

计算机图形学原理

-----教程2

李瑞辉

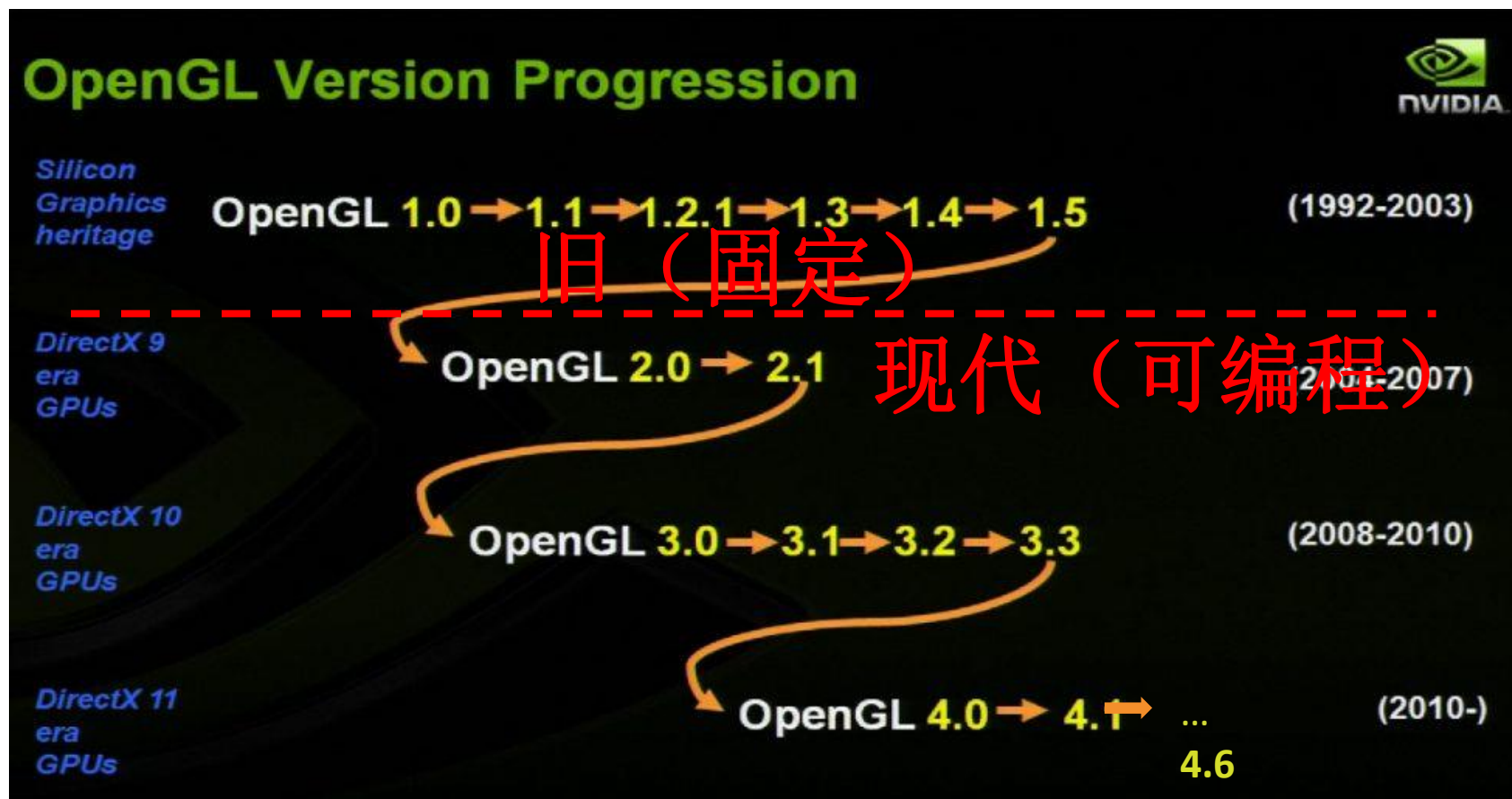
大纲

旧 vs. 现代 OpenGL

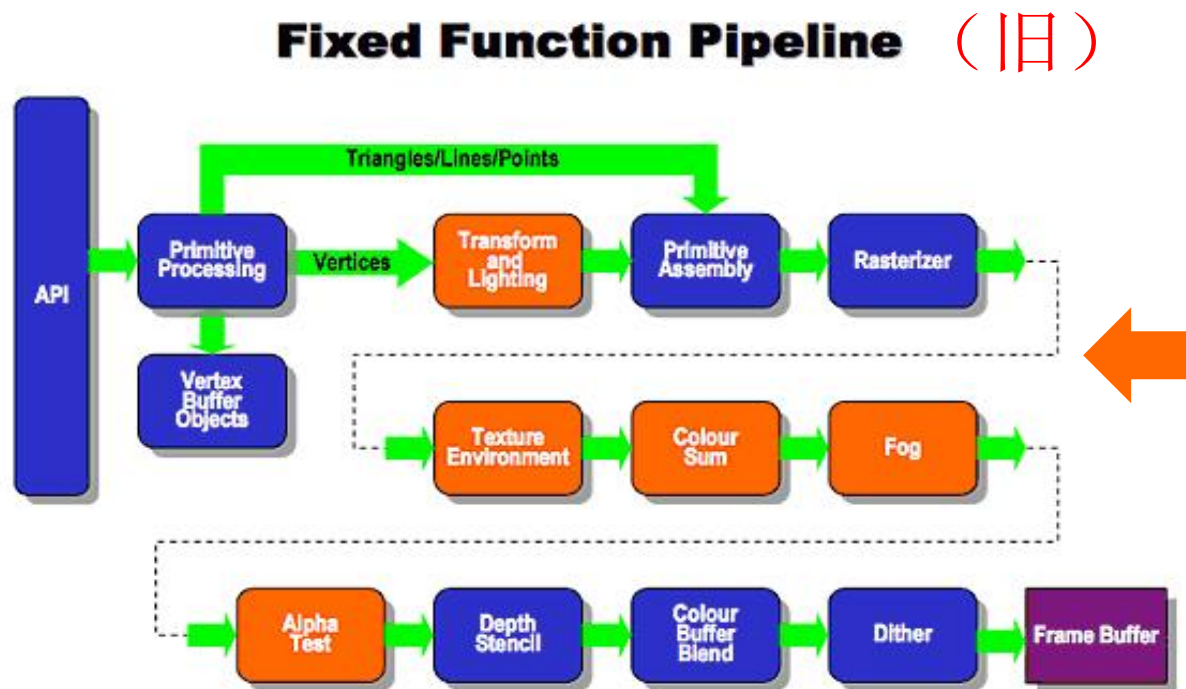
- 识别计算机上的 OpenGL 版本
- 基本的 OpenGL 编程
- 作业1的整体组件

旧版本VS与现代

OpenGL历史

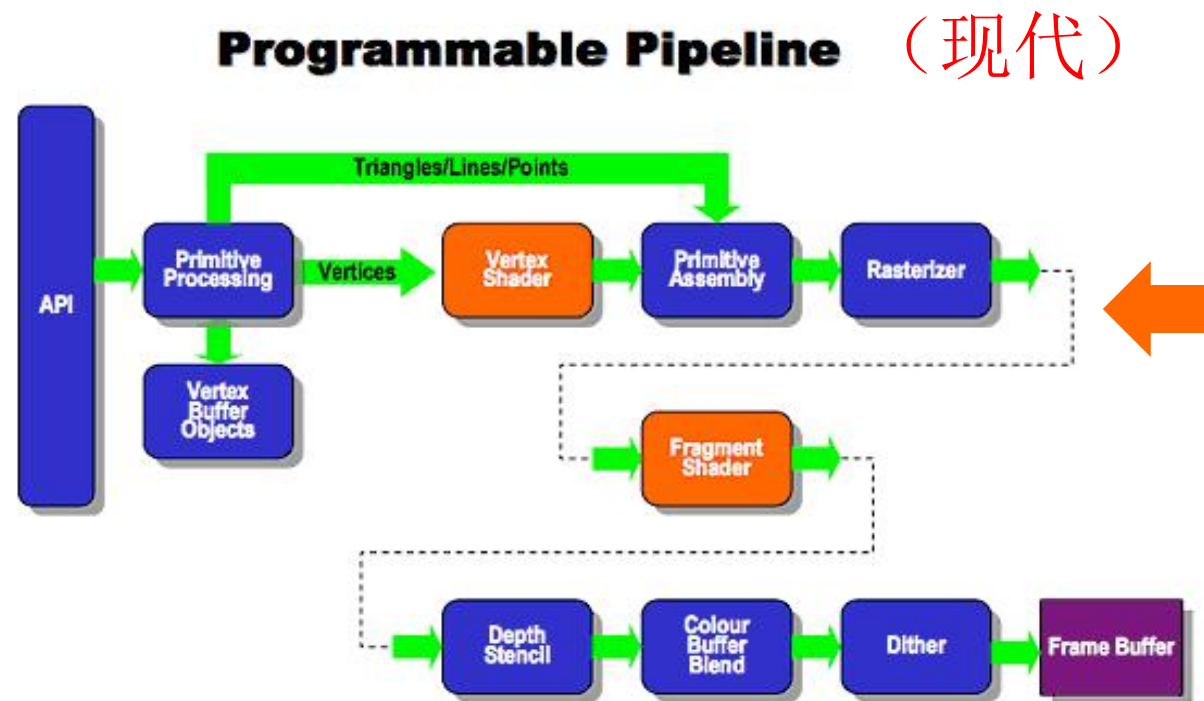


旧版本VS与现代



- 旧版本的 OpenGL 使用固定功能管道。
- 不可控——几何体转换的确切方法，以及片段获取深度和颜色值的方式都内置在硬件中，无法更改。

旧版本VS与现代



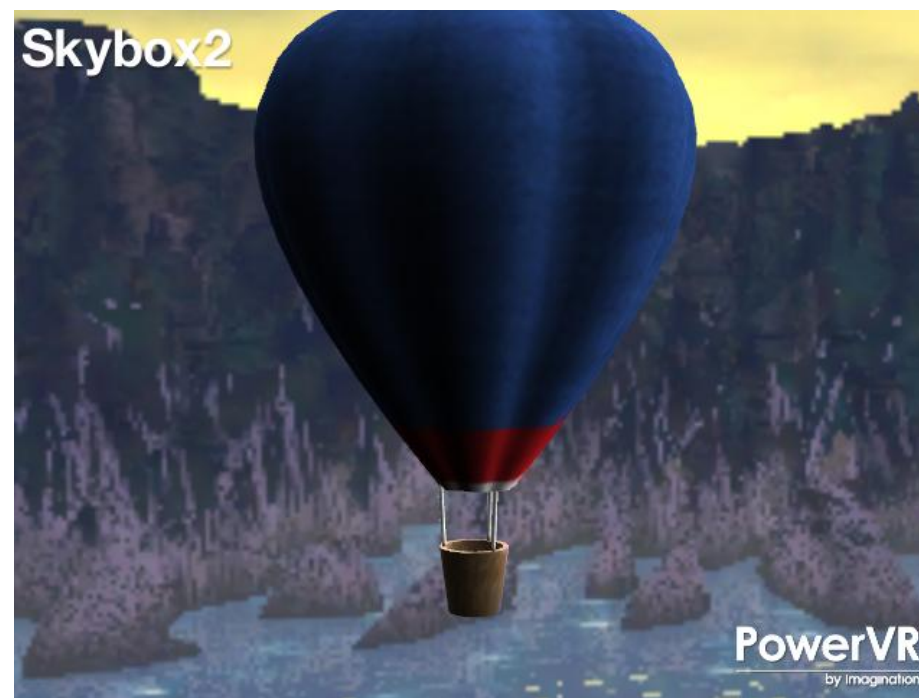
- 现代 GPU 具有可编程流水线。
- 以前的内置阶段已被可以通过称为“着色器”的代码控制的阶段所取代。

旧版本VS与现代

比较



固定管道



可编程流水线

旧版本VS与现代

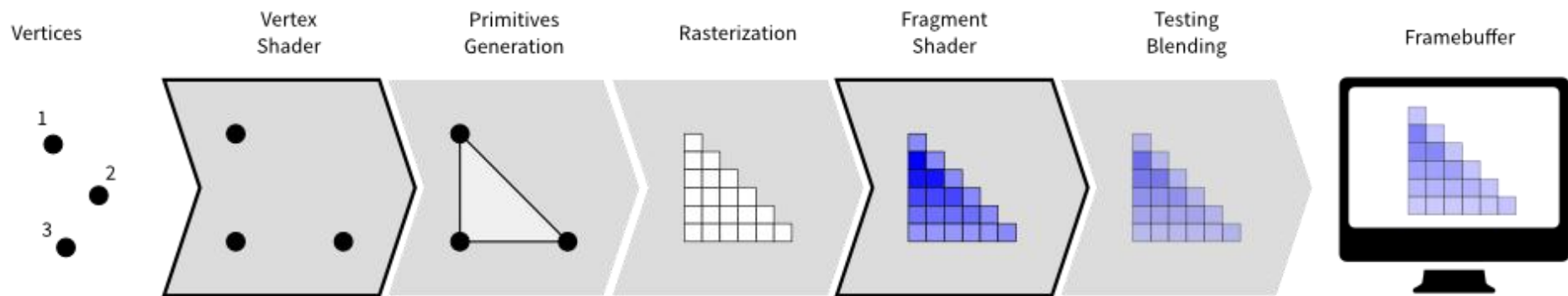


图 1 渲染管线

构建在 GPU 上并在渲染管道期间执行的程序片段（使用C 风格语言glsl）。有许多类型的着色器在渲染管道的不同阶段起作用。

其中顶点着色器和片段着色器最为重要。

- 顶点着色器作用于顶点并且应该输出顶点位置。
- 片段着色器作用于片段级别并且应该输出颜色。

旧版本VS与现代

GLSL (OpenGL 着色语言)

- 高级着色语言让开发人员可以直接控制图形管道。
- 在本课程中，我们只需要进行简单的GLSL编程即可。
(更多信息: <https://www.opengl.org/documentation/glsl/>)

```
VertexShaderCode.glsl  ↗ ✕
1   #version 430
2
3   in layout(location = 0) vec3 position;
4   in layout(location = 1) vec3 vertexColor;
5
6   out vec3 theColor;
7
8   void main()
9   {
10      gl_Position = position;
11      theColor = vertexColor;
12  }
13
```

顶点着色器的一个最简单的例子

旧版本VS与现代

旧的 OpenGL 代码（示例）

```
glBegin (type);  
  
glVertex3f ( ... );  
  
glVertex3f ( ... );  
  
glVertex3f ( ... );  
  
.....  
  
glEnd ();
```

```
glMatrixMode ( GL_MODELVIEW );  
glLoadIdentity ();  
glPushMatrix ();  
    glTranslatef ( ball_X , ball_Y , ball_Z );  
    glRotatef ( ball_ang , ball_dirX , ball_dirY , ball_dirZ );  
    glScalef ( ball_Sx , ball_Sy , ball_Sz );  
    抽奖球 ();  
glPopMatrix ();  
glPushMatrix ();  
    glTranslatef ( cube_X , cube_Y , cube_Z );  
    glRotatef ( cube_ang , cube_dirX , cube_dirY , cube_dirZ );  
    glScalef ( cube_Sx , cube_Sy , cube_Sz );  
    绘制立方体 ();  
glPopMatrix ();
```

旧版本VS与现代

现代 OpenGL 代码（示例）。 **VAO 和 VBO：花时间了解！**

- **Vertex Array Object (VAO)**: 一个状态对象，它存储了一个顶点数组（对象，例如三角形）的所有状态。一个对象需要一个 VAO。直观上，VAO 是一个对象的符号或昵称。
- **顶点缓冲区对象 (VBO)**: 保存顶点数组数据的缓冲区对象。

```
const GLfloat triangle[] =  
{  
x, y, z    -0.5f, -0.5f, +0.0f, // left  
r, g, b    +1.0f, +0.0f, +0.0f, // color  
  
           +0.5f, -0.5f, +0.0f, // right  
           +1.0f, +0.0f, +0.0f,  
  
           +0.0f, +0.5f, +0.0f, // top  
           +1.0f, +0.0f, +0.0f,  
};
```

定义对象数据

```
GLuint vaoID;  
glGenVertexArrays(1, &vaoID);  
glBindVertexArray(vaoID); //first VAO  
  
GLuint vboID;  
glGenBuffers(1, &vboID);  
glBindBuffer(GL_ARRAY_BUFFER, vboID);  
glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW);  
  
// 1st attribute: vertex position  
glEnableVertexAttribArray(0);  
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), 0);  
  
// 2nd attribute: vertex color  
glEnableVertexAttribArray(1);  
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),  
                      (char*)(3 * sizeof(float)));
```

使用 OpenGL 语言（VAO、VBO）将对象数据发送到硬件

旧版本VS与现代

OpenGL function replacements

`glRotate[fd]`

`glm::rotate`

`glScale[fd]`

`glm::scale`

`glTranslate[fd]`

`glm::translate`

`glLoadIdentity`

The default constructor of all matrix types creates an identity matrix.

`glm::mat4x4`

Per the GLSL specification, the multiplication operator is overloaded for all matrix types. Multiplying two matrices together will perform matrix multiplication.

`glm::mat4x4`

`glm::transpose`

`glm::mat4x4`

Combine the last two.

`glm::frustum`

`glm::frustum`

`glm::ortho`

`glm::ortho`

`gluLookAt`

`glm::lookAt`

GLU function replacements

`gluOrtho2D`

`glm::ortho`

`gluPerspective`

`glm::perspective`

`gluProject`

`glm::project`

`gluUnProject`

`glm::unProject`

尽量避免使用这些已弃用的功能！

GLM (OpenGL Mathematics) 是一个提供相同功能且易于使用的 C++ 数学库。

大纲

- 旧版本 vs. 现代 OpenGL

识别计算机上的 **OpenGL** 版本

- 基本的 OpenGL 编程
- 作业 1 的整体组成部分

OpenGL 版本检查

检查计算机上的 OpenGL #1:

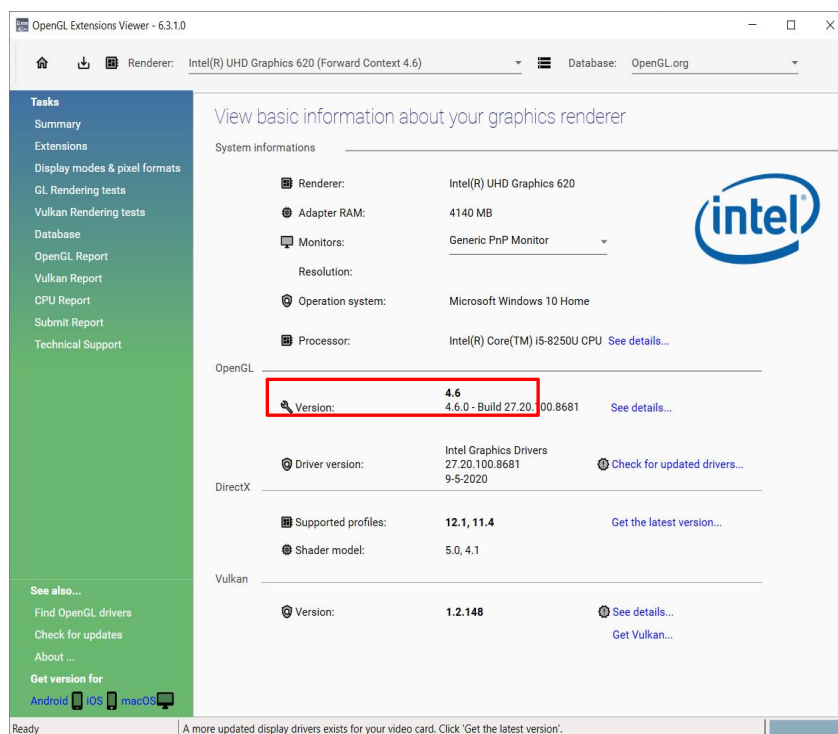
```
void get_OpenGL_info() {  
    // OpenGL information  
    const GLubyte* name = glGetString(GL_VENDOR);  
    const GLubyte* renderer = glGetString(GL_RENDERER);  
    const GLubyte* glversion = glGetString(GL_VERSION);  
    std::cout << "OpenGL company: " << name << std::endl;  
    std::cout << "Renderer name: " << renderer << std::endl;  
    std::cout << "OpenGL version: " << glversion << std::endl;  
}
```

```
OpenGL company: NVIDIA Corporation  
Renderer name: GeForce RTX 2060/PCIe/SSE2  
OpenGL version: 3.3.0 NVIDIA 452.06
```

OpenGL 版本检查

检查计算机上的 OpenGL #2:

- OpenGL 扩展查看器: <http://www.realtech-vr.com/glview/>



大纲

- 旧版本 vs. 现代 OpenGL
- 识别计算机上的 OpenGL 版本

基本的 OpenGL 编程





- 作业 1 的整体组成部分

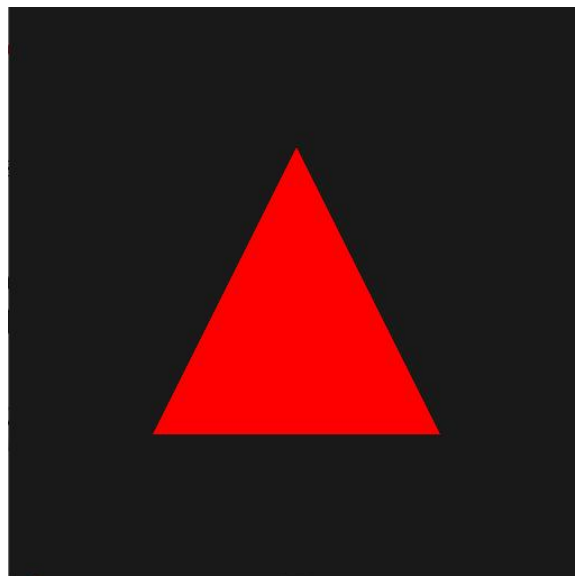
编程

红三角Demo代码（重要！！） （可以从Blackboard下载）

目标：

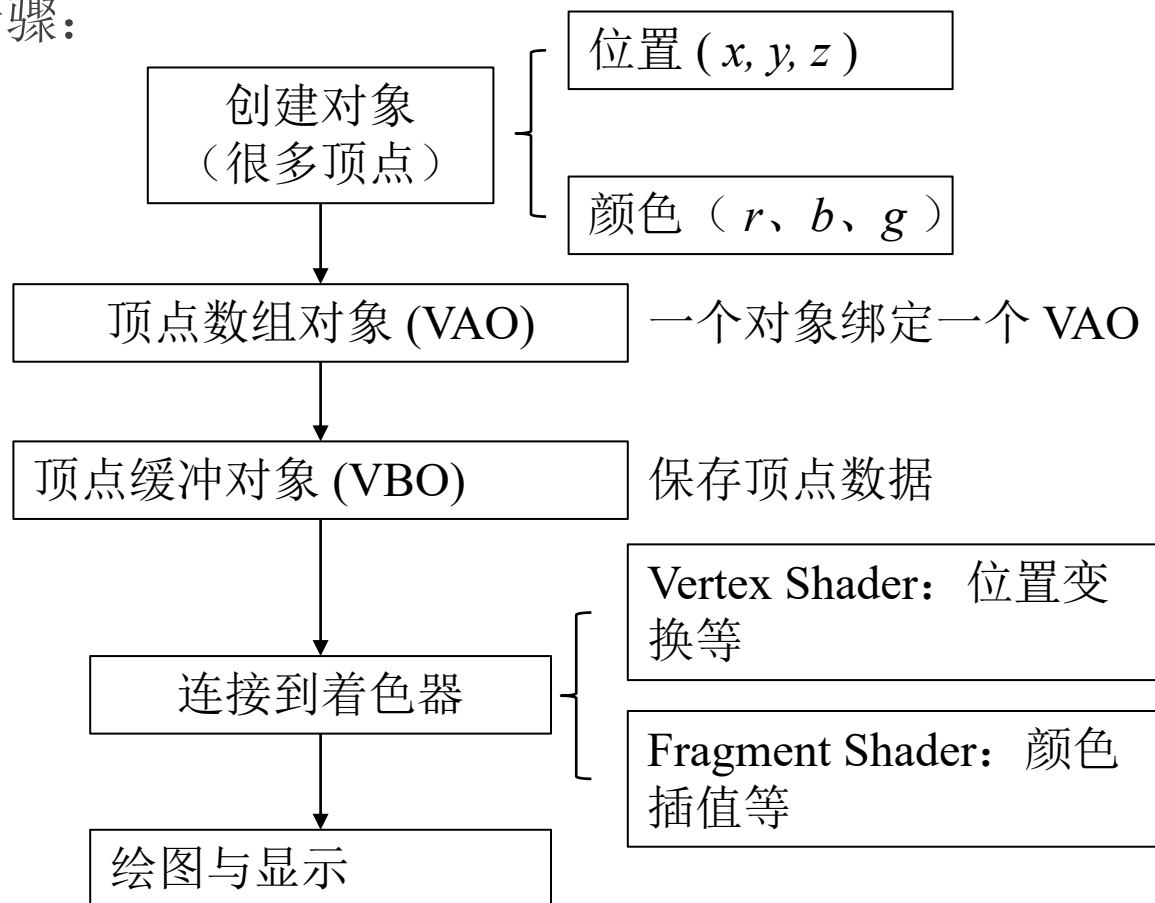
- 创建一个背景色为黑色的窗口。
- 在窗口中央画一个红色三角形。
- 按“a”键将三角形向左移动。
- 按“d”键将三角形向右移动。

 FragmentShaderCode.glsl
 main.cpp
 ReadMe.txt
 VertexShaderCode.glsl



编程

渲染对象的主要步骤：



编程

主要功能（*main.cpp*）：

```
int main(int argc, char* argv[]) {
    GLFWwindow* window;

    /* Initialize the glfw */
    if (!glfwInit()) {
        std::cout << "Failed to initialize GLFW" << std::endl;
        return -1;
    }

    /* glfw: configure; necessary for MAC */
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    /* do not allow resizing */
    glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);

    /* Create a windowed mode window and its OpenGL context */
    window = glfwCreateWindow(512, 512, argv[0], NULL, NULL);
    if (!window) {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
}
```

```
/* Make the window's context current */
glfwMakeContextCurrent(window);
glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
glfwSetKeyCallback(window, key_callback);

/* Initialize the glew */
if (GLEW_OK != glewInit()) {
    std::cout << "Failed to initialize GLEW" << std::endl;
    return -1;
}

get_OpenGL_info();
initializedGL();

/* Loop until the user closes the window */
while (!glfwWindowShouldClose(window)) {
    /* Render here */
    paintGL();

    /* Swap front and back buffers */
    glfwSwapBuffers(window);

    /* Poll for and process events */
    glfwPollEvents();
}

glfwTerminate();
return 0;
}
```

编程

初始化GL () 函数（只运行一次）：

```
void initializedGL(void) {  
    // run only once  
    sendDataToOpenGL();  
    installShaders();  
}
```

- 1) 创建三角形对象
（普通 C++ 数组）
- 2) 向硬件发送数据
（OpenGL语言）

加载着色器代码（ glsl语言）

installShaders ()不需要自己写（差不多就是个模板），
我们会在您做作业时为您提供此功能。

编程

```
void sendDataToOpenGL() {  
    const GLfloat triangle[] =  
    {  
        -0.5f, -0.5f, +0.0f, // left  
        +1.0f, +0.0f, +0.0f, // color  
  
        +0.5f, -0.5f, +0.0f, // right  
        +1.0f, +0.0f, +0.0f,  
  
        +0.0f, +0.5f, +0.0f, // top  
        +1.0f, +0.0f, +0.0f,  
    };  
  
    GLuint vaoID;  
    glGenVertexArrays(1, &vaoID);  
    glBindVertexArray(vaoID); //first VAO  
  
    GLuint vboID;  
    glGenBuffers(1, &vboID);  
    glBindBuffer(GL_ARRAY_BUFFER, vboID);  
    glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW);  
  
    // 1st attribute: vertex position  
    glEnableVertexAttribArray(0);  
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), 0);  
  
    // 2nd attribute: vertex color  
    glEnableVertexAttribArray(1);  
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),  
        (char*)(3 * sizeof(float)));  
}
```

创建对象：例如，三角形的 3 个顶点。
每个顶点：一个 (x, y, z) 位置和一个 (r, g, b) 颜色。

一个对象绑定到一个**VAO**：OpenGL生成一个uint *vaoID* 参考三角形，这个 *vaoID* \Leftrightarrow 三角形。

VBO将数据发送到硬件。
将位置和颜色作为两个属性分别发送。
数据连接到绑定 虚名。

编程

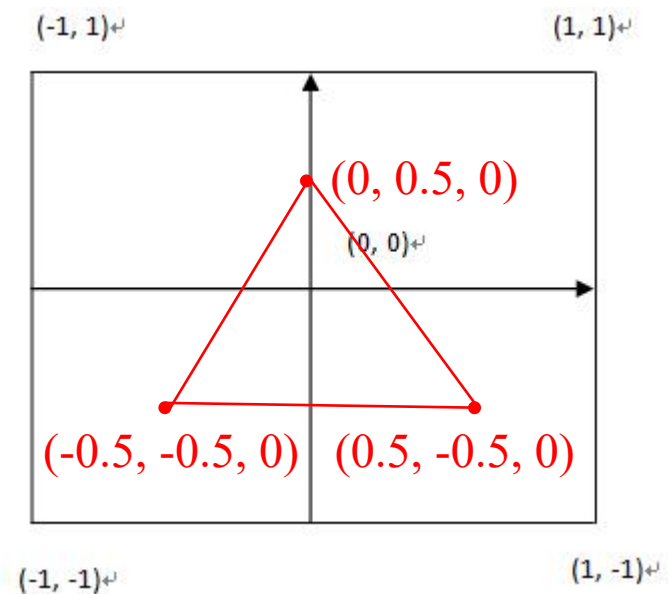
表1 OpenGL变量类型和对应的C数据类型

OpenGL Data Type	Internal Representation	Defined as C Type	C Literal Suffix
GLbyte	8-bit integer	Signed char	b
GLshort	16-bit integer	Short	s
GLint, GLsizei	32-bit integer	Long	i
GLfloat, GLclampf	32-bit floating point	Float	f
GLdouble, GLclampd	64-bit floating point	Double	d
GLubyte, GLboolean	8-bit unsigned integer	Unsigned char	ub
GLushort	16-bit unsigned integer	Unsigned short	us
GLuint, GLenum	32-bit unsigned integer	Unsigned long	ui

- *GLsizei*是一个 OpenGL 变量，表示由整数表示的大小参数。
- *clamp*用于颜色合成，代表颜色振幅。

编程

```
void sendDataToOpenGL() {  
    const GLfloat triangle[] =  
    {  
        -0.5f, -0.5f, +0.0f, // left  
        +1.0f, +0.0f, +0.0f, // color  
  
        +0.5f, -0.5f, +0.0f, // right  
        +1.0f, +0.0f, +0.0f,  
  
        +0.0f, +0.5f, +0.0f, // top  
        +1.0f, +0.0f, +0.0f,  
    };  
  
    GLuint vaoID;  
    glGenVertexArrays(1, &vaoID);  
    glBindVertexArray(vaoID); //first VAO  
  
    GLuint vboID;  
    glGenBuffers(1, &vboID);  
    glBindBuffer(GL_ARRAY_BUFFER, vboID);  
    glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW);  
  
    // 1st attribute: vertex position  
    glEnableVertexAttribArray(0);  
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), 0);  
  
    // 2nd attribute: vertex color  
    glEnableVertexAttribArray(1);  
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),  
        (char*)(3 * sizeof(float)));  
}
```



二维坐标系

编程

(VAO & VBO: 直观理解)

```
void sendDataToOpenGL() {
    const GLfloat triangle[] =
    {
        -0.5f, -0.5f, +0.0f, // left
        +1.0f, +0.0f, +0.0f, // color

        +0.5f, -0.5f, +0.0f, // right
        +1.0f, +0.0f, +0.0f,

        +0.0f, +0.5f, +0.0f, // top
        +1.0f, +0.0f, +0.0f,
    };

    GLuint vaoID;
    glGenVertexArrays(1, &vaoID);
    glBindVertexArray(vaoID); //first VAO

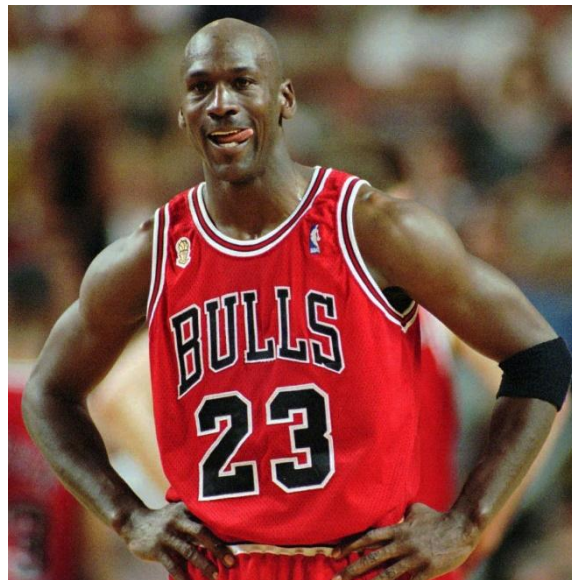
    GLuint vboID;
    glGenBuffers(1, &vboID);
    glBindBuffer(GL_ARRAY_BUFFER, vboID);
    glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW);

    // 1st attribute: vertex position
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), 0);

    // 2nd attribute: vertex color
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),
        (char*)(3 * sizeof(float)));
}
```

顶点数组对象

顶点缓冲对象



这个人（一个对象）被命名为 Michael Jordan 或编号为 No.23 (*vaoID*)。所以，我们可以用 No.23 (*vaoID*) 来指代这个人。

这个人有一些属性，例如，*height*，*weight*等。我们使用 VBO 来存储这些数据并发送到某个地方。

编程

两个属性

```
void sendDataToOpenGL() {  
    const GLfloat triangle[] =  
    {  
        -0.5f, -0.5f, +0.0f, // left  
        +1.0f, +0.0f, +0.0f, // color  
  
        +0.5f, -0.5f, +0.0f, // right  
        +1.0f, +0.0f, +0.0f,  
  
        +0.0f, +0.5f, +0.0f, // top  
        +1.0f, +0.0f, +0.0f,  
    };  
  
    GLuint vaoID;  
    glGenVertexArrays(1, &vaoID);  
    glBindVertexArray(vaoID); //first VAO  
  
    GLuint vboID;  
    glGenBuffers(1, &vboID);  
    glBindBuffer(GL_ARRAY_BUFFER, vboID);  
    glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW);  
  
    // 1st attribute: vertex position  
    glEnableVertexAttribArray(0);  
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), 0);  
  
    // 2nd attribute: vertex color  
    glEnableVertexAttribArray(1);  
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),  
        (char*)(3 * sizeof(float)));  
}
```

x, y, z
 r, g, b

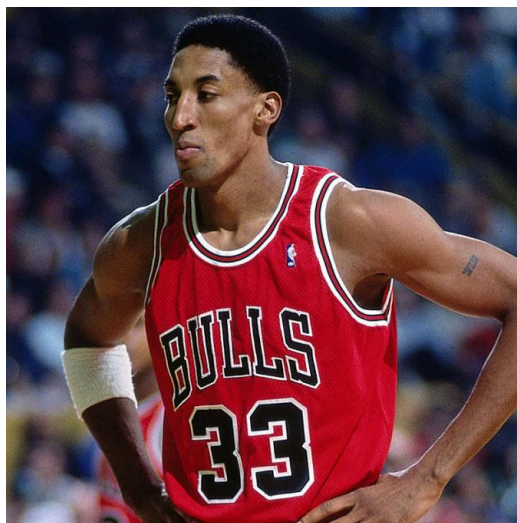
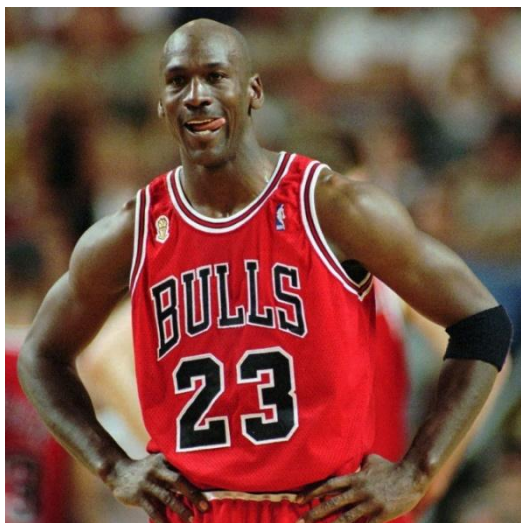
通过数据发送 (VBO) 将
VAO 与
对象链接起来

编程

(VAO & VBO: 直观理解)

OpenGL 是一个状态机:

- 如果您有多个对象, 则对对象逐个进行操作



调用一个人的名字 (绑定一个对象的 *vaoID*)。
做这个操作。

sendDataToOpenGL 中:

No.23 进行操作:

- 绑定 23号 的 *vaoID*
- 使用 VBO

No.33 进行操作:

- 绑定 33号 的 *vaoID*
- 发送 No.33 的数据

.....

或者在 *paintGL* 中:

No.23 进行操作:

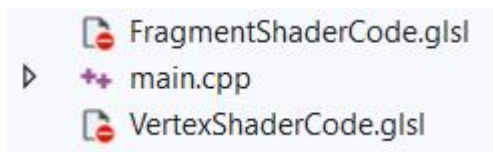
- 绑定 23号 的 *vaoID*
- 23号

No.33 进行操作:

- 绑定 33号 的 *vaoID*
- 33号

编程

顶点着色器和片段着色器



```
VertexShaderCode.glsl  FragmentShaderCode.glsl  main.cpp
1  #version 430
2
3  in layout(location=0) vec3 position;
4  in layout(location=1) vec3 vertexColor;
5
6  uniform mat4 modelTransformMatrix;
7
8  out vec3 theColor;
9
10 void main()
11 {
12     vec4 v = vec4(position, 1.0);
13     vec4 out_position = modelTransformMatrix * v;
14     gl_Position = out_position;
15     theColor = vertexColor;
16 }
```

回忆两个属性

```
VertexShaderCode.glsl  FragmentShaderCode.glsl  main.cpp
1  #version 430
2
3  out vec4 Color;
4  in vec3 theColor;
5
6  void main()
7  {
8      Color = vec4(theColor, 1.0);
9  }
10
```

一个简单的例子。

(赋值时需要修改)

- *win*和*mac*之间的细微差别。
参考三角形演示代码。
- *in* : 由 VAO 和 VBO (*sendDataToOpenGL*) 给出。
- *uniform* : 由 *main.cpp* (*paintGL*) 给出。

编程

顶点着色器和片段着色器

```
GLuint vaoID;
glGenVertexArrays(1, &vaoID);
glBindVertexArray(vaoID); //first VAO

GLuint vboID;
glGenBuffers(1, &vboID);
glBindBuffer(GL_ARRAY_BUFFER, vboID);
glBufferData(GL_ARRAY_BUFFER, sizeof(triangle), triangle, GL_STATIC_DRAW);

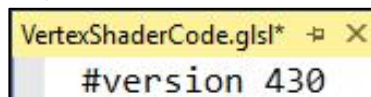
// 1st attribute: vertex position
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);

// 2nd attribute: vertex color
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
```

```
VertexShaderCode.glsl  FragmentShaderCode.glsl  main.cpp
1   #version 430
2
3   in layout(location=0) vec3 position;
4   in layout(location=1) vec3 vertexColor;
5
6   uniform mat4 modelTransformMatrix;
7
8   out vec3 theColor;
9
10  void main()
11  {
12      vec4 v = vec4(position, 1.0);
13      vec4 out_position = modelTransformMatrix * v;
14      gl_Position = out_position;
15      theColor = vertexColor;
16  }
```

编程

第一行Shader代码:



⇒ 指定应该使用
哪个版本的 GLSL来编译/链接着色器程序。

GLSL 版本与特定版本的 OpenGL API 一起发展。
只有 OpenGL 版本 3.3 及更高版本的 GLSL 和 OpenGL 版本号匹配。
这些 GLSL 和 OpenGL 版本在下表中相关。

GLSL Version	OpenGL Version	Date	Shader Preprocessor
1.10.59 ^[1]	2.0	April 2004	#version 110
1.20.8 ^[2]	2.1	September 2006	#version 120
1.30.10 ^[3]	3.0	August 2008	#version 130
1.40.08 ^[4]	3.1	March 2009	#version 140
1.50.11 ^[5]	3.2	August 2009	#version 150
3.30.6 ^[6]	3.3	February 2010	#version 330
4.00.9 ^[7]	4.0	March 2010	#version 400
4.10.6 ^[8]	4.1	July 2010	#version 410
4.20.11 ^[9]	4.2	August 2011	#version 420
4.30.8 ^[10]	4.3	August 2012	#version 430
4.40 ^[11]	4.4	July 2013	#version 440
4.50 ^[12]	4.5	August 2014	#version 450

编程

在 OpenGL 中，还有一种特殊的变量类型：

- **uniform变量**：用于与“外部”（*main.cpp*）的顶点或片段着色器通信。它们在每次*paintGL*运行中可能会有所不同。在您的着色器中，您使用**uniform**来声明变量。

```
uniform float myVariable;
```

顶点或片段着色器源代码

顶点着色器

- **uniform**变量在着色器代码中是只读的。您只能在 C++（*main.cpp*）

片段着色器

- 例子：

```
# version 430
uniform float Scale;

void main (void)
{
    vec4 a = gl_Vertex;
    a.x = a.x * Scale;
    a.y = a.y * Scale;

    gl_Position = gl_ModelViewProjectionMatrix * a;
}
```

Scale x, y

```
#version 430
uniform vec4 color;

void main (void)
{
    gl_FragColor = color;
}
```

明确指定颜色

编程

在 C++ 中更改Uniform值

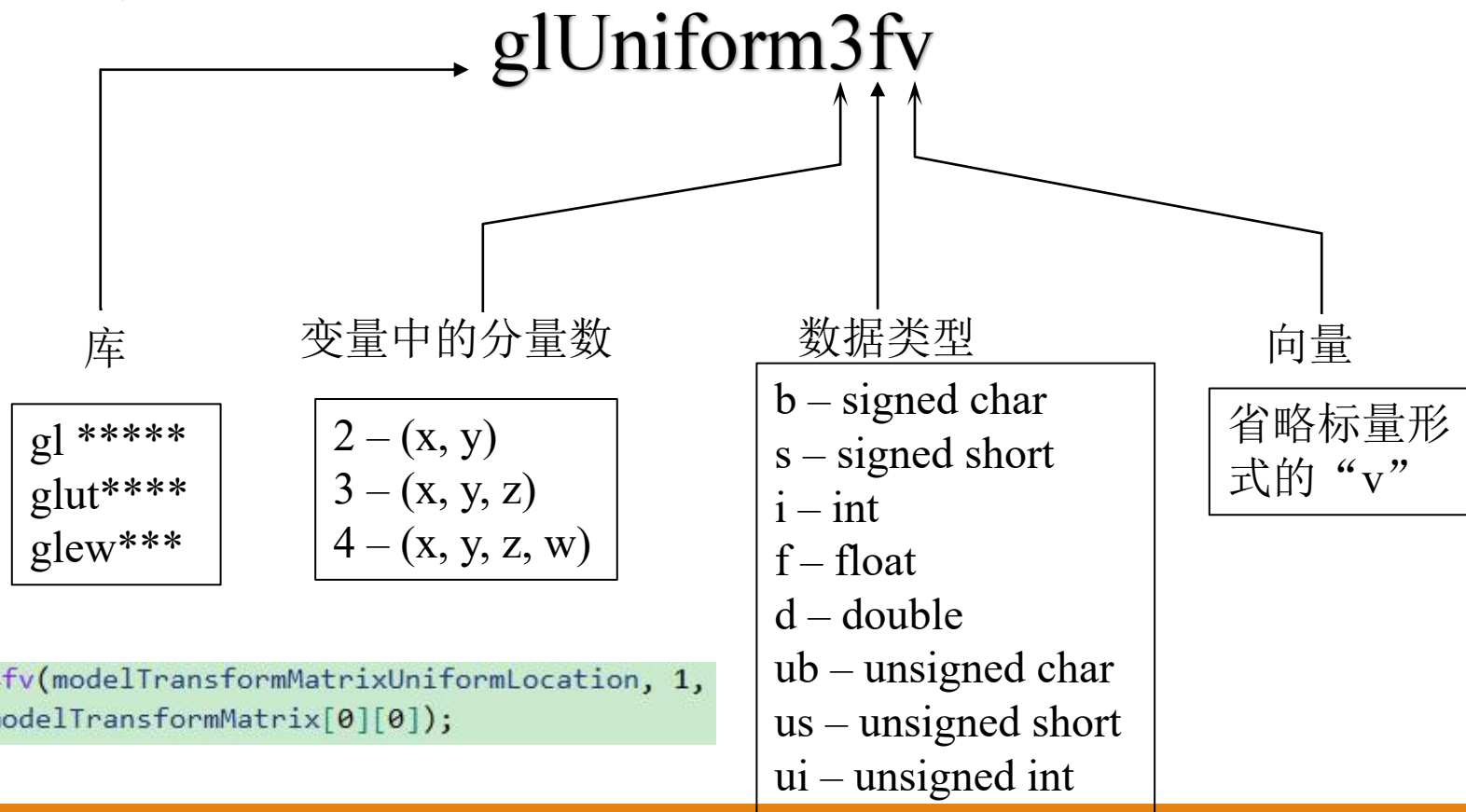
C++ 源代码

```
GLint loc = glGetUniformLocation ( programID , “Scale” );  
if ( loc ! = -1 )  
{  
    glUniform1f (loc, 0.432);  
}
```

- *glGetUniformLocation* : 获取指定程序对象的统一变量的位置（例如，三角形演示代码中的 “*programID*” ）。
- *glUniform1f* : 设置统一变量（1个浮点变量）的值。
- 这是改变变量值的 OpenGL 方式。

编程

OpenGL 函数命名约定：（为当前程序对象指定一个uniform变量的值）



```
glUniformMatrix4fv(modelTransformMatrixUniformLocation, 1,  
GL_FALSE, &modelTransformMatrix[0][0]);
```

编程

Gl Uniform

```
void glUniform1f (GLint location, GLfloat v0);
```

```
void glUniform2f (GLint location, GLfloat v0, GLfloat v1);
```

```
void glUniform3f (...), void glUniform4f (...)
```

```
void glUniform1i (GLint location, GLint v0);
```

```
void glUniform2i (GLint location, GLint v0, GLint v1);
```

```
void glUniform3i (...), void glUniform4i (...)
```

```
void glUniform1fv (GLint location, GLsizei count, const GLfloat *value);
```

```
void glUniform2fv (...), void glUniform3fv (...), ...
```

```
void glUniform2iv (...), void glUniform3iv (...), ...
```


编程

*paintGL ()*函数（一直运行）：

```
void paintGL(void) {  
    // always run  
    glClearColor(0.1f, 0.1f, 0.1f, 1.0f); //specify the background color  
    glClear(GL_COLOR_BUFFER_BIT);  
  
    glm::mat4 modelTransformMatrix = glm::mat4(1.0f);  
    modelTransformMatrix = glm::translate(glm::mat4(1.0f),  
        glm::vec3(x_delta * x_press_num, 0.0f, 0.0f));  
    GLint modelTransformMatrixUniformLocation =  
        glGetUniformLocation(programID, "modelTransformMatrix");  
    glUniformMatrix4fv(modelTransformMatrixUniformLocation, 1,  
        GL_FALSE, &modelTransformMatrix[0][0]);  
  
    glDrawArrays(GL_TRIANGLES, 0, 6); //render primitives from array data  
}
```

没有绑定*vaoID*：因为我们在三角形演示代码中只有一个对象。它在*sendDataToOpenGL*阶段绑定*vaoID*。

编程

键盘功能:

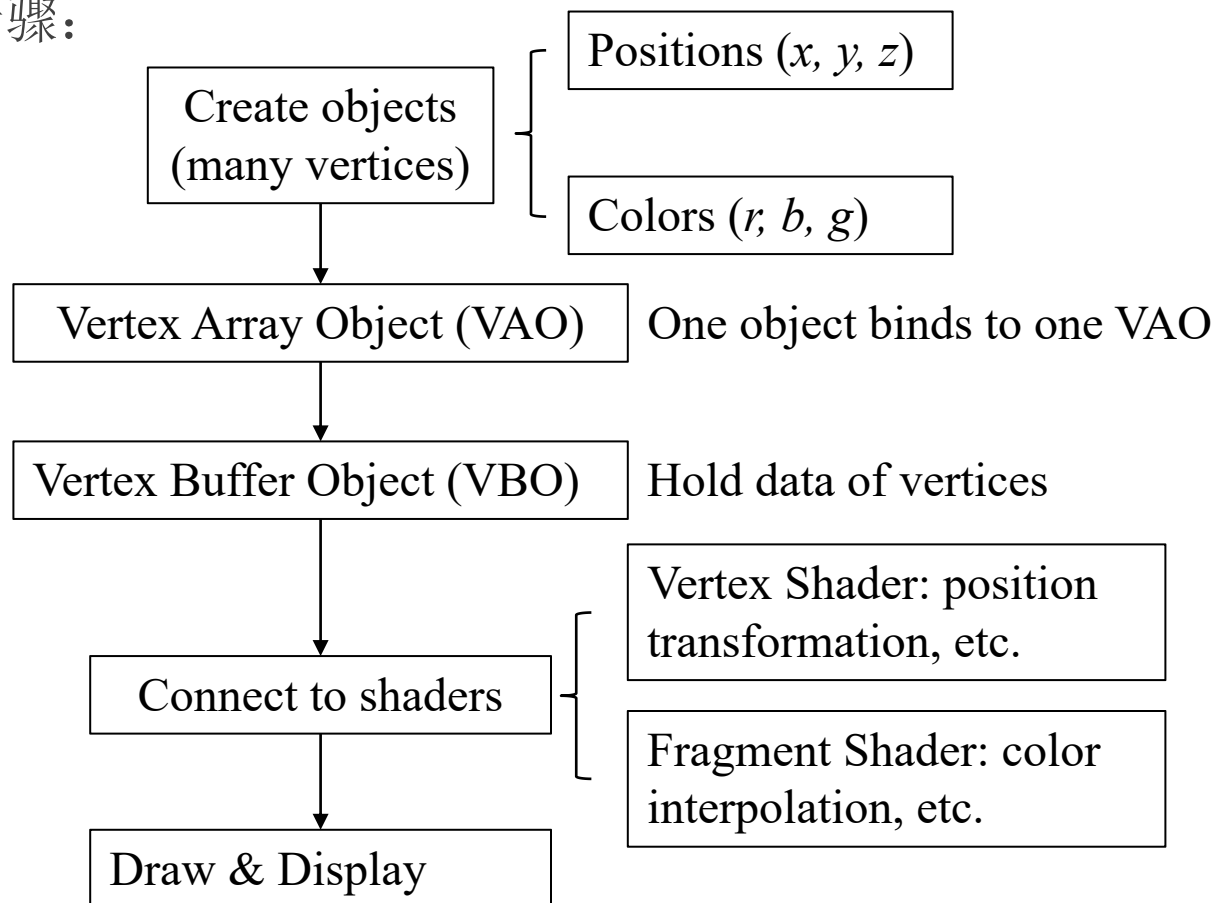
```
void paintGL(void) {  
    // always run  
    glClearColor(0.1f, 0.1f, 0.1f, 1.0f); //specify the background color  
    glClear(GL_COLOR_BUFFER_BIT);  
  
    glm::mat4 modelTransformMatrix = glm::mat4(1.0f);  
    modelTransformMatrix = glm::translate(glm::mat4(1.0f),  
        glm::vec3(x_delta * x_press_num, 0.0f, 0.0f));  
    GLint modelTransformMatrixUniformLocation =  
        glGetUniformLocation(programID, "modelTransformMatrix");  
    glUniformMatrix4fv(modelTransformMatrixUniformLocation,  
        GL_FALSE, &modelTransformMatrix[0][0]);  
  
    glDrawArrays(GL_TRIANGLES, 0, 6); //render primitive  
}
```

```
VertexShaderCode.glsl  FragmentShaderCode.glsl  main.cpp  
1  #version 430  
2  
3  in layout(location=0) vec3 position;  
4  in layout(location=1) vec3 vertexColor;  
5  
6  uniform mat4 modelTransformMatrix;  
7  
8  out vec3 theColor;  
9  
10 void main()  
11 {  
12     vec4 v = vec4(position, 1.0);  
13     vec4 new_position = modelTransformMatrix * v;  
14     gl_Position = new_position;  
15     theColor = vertexColor;  
16 }
```

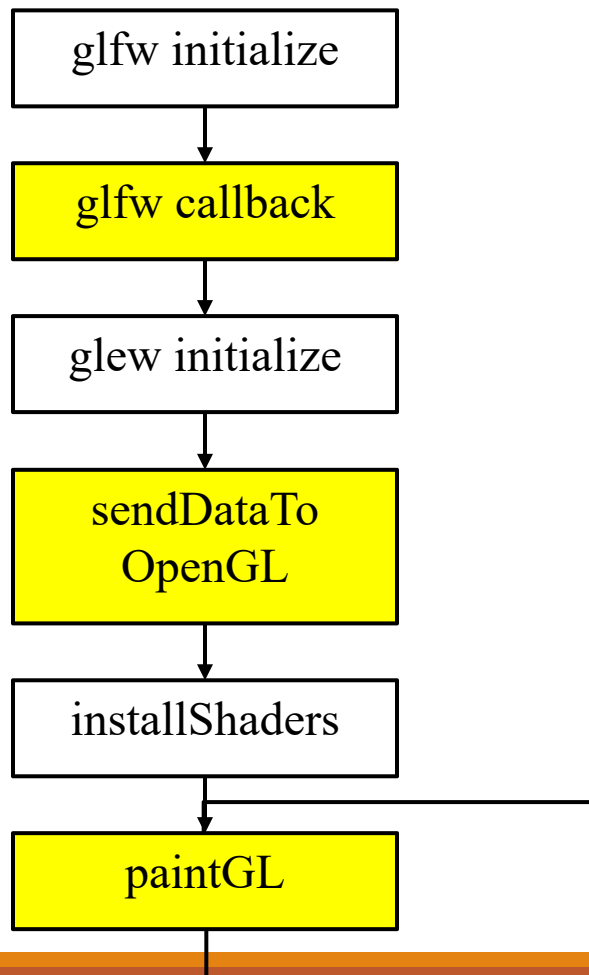
```
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods) {  
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)  
        glfwSetWindowShouldClose(window, true);  
  
    if (key == GLFW_KEY_A && action == GLFW_PRESS) {  
        x_press_num -= 1;  
    }  
    if (key == GLFW_KEY_D && action == GLFW_PRESS) {  
        x_press_num += 1;  
    }  
}
```

编程

渲染对象的主要步骤：



整体组件



- Others: shader code, depth test, etc.

TODO

- 下一教程:

- 作业一简介
- 如何渲染3D 对象