中国科学技术大学
University of Science and Technology of China

计算机图形学理论和应用

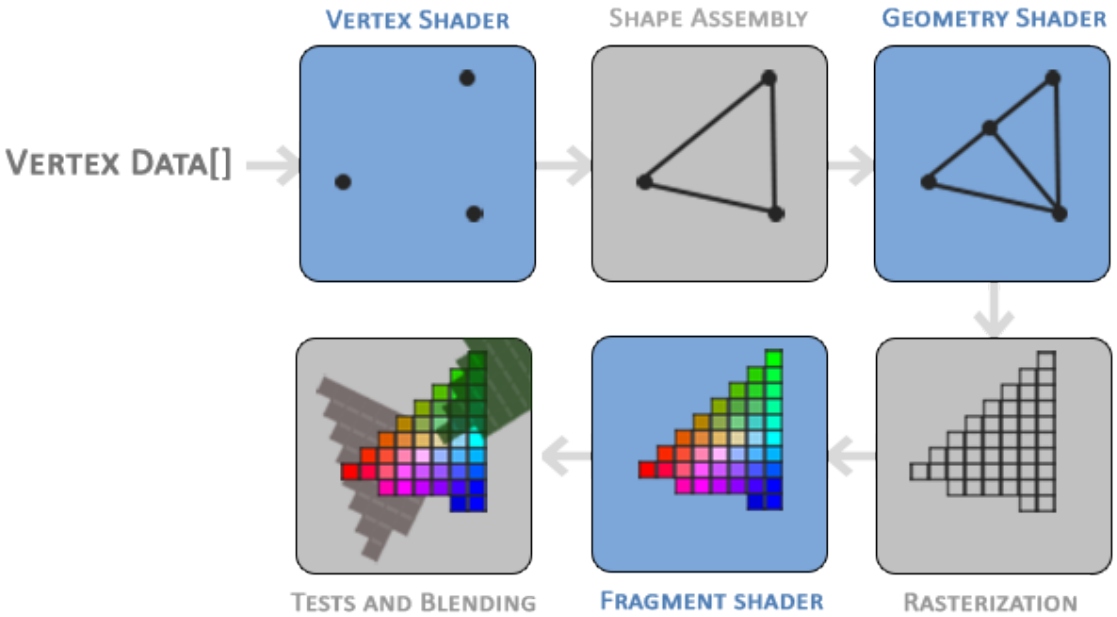
OpenGL 三角形

中国科学技术大学，计算机科学与技术学院

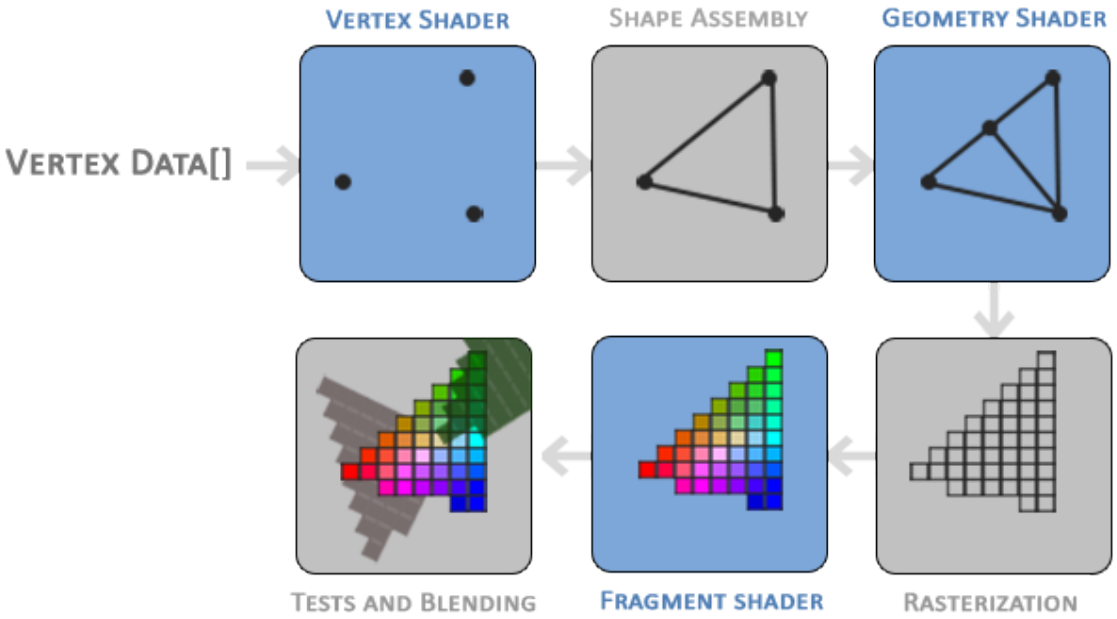
2022 年 9 月 16 日

图形渲染管线

- 图形渲染管线接受一组 3D 坐标，然后把他们转换成屏幕的有色 2D 像素输出。
- 划分成几个阶段，每个阶段把前一个阶段的输出作为输入。
- 每个阶段运行的程序称为着色器 (Shader)。
- OpenGL 着色器语言 (OpenGL Shading Language, GLSL)。



- 顶点着色器、图元装配、集合着色器、光栅化、片段着色器、测试与混合。
- 顶点着色器：把单独的顶点作为输入，把 3D 坐标转换为另一种 3D 坐标，允许我们对顶点属性做一些基本处理。
- 片段着色器：主要目的是计算一个像素的最终颜色，高级效果产生的地方。



- 顶点着色器、图元装配、集合着色器、光栅化、片段着色器、测试与混合。
- 现代 OpenGL，必须定义至少一个顶点着色器和一个片段着色器（因为 GPU 中没有默认的顶点/片段着色器）

```
#version 330 core
layout (location = 0) in vec3 aPos;

void main()
{
    gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);
}
```

- 把位置数据赋值给预定义的 gl_Position 变量，对输入数据未做处理。

```
unsigned int vertexShader;
vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);
```

- 创建着色器对象，将着色器源码附加到着色器对象上，然后编译它。

片段着色器


```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(1.0f, 0.5f, 0.2f, 1.0f);
}
```

- 片段着色器所做的是计算像素最后的颜色输出。
- 让片段着色器输出固定颜色。

```
unsigned int fragmentShader;
fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
glCompileShader(fragmentShader);
```

- 编译片段着色器。

```
unsigned int shaderProgram;
shaderProgram = glCreateProgram();

glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);

glLinkProgram(shaderProgram);
```

- 创建着色器程序对象，将着色器附加到程序对象上，链接程序即可。

```
glUseProgram(shaderProgram);

glDeleteShader(vertexShader);
glDeleteShader(fragmentShader);
```

- 激活着色器程序对象，删除着色器对象。
- 实现着色器类，可以更方便使用和移植。

顶点输入

```
float vertices[] = {
  -0.5f, -0.5f, 0.0f,
  0.5f, -0.5f, 0.0f,
  0.0f,  0.5f, 0.0f
  };
```

- OpenGL 仅当 3D 坐标在 3 个轴（x、y 和 z）上-1.0 到 1.0 的范围内时才处理。
- 将定义的顶点发送给顶点着色器，它会在 GPU 上创建内存来存储顶点数据。
- 谁来管理内存呢？

顶点缓冲对象 (Vertex Buffer Object, VBO)

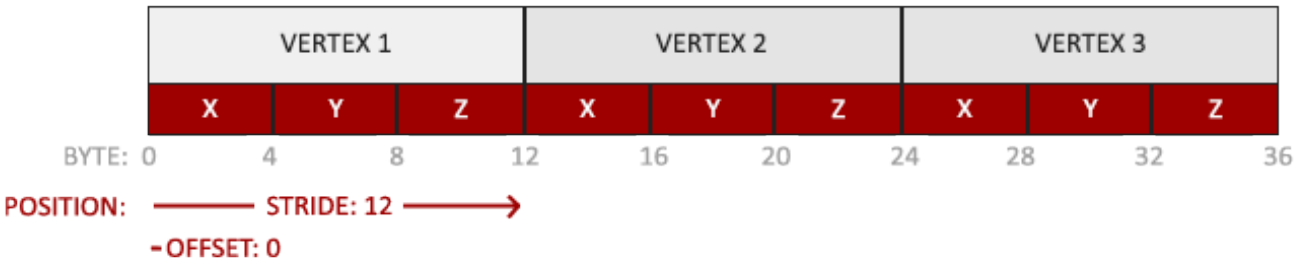

```
unsigned int VBO;
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
```

- 使用 glGenBuffers 函数和一个缓冲 ID 生成一个 VBO 对象。
- 使用 glBindBuffer 函数把新创建的缓冲绑定到 GL_ARRAY_BUFFER 目标上，这样在 GL_ARRAY_BUFFER 上的缓冲调用都会用来配置当前绑定的 VBO。

```
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

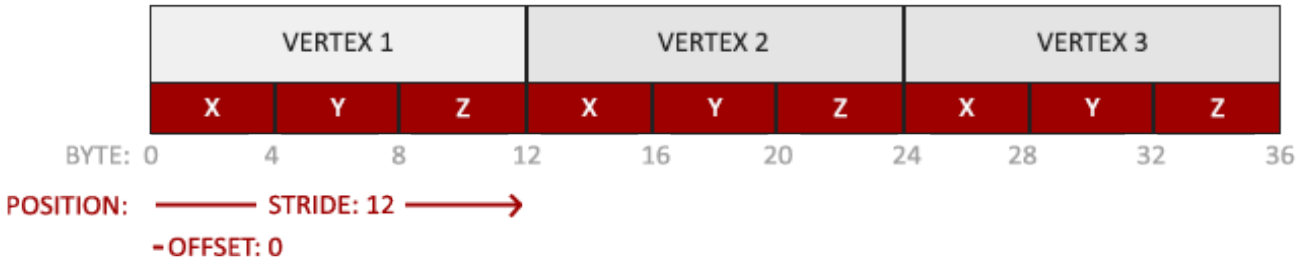
- glBufferData 用来把用户定义的数据复制到当前绑定的缓冲中。
- (目标缓冲的类型，传输数据的大小，发送的实际数据，如何管理给定的数据)
- OpenGL 如何解释数据呢？

- 必须手动指定输入数据的哪一部分对应顶点着色器的哪一个顶点属性。




```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0)
glEnableVertexAttribArray(0);
```

- 使用 glVertexAttribPointer 函数告诉 OpenGL 该如何解析顶点数据。




```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0)
glEnableVertexAttribArray(0);
```

- glVertexAttribPointer
  1. 配置的顶点属性，0 与顶点着色器中 layout(location = 0) 对应，说明我们把数据传入到这个属性中。
  2. 顶点属性的大小，3 个值。
  3. 数据的类型。
  4. 数据是否需要标准化，不需要。
  5. 步长，连续的顶点属性之间的间隔。
  6. 数据在缓冲中起始位置的偏移量。

- 绘制一个物体的代码。

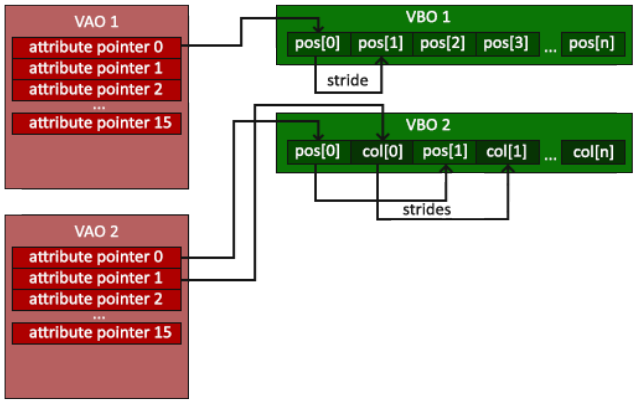
```
// 0. 复制顶点数组到缓冲中供OpenGL使用
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
// 1. 设置顶点属性指针
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0)
glEnableVertexAttribArray(0);
// 2. 当我们渲染一个物体时要使用着色器程序
glUseProgram(shaderProgram);
// 3. 绘制物体
someOpenGLFunctionThatDrawsOurTriangle();
```

- 每次绘制需要重复这一过程，物体过多会不会太麻烦？

顶点数组对象 (Vertex Array Object, VAO)

- 将顶点属性的调用存储在 VAO 中，这样当配置顶点属性指针后，后续只要绑定 VAO 就可以了。
- OpenGL 的核心模式要求我们使用 VAO，如果绑定 VAO 失败，OpenGL 会拒绝绘制任何东西

顶点数组对象 (Vertex Array Object, VAO)



- VAO 存储
  1. glEnableVertexAttribArray 和 glDisableVertexAttribArray 的调用。
  2. 通过 glVertexAttribPointer 设置的顶点属性配置。
  3. 通过 glVertexAttribPointer 调用与顶点属性关联的顶点缓冲对象。

顶点数组对象 (Vertex Array Object, VAO)


```
unsigned int VAO;
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);
```

- 创建一个 VAO。
- 使用 glBindVertexArray 绑定 VAO。
- 之后起，我们应该绑定和配置对应的 VBO 和属性指针。

顶点数组对象 (Vertex Array Object, VAO)

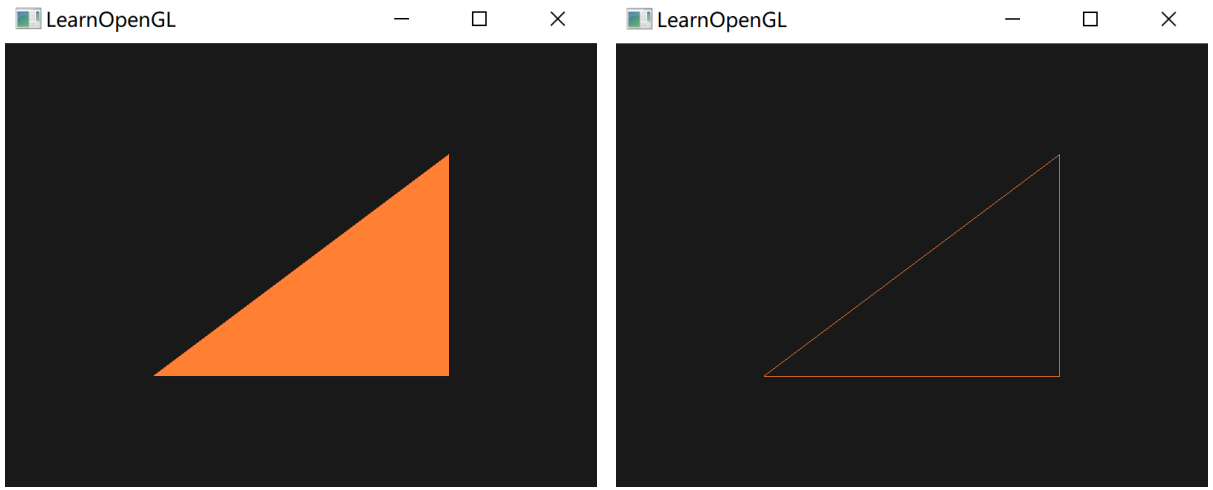

```
// 1. 绑定VAO
glBindVertexArray(VAO);
// 2. 把顶点数组复制到缓冲中供OpenGL使用
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
// 3. 设置顶点属性指针
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0)
glEnableVertexAttribArray(0);
[...]
// ..:: 绘制代码（渲染循环中） :: ..
// 4. 绘制物体
glUseProgram(shaderProgram);
glBindVertexArray(VAO);
someOpenGLFunctionThatDrawsOurTriangle();
```

- 首先要生成/配置所有的 VAO。
- 绘制物体的时候就拿出相应的 VAO，绑定它，绘制完物体后，再解绑 VAO。

绘制三角形


```
glUseProgram(shaderProgram);
glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, 0, 3);
```

- glDrawArrays 函数第一个参数是我们打算绘制的 OpenGL 图元的类型，GL_TRIANGLES 说明我们要绘制三角形。
- 第二个参数指定了顶点数组的起始索引，填 0.
- 最后一个参数指定我们要绘制多少个顶点。



1. 基于 HelloTriangle_VAO.cpp，修改三角形颜色为科大蓝 (1, 75, 150)。
2. 基于 HelloTriangle_VAO.cpp，再添加三个顶点，绘制出两个三角形。

谢谢!

[1]参考资料:https://learnopengl-cn.github.io/