# 1.HELLO WINDOW

## 1.引用库文件

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

注意glad有两层引用

## 2.创建MAIN函数：初始化GLFW

```
int main()
{
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
}
```

`glfwInit`来**初始化**GLFW
`glfwInit`**配置**GLFW

- **第一个参数代表选项的名称，可以从很多以`GLFW_`开头的枚举值中选择；**
- **第二个参数接受一个整型，用来设置这个选项的值。**
  - `GLFW_CONTEXT_VERSION_MAJOR`、`GLFW_CONTEXT_VERSION_MINOR`将主版本号(Major)和次版本号(Minor)都设为3。
  - `GLFW_OPENGL_PROFILE`告诉GLFW我们使用的是核心模式(Core-profile)

### MAC适配

## 3.MAIN函数：创建窗口对象

这个窗口对象存放了**所有和窗口相关的数据**
会被GLFW的其他函数频繁地用到

```
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);
```

`glfwCreateWindow`

- **参数列表**
  - 宽
  - 高
  - 窗口名称
- **返回值**
  - GLFWwindow对象，会在其它的GLFW操作中使用到
    `glfwMakeContextCurrent`
    创建完窗口**通知GLFW：**
    将我们**窗口的上下文**设置为**当前线程**的**主上下文**

# 4.MAIN函数：初始化GLAD

因为下一步需要使用opengl函数，所以需要初始化GLAD

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

GLFW给我们的是`glfwGetProcAddress`，它根据我们编译的系统定义了正确的函数
将其转换为`GLADloadproc`
再传给`gladLoadGLLoader`

# 5.MAIN函数：设置VIEWPORT

必须告诉**OpenGL渲染窗口的尺寸大小**，即**视口(Viewport)**，
OpenGL才能知道怎样根据**窗口大小显示数据和坐标**

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

`glViewport`参数

- 前两个参数控制**窗口左下角的位置**
- 第三个和第四个参数控制渲染窗口的**宽度和高度（像素）**

## GLVIEWPORT的坐标映射

| OpenGL中的位置坐标 | 屏幕坐标 |
| --- | --- |
| (-1到1)和(-1到1) | 自定义，此处为(0, 800)和(0, 600) |

### GLFW视口大小和OPENGL视口大小

实际上也可以**将视口的维度**设置为**比GLFW的维度小，**
这样子之后所有的OpenGL渲染将**会在一个更小的窗口中显示，**
这样子的话我们可以**将一些其它元素显示在OpenGL视口之外**

## 处理用户输入（CALLBACK函数实现）

当用户改变窗口的大小的时，视口也应该被调整
对窗口注册一个回调函数(Callback Function)，会在每次窗口大小被调整的时候被调用

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

GLFWwindow作为它的第一个参数，以及两个整数表示窗口的新维度。

### 注册函数

**我们会在创建窗口之后，渲染循环初始化之前注册这些回调函数。**

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

**注册完成后，每当窗口改变大小，GLFW会调用这个函数并填充相应的参数**
当窗口**被第一次显示的时候，**`framebuffer_size_callback`**也会被调用**

我们还可以将我们的函数注册到其它很多的回调函数中。比如说，我们可以创建一个回调函数来处理手柄输入变化，处理错误消息等。

# 6.MAIN函数：渲染循环

希望程序在我们主动关闭它之前**不断绘制图像并能够接受用户输入**

```
while(!glfwWindowShouldClose(window))
{
     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，该函数返回`true`，渲染循环将停止运行，之后我们就可以关闭应用程序。

- `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，**
  - **调用对应的回调函数（可以通过回调方法手动设置）。**
- `glfwSwapBuffers`函数会**交换颜色缓冲**
  - 它是**一个储存着GLFW窗口每一个像素颜色值的大缓冲**
  - 它在这一loop中**被用来绘制**，并且将会**作为输出显示在屏幕上**。

## 渲染指令

```
// 渲染循环
while(!glfwWindowShouldClose(window))
{
     // 输入
     processInput(window);

     // 渲染指令
     ...

     // 检查并调用事件，交换缓冲
     glfwPollEvents();
     glfwSwapBuffers(window);
}
```

**渲染指令在输入之后，检查和swap之前**

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

glClearColor函数是一个**状态设置**函数

- **清空屏幕所用的颜色，RGBA**
  glClear函数则是一个**状态使用**的函数
- 新的渲染迭代开始的时候我们总是希望清屏，否则我们仍能看见上一次迭代的渲染结果
- 使用了**当前的状态来获取应该清除为的颜色**
- **缓冲位**有
  - GL_COLOR_BUFFER_BIT，
  - GL_DEPTH_BUFFER_BIT
  - GL_STENCIL_BUFFER_BIT
    **当调用glClear函数，清除颜色缓冲之后，整个颜色缓冲都会被填充为glClearColor里所设置的颜色**

## 双缓冲(DOUBLE BUFFER)

使用**单缓冲**绘图时可能会存在**图像闪烁**的问题。
这是因为生成的**图像不是一下子被绘制**出来的，
而是按照**从左到右，由上而下逐像素地绘制**而成的。
最终图像**不是在瞬间**显示给用户，而是通过**一步一步生成**的，
这会导致渲染的结果很不真实。

| **场景角色** | **场景角色** |
| --- | --- |
| 观众看到的画面 | 直接盯着 “正在画的画板” |
| 你（程序）的操作 | 拿着画笔在 “观众盯着的画板” 上一点点画 |
| 最终呈现 | 观众能看到你 “画一笔、显一笔”（比如先画轮廓、再填色，过程中画面是残缺的），这就是 “闪烁” |

为了规避这些问题，我们应用双缓冲渲染窗口应用程序。
**前**缓冲保存着**最终输出的图像**，它会**在屏幕上显示**；
而所有的的**渲染指令**都会在**后**缓冲上绘制。
当所有的渲染指令执行完毕后，我们**交换**(Swap)前缓冲和后缓冲，这样图像就立即呈显出来，之前提到的不真实感就消除了。

| **对比维度** | **前缓冲（Front Buffer）** | **后缓冲（Back Buffer）** |
| --- | --- | --- |
| 核心角色 | 展示端（给用户看） | 创作端（程序绘图） |
| 存储内容 | 当前屏幕显示的完整图像 | 程序正在绘制的下一帧完整图像 |
| 是否允许修改 | 不允许（只读，供屏幕读取） | 允许（只写，接收程序绘图指令） |
| 关键动作 | 被 “交换” 替换 | 画完后 “交换” 成为新的前缓冲 |

## 7.MAIN函数：释放/删除之前的分配的所有资源

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

这样便能清理所有的资源并正确地退出应用程序。