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

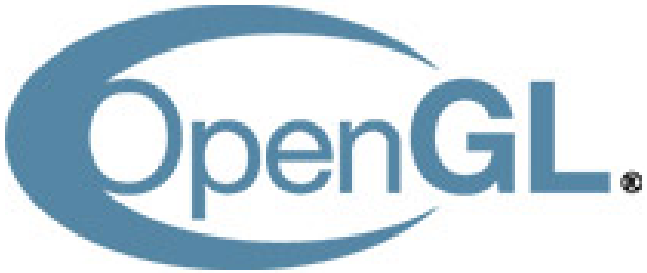
计算机图形学理论和应用

OpenGL 简介

中国科学技术大学，计算机科学与技术学院

2022 年 9 月 15 日

- 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。
- 由一个组织制定并维护的规范 (Specification)。
- OpenGL 规范严格规定了每个函数该如何执行，以及它们的输出值。
- 掌握 C++。



核心模式与立即渲染模式

- OpenGL 编程大致流程
  1. 系统及窗口初始化
  2. 顶点及属性数据的准备
  3. 编辑和加载着色器程序
  4. 绑定或映射数据
  5. 绘制相关图元
- 立即渲染模式，也就是固定渲染管线，开发者很少有控制 OpenGL 如何进行计算的自由。从 OpenGL3.2 开始被废除。
- 核心模式，使用现代的函数，具有更高的灵活性和效率，也更难以学习。
- 差别：核心模式完全基于着色器，每个应用程序必须提供一个顶点着色器和片段着色器，不支持立即渲染模式。
- 使用现代函数要求使用者真正理解 OpenGL 和图形编程。

- OpenGL 自身是一个巨大的状态机 (State Machine)：一系列的变量描述 OpenGL 此刻应当如何运行。
- OpenGL 上下文 (Context)，我们通过设置选项，操作缓冲去改变 OpenGL 状态，最后使用当前 OpenGL 上下文来渲染。

- 一个专门针对 OpenGL 的 C 语言库，它提供了一些渲染物体所需的最低限度的接口。
- 允许用户创建 OpenGL 上下文、定义窗口参数以及处理用户输入。
- 流行的库还有 GLUT、SDL、SFML，可自行选用。

- OpenGL 只是一个标准/规范，具体的实现是由驱动开发商针对特定显卡实现的。
- OpenGL 驱动版本众多，它大多数函数的位置都无法在编译时确定下来，需要在运行时查询。
- 实现对底层 OpenGL 接口封装，管理 OpenGL 的函数指针，可以实现跨平台。

```
// 创建对象
unsigned int objectId = 0;
glGenObject(1, &objectId);
// 绑定对象至上下文
glBindObject(GL_WINDOW_TARGET, objectId);
// 设置当前绑定到 GL_WINDOW_TARGET 的对象的一些选项
glSetObjectOption(GL_WINDOW_TARGET, GL_OPTION_WINDOW_WIDTH, 800);
glSetObjectOption(GL_WINDOW_TARGET, GL_OPTION_WINDOW_HEIGHT, 600);
// 将上下文对象设回默认
glBindObject(GL_WINDOW_TARGET, 0);
```

1. 创建一个对象，然后用一个 id 保存它的引用（实际数据被储存在后台）
2. 将对象绑定至上下文的目标位置（例子中窗口对象目标的位置被定义成 GL_WINDOW_TARGET）
3. 设置窗口的选项
4. 将目标位置的对象 id 设回 0，解绑这个对象

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
```

- 在包含 GLFW 的头文件之前包含 GLAD 的头文件。GLAD 的头文件包含了正确的 OpenGL 头文件（例如 GL/gl.h），所以需要在其它依赖于 OpenGL 的头文件之前包含 GLAD。

```
int main()
{
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    //glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);

    return 0;
}
```

- 初始化 GLFW，使用 OpenGL 3.3，核心模式。

创建窗口


```
GLFWwindow* window = glfwCreateWindow(800, 600, "LearnOpenGL", NULL, NULL);
if (window == NULL)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();
    return -1;
}
glfwMakeContextCurrent(window);
```

- 创建一个窗口对象，glfwCreateWindow 返回一个 GLFWwindow 对象。
- 通知 GLFW 将我们窗口的上下文设置为当前线程的主上下文。

```
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
    std::cout << "Failed to initialize GLAD" << std::endl;
    return -1;
}
```

- 初始化 GLAD，根据我们编译的系统定义正确的函数。

```
glViewport(0, 0, 800, 600);
```

- 设置视口，前两个参数控制视口左下角的位置，第三个和第四个参数控制视口的宽度和高度（像素）。
- 也就是 OpenGL 渲染窗口的大小。(-1, 1)->(0, 800)

```
void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
  glViewport(0, 0, width, height);
}
```

- 定义一个回调函数，调整视口。

```
glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
```

- 注册窗口监听函数，告诉 GLFW 我们希望每当窗口调整大小的时候调用这个函数。

```
void processInput(GLFWwindow *window)
{
  if(glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
      glfwSetWindowShouldClose(window, true);
}
```

- glfwGetKey 检测按键是否被按下。
- 如果被按下了，把 WindowShouldClose 属性设置为 true 从而关闭 GLFW。

```
glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
```

- glClearColor 来设置清空屏幕所用的颜色。
- 当调用 glClear 函数后，整个颜色缓冲都会被填充为 glClearColor 里所设置的颜色。

渲染循环


```
while (!glfwWindowShouldClose(window))
{
    processInput(window);

    glClearColor(25.0 / 255.0, 25.0 / 255.0, 25.0 / 255.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);

    glfwSwapBuffers(window);
    glfwPollEvents();
}
```

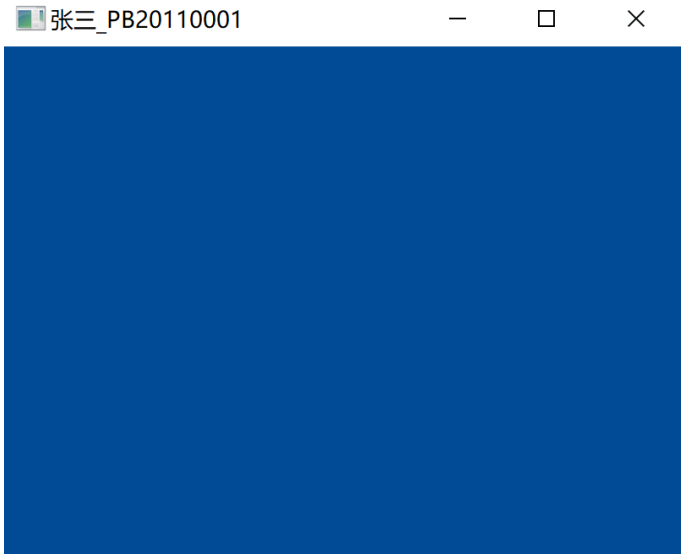
- glfwWindowShouldClose 函数在我们每次循环的开始前检查一次 GLFW 是否被要求退出，如果是的话该函数返回 true 然后渲染循环便结束了。
- glfwSwapBuffers 函数会交换颜色缓冲，它在这一迭代中被用来绘制，并且将会作为输出显示在屏幕上。
- glfwPollEvents 函数检查有没有触发什么事件（比如键盘输入、鼠标移动等）、更新窗口状态，并调用对应的回调函数（可以通过回调方法手动设置）。

释放资源

```
glfwTerminate();
return 0;
```

- 渲染循环结束后我们需要正确释放/删除之前的分配的所有资源。

1. 配置 OpenGL 实验环境。
2. 修改 HelloWindow.cpp，使得窗口标题为" 姓名 _ 学号"，如" 张三 _PB20110001"。
3. 修改窗口背景颜色为科大蓝 (1, 75, 150)。



谢谢!

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