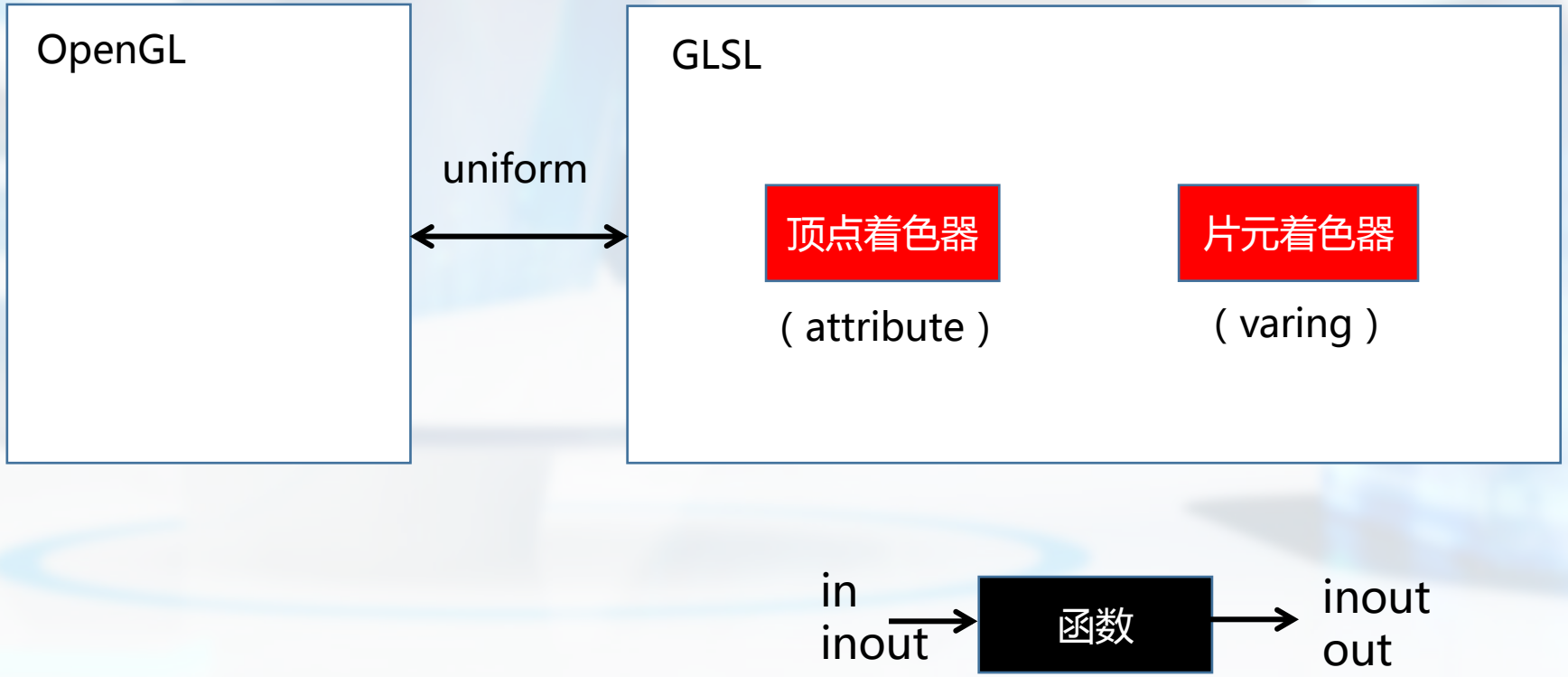
step3 编译着色器

step4 创建一个程序对象

step5 创建一个程序对象

2 GLSL

与OpenGL的通信



2 GLSL

数据类型

◆标量：整数（int）、无符号整数（uint）、布尔类型（bool）及浮点数（float）

◆矢量：一个矢量可以有2、3、4个分量

```
vec4 a=vec4(1.0,2.0,3.0,4.0);
```

◆矩阵：

```
mat2 m=mat2(1.0,2.0,3.0,4.0);
```

$$m=\begin{bmatrix}1.0 & 3.0\\ 2.0 & 4.0\end{bmatrix}$$

◆结构和数组： 结构体的成员或者数组的基类型可以为任意的数据类型

```
//lightsource为光源
struct lightsource{
  vec3 color;  //颜色
  vec3 position;//位置}
lightsource spotlights[10];//10个光源
```

2 GLSL

控制结构

◆循环结构：for、while和do-while

◆选择结构：if-then、if-then-else，Glsl3.0版本引入了switch结构

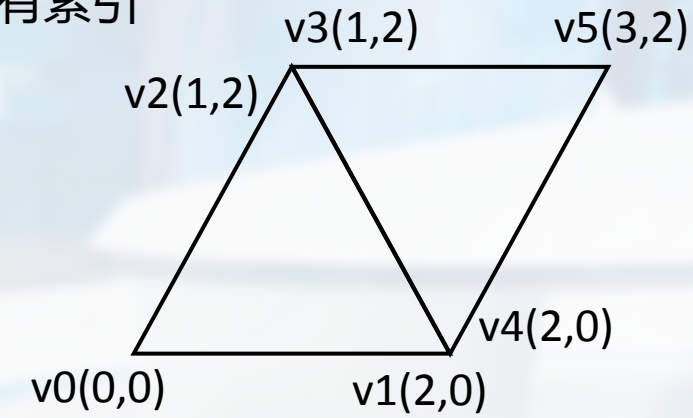
注意：goto语句和标签是不允许使用的

3 EBO、VBO和VAO

EBO

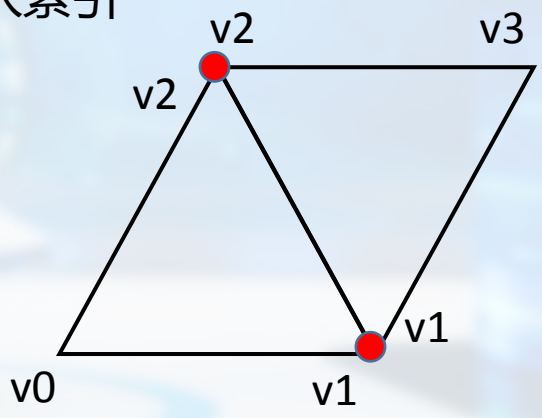
EBO(Element Buffer Object，也叫IBO:Index Buffer Object)索引缓冲区对象，这个缓冲区主要用来存储顶点的索引信息。

➢没有索引



[0,0, 2,0, 1,2, 1,2, 2,0, 3,2]

➢加入索引



[0,1,2,2,1,3]

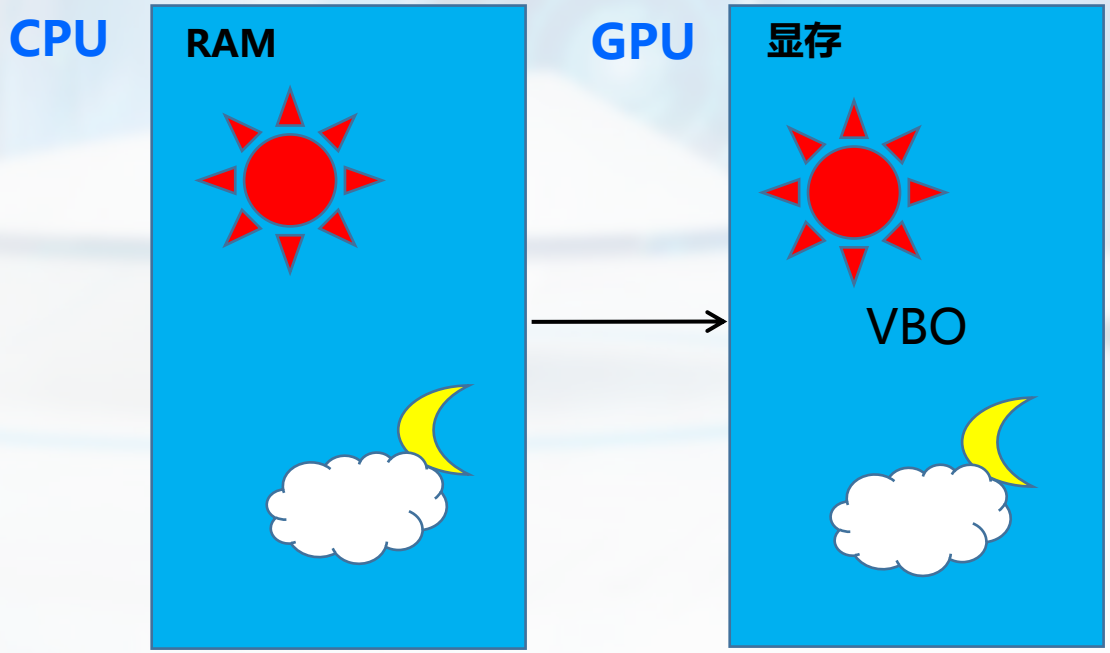
[0,0, 2,0, 1,2, 3,2]

3 EBO、VBO和VAO

VBO

VBO(Vertex Buffer Object)顶点缓冲区对象，主要用来存储顶点的各种信息。

好处：模型的顶点信息放进VBO，这样每次画模型时， 数据不用再从CPU的势力范围内存里取，而是直接从GPU的显存里取，从而提高效率。



3 EBO、VBO和VAO

VAO

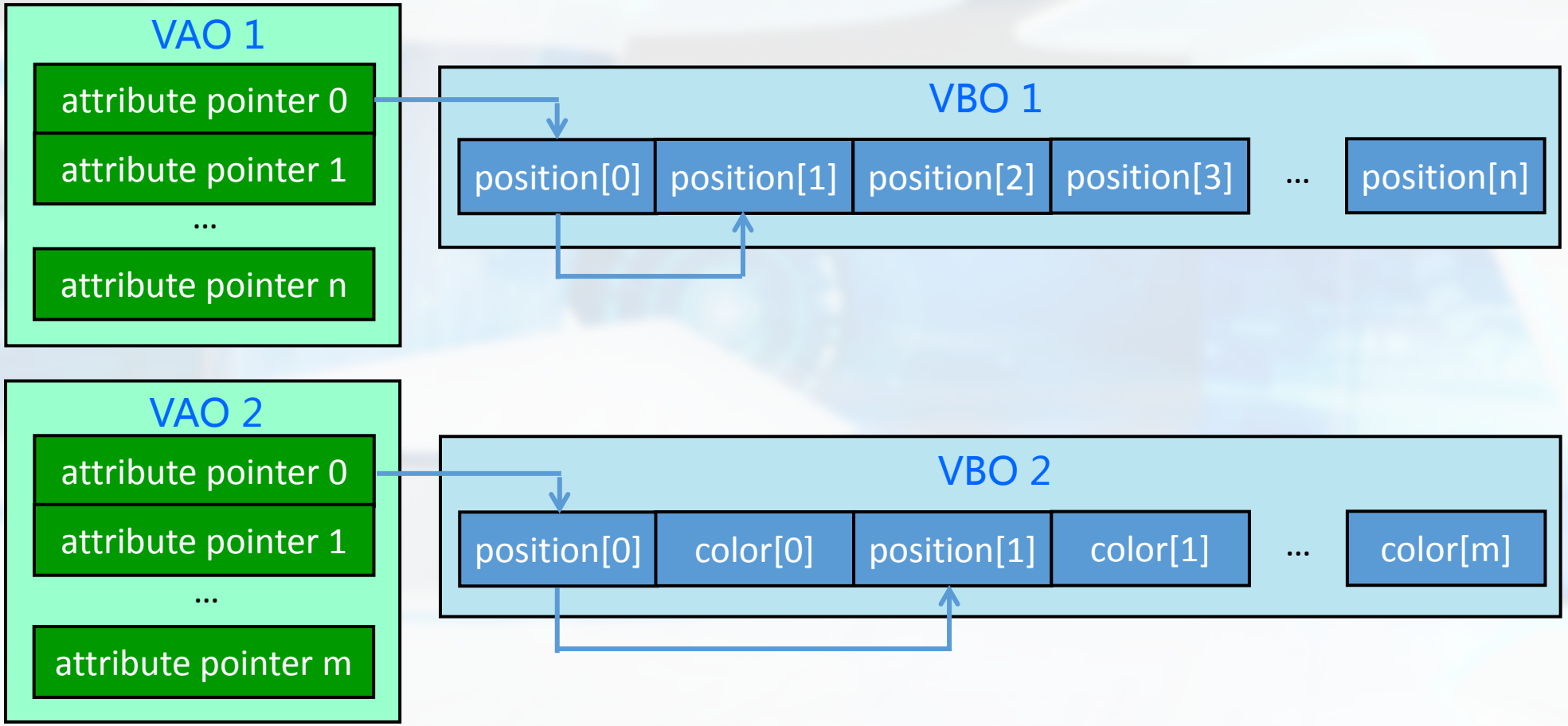
VAO(Vertex Arrary Object)顶点数组对象

VAO是一个保存了所有顶点数据属性的状态结合，它存储了顶点数据的格式以及顶点数据所需的VBO对象的引用。

VAO本身并没有存储顶点的相关属性数据，这些信息是存储在VBO中的，VAO相当于是对很多个VBO的引用，把一些VBO组合在一起作为一个对象统一管理。

3 EBO、VBO和VAO

VAO和VBO之间的关系



4 例程

绘制四边形：

4 例程

绘制四边形：

方法一：VAO+VBO绘制四边形

```
// 渲染之前VAO/VBO的绑定生成
// 顶点数据
float vertices[] = {
                // 第一个三角形
                0.5f, 0.5f, 0.0f,     // 右上
                0.5f, -0.5f, 0.0f,    // 右下
                -0.5f, -0.5f, 0.0f,   // 左下
                // 第二个三角形
                -0.5f, -0.5f, 0.0f,   // 左下
                0.5f, 0.5f, 0.0f,     // 右上
                -0.5f, 0.5f, 0.0f     // 左上
};
// 生成立方体的VAO、VBO
unsigned int VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
// 绑定VAO
glBindVertexArray(VAO);
// 绑定VBO并传入顶点数据
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);
// 设置顶点属性指针
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
glEnableVertexAttribArray(0);
// 解绑VBO
glBindBuffer(GL_ARRAY_BUFFER, 0);
// 解绑VAO
glBindVertexArray(0);
// 渲染时
// 使用之前链接好的着色器程序
glUseProgram(shaderProgram);
// 绑定VAO
glBindVertexArray(VAO);
// 用VAO绘制四边形
glDrawArrays(GL_TRIANGLES, 0, 6);
```

4 例程

绘制四边形：

方法二：VAO+VBO+EBO绘制四边形

```
// 渲染之前VAO/VBO/EBO的绑定生成
// 顶点数据
float vertices[] = {
        0.5f, 0.5f, 0.0f,     // 右上
        0.5f, -0.5f, 0.0f,    // 右下
        -0.5f, -0.5f, 0.0f,   // 左下
        -0.5f, 0.5f, 0.0f     // 左上
};
// 索引数据(注意这里是从0开始的)
unsigned int indices[] = {
        0, 1, 3,              // 第一个三角形
        1, 2, 3               // 第二个三角形
};
// 生成立方体的VAO、VBO和EBO
unsigned int VBO, VAO, EBO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glGenBuffers(1, &EBO);
// 绑定VAO
glBindVertexArray(VAO);
```

4 例程

```
// 绑定VBO并传入顶点数据
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
// 绑定EBO并传入索引数据
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
// 设置顶点属性指针
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
// 解绑VBO
glBindBuffer(GL_ARRAY_BUFFER, 0);
// 注意不要解绑EBO，因为EBO存储在VAO中
//glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
// 解绑VAO
glBindVertexArray(0);

// 渲染时
// 使用之前链接好的着色器程序
glUseProgram(shaderProgram);
// 绑定VAO(实际上不需要每次绑定)
glBindVertexArray(VAO);
// 用EBO绘制四边形
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```


谢谢
软件学院　万琳