OpenGL

# OpenGL是什么，能为我们做什么？

OpenGL（Open Graphics Library）是一个跨平台、跨语言的开放式图形编程接口（API）标准。它用于渲染2D和3D矢量图形，通过与计算机的GPU（图形处理器）通信，实现硬件加速的图形渲染。简单来说，它是一套规范，定义了软件如何高效地命令显卡绘制复杂图形。

**OpenGL 只定义函数名、参数和功能标准，具体实现由显卡厂商（如NVIDIA/AMD）在驱动程序中完成。**开发者无需了解不同GPU的机器指令（如NVIDIA PTX或AMD GCN），只需调用统一的OpenGL API，驱动会将其“翻译”成当前硬件的可执行代码。

OpenGL只是一个允许我们控制显卡的规范

## 现代OpenGL的着色器：

OpenGL 的着色器（Shader）是用特殊的编程语言（GLSL，OpenGL Shading Language）编写的小型程序，它们直接在GPU上运行，负责控制图形渲染管线中的关键计算步骤。你可以将着色器理解为“注入到渲染流程中的自定义算法”，用于实现顶点变换、光照、颜色、纹理等效果的精细控制。

### 为什么需要着色器？

在早期OpenGL（固定管线时代），渲染效果（如光照、雾效）是由预定义的固定函数控制的，开发者只能通过参数调整，灵活性极低。

而现代OpenGL（可编程管线）将核心计算过程开放给开发者，通过编写着色器，你可以：

* 完全掌控渲染的数学与物理过程
* 实现任意复杂的视觉效果（如卡通渲染、水波纹、动态光影）
* 最大化发挥GPU并行计算能力

现代OpenGL必须使用着色器：固定管线已在OpenGL 3.1+中被废弃。

OpenGL 的着色器（Shader） 是用特殊的编程语言（GLSL，OpenGL Shading Language）编写的小型程序，**它们直接在 GPU 上运行**（普通编程语言如C++等是在CPU运行），负责控制图形渲染管线中的关键计算步骤。你可以将着色器理解为“注入到渲染流程中的自定义算法”，用于实现顶点变换、光照、颜色、纹理等效果的精细控制。

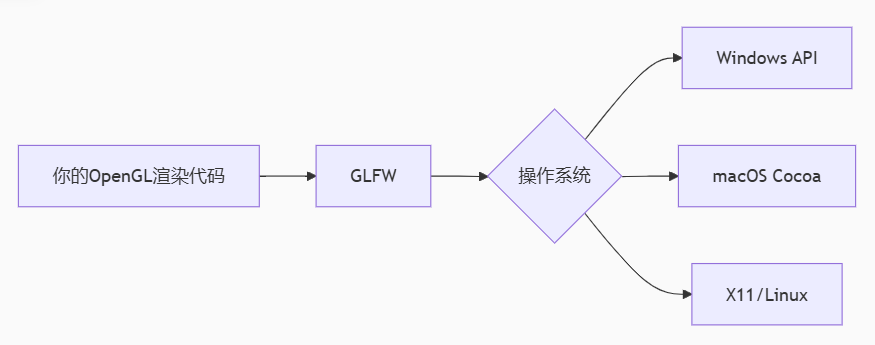
# OpenGL代码创建一个窗口并在窗口画一个三角形（GLFW库）

## GLFW是什么？

GLFW（OpenGL Framework）是一个轻量级、开源的跨平台库，专门用于创建和管理OpenGL渲染上下文所需的窗口、输入和事件系统。你可以将它理解为OpenGL的“启动器”和“控制台”——它不参与图形渲染本身，但为OpenGL提供了一个运行环境。

OpenGL本身只关注图形渲染，但渲染需要基础环境：

* ❌ 无法创建窗口或处理操作系统交互
* ❌ 无法读取键盘鼠标输入
* ❌ 无法管理窗口事件（如缩放、关闭）





GLFW 本质上是位于 OpenGL 之上的一层抽象 API，但它不参与图形渲染本身，而是专注于为 OpenGL（或 Vulkan/OpenGL ES）提供运行环境的基础设施。



类比：



GLFW 是 OpenGL 的“服务层”，它提供渲染所需的窗口环境，而 OpenGL 专注图形计算。两者协同工作，但各司其职。

1. 底层：操作系统与硬件

* 操作系统（Windows/macOS/Linux）：提供原生图形接口（如WGL/GLX/Cocoa）。
* 显卡驱动：实现 OpenGL 规范，将 API 调用转为 GPU 指令。
* GPU：执行实际渲染计算。

2. 中层：OpenGL API

图形渲染的核心层（负责绘制三角形、纹理贴图、着色器等）。

3. 上层：GLFW

角色：OpenGL 的“服务提供者”，解决以下问题：

* ✅ 窗口创建：生成一个支持 OpenGL 渲染的窗口。
* ✅ 上下文管理：绑定 OpenGL 与当前窗口。
* ✅ 输入处理：转换键盘/鼠标动作为可读事件。
* ✅ 事件循环：管理窗口缩放、关闭等系统事件。

## 下载GLFW

<https://www.glfw.org/>

自己不想编译看源码就下载官方编译好的文件。

## VS2022链接GLFW

### 静态链接和动态链接的区别

* ‌静态链接‌：在编译阶段完成，库代码（如.a或.lib文件）被直接合并到可执行文件中。‌‌‌‌
* ‌动态链接‌：在程序运行时加载共享库（如.so或.dll），仅记录库的引用信息。‌‌

‌文件与内存特性‌：

静态链接：可执行文件较大（包含所有库代码）。内存占用高（多程序重复加载相同库）

动态链接：可执行文件较小（仅含引用信息）。内存占用低（多程序共享同一库副本）。

维护与更新：

静态链接：库更新需重新编译程序，维护成本高。

动态链接：库可独立更新，无需重新编译程序，但可能引发版本冲突（如“DLL地狱”）。

适用场景：

静态链接：嵌入式系统、独立分发程序或对启动速度要求高的场景。

动态链接：大型软件、插件化架构或需频繁更新库的场景。??

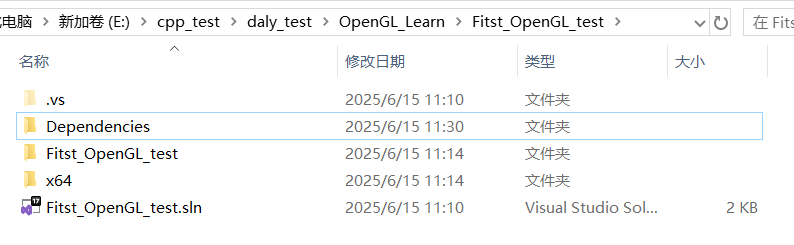
性能差异

静态链接启动更快（无库加载开销），但现代操作系统对动态库有缓存机制，实际差异可能不明显。

### 我们采用静态链接

1.拷贝GLFW官网的Documentation的示例文件，然后粘贴到自己的vs项目中。

2.打开自己VS项目的方案文件路径，创建一个Dependencies文件夹：

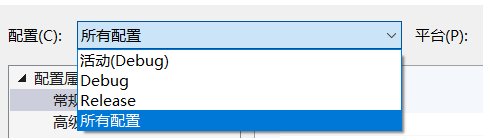


在Dependeices里面创建一个文件夹GLFW，把之前下载的压缩包中的include和lib-vc2015粘贴到GLFW文件夹内：

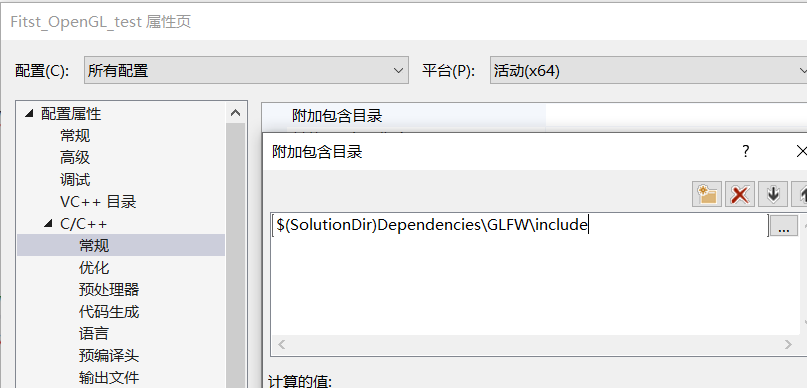


3.让自己的项目的“包含目录”包含include文件夹：

右键项目，属性，先选择所有配置：

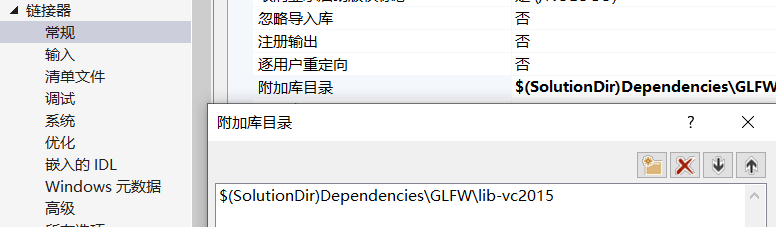


在C/C++，常规的附加包含目录输入：$(SolutionDir) 代表着项目的根目录，然后输入include的相对路径，这样可以在其他电脑上也能使用这个路径：

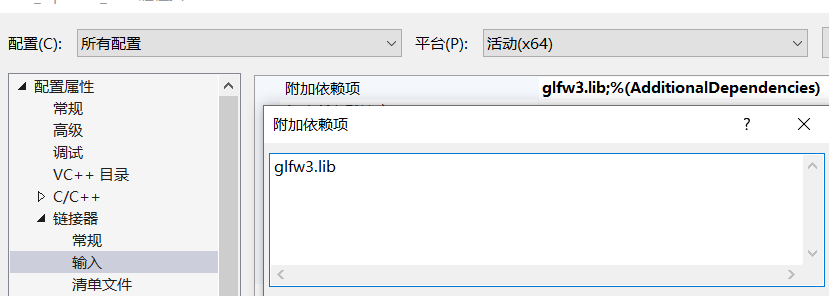


4.让自己项目的“库目录”包含lib文件：

在链接器，常规，附加库目录：包含lib-vc2015：

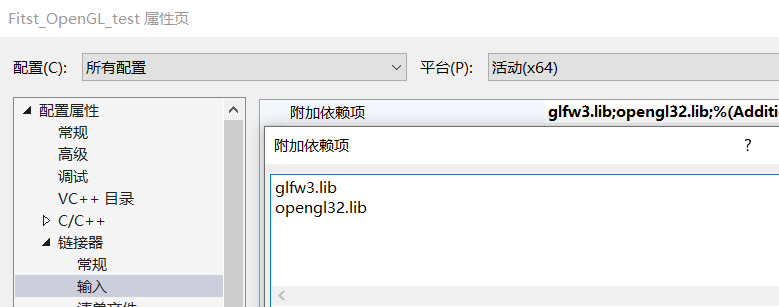


5.在链接器，输入，附加依赖项中添加： glfw3.lib

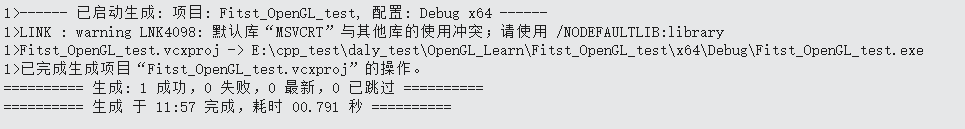


6.此时按f5，发现编译成功但是链接失败。是因为我们在实例中调用了OpenGL的函数，但是没链接OpenGL：

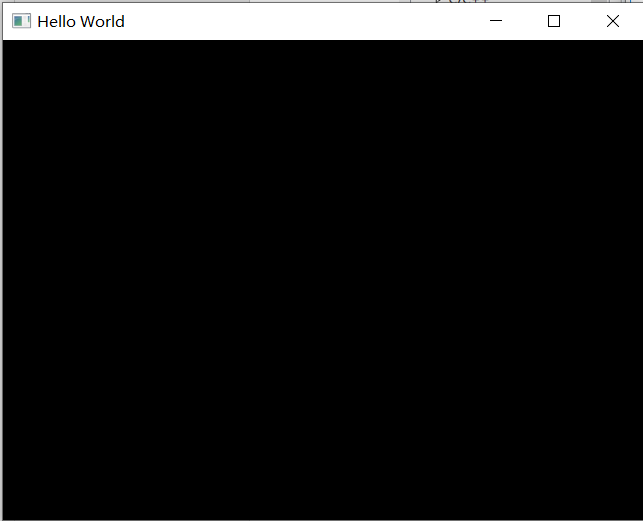
再次在链接器，输入，附加依赖项中添加： opengl32.lib



7.右键项目生成，可以看到，项目已经生成成功了：

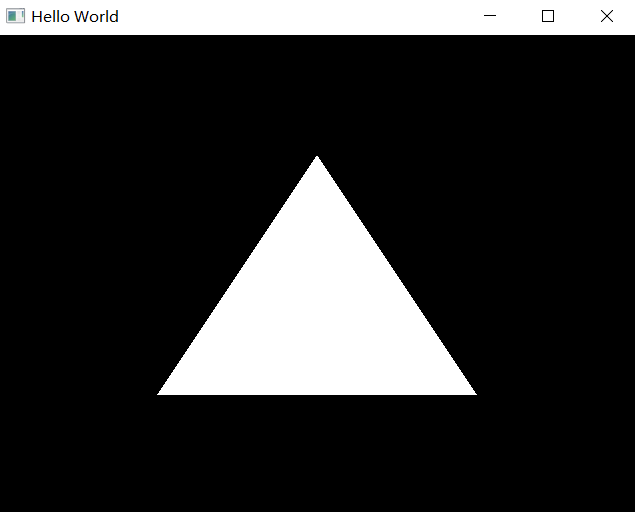
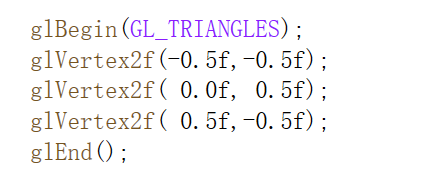


8.点击运行，可以看到生成了一个黑窗口：



## 在之前的黑窗口上创建一个三角形（GLFW库实现）

在代码中添加：



就会在之前画的黑色窗口中显示一个三角形，坐标为代码所示。

原点（0, 0）位于屏幕的中心。

* x 坐标的范围是从 -1 到 1，表示屏幕的水平方向。-1 表示屏幕的最左边，1 表示屏幕的最右边。
* y 坐标的范围是从 -1 到 1，表示屏幕的垂直方向。-1 表示屏幕的最下边，1 表示屏幕的最上边。

# GLEW库

## GLEW介绍

### GLEW产生的背景

GLEW（OpenGL Extension Wrangler Library）是用于处理OpenGL扩展的库。

OpenGL 是一个跨语言、跨平台的编程接口，用于渲染 2D、3D 矢量图形。随着图形硬件的发展，出现了许多新的 OpenGL 扩展功能。这些扩展功能允许开发者利用新的图形硬件特性来实现更复杂的图形效果。然而，不同厂商的显卡支持的 OpenGL 扩展可能不同，并且不同版本的 OpenGL 也有差异。GLEW帮助开发者处理这些复杂的扩展情况。

### GLEW工作原理

GLEW 会在程序运行时自动检测当前使用的 OpenGL 驱动程序所支持的扩展。它提供了一组易用的函数和宏定义，使得开发者可以方便地访问这些扩展功能。例如，对于每个 OpenGL 扩展函数，GLEW 会生成一个函数指针，并通过检测机制来确定该函数是否可用。如果可用，开发者就可以像使用普通的 OpenGL 函数一样来调用这些扩展函数。

### GLEW优点

* 自动检测 ：减少了开发者手动检查 OpenGL 扩展支持情况的工作量。在没有 GLEW 的情况下，开发者需要自己编写代码来判断显卡是否支持某个特定的扩展，这既繁琐又容易出错。而 GLEW 可以自动完成这个检测过程，提高了开发效率。
* 跨平台支持 ：在 Windows、Linux 和 macOS 等多个操作系统平台上都可以使用 GLEW。这使得开发者能够更容易地编写跨平台的图形应用程序，不用担心不同平台下 OpenGL 扩展处理的不同之处。
* 易于使用 ：它提供了一个简单易懂的接口。对于大多数扩展函数，开发者只需要包含 GLEW 的头文件并正确初始化，就可以立即使用这些函数。

### 典型应用场景

* 在游戏开发中，游戏通常需要利用各种高级的图形特效，这些特效往往依赖于 OpenGL 扩展。GLEW 可以帮助游戏开发者快速地集成这些扩展功能，如阴影贴图、高级纹理等功能，以提高游戏的视觉质量。
* 对于图形渲染软件，如 3D 建模工具和科学可视化软件，GLEW 也非常重要。这些软件需要充分利用现代显卡的硬件加速能力来渲染复杂的图形场景，GLEW 使得软件能够方便地访问各种 OpenGL 扩展来实现高效、高质量的图形渲染。

### GLEW 的核心功能

GLEW的核心使命是简化现代OpenGL函数指针的加载过程。

|  |  |
| --- | --- |
| 功能 | 说明 |
| 自动加载函数指针 | 一次性加载OpenGL 1.0到最新版（如4.6）的所有标准函数 |
| 扩展支持检测 | 检查硬件是否支持特定扩展（如 if(GLEW\_ARB\_geometry\_shader4) ） |
| 跨平台兼容 | 支持Windows/macOS/Linux（注意：macOS需额外处理） |
| 轻量级 | 仅头文件 glew.h + 源文件 glew.c（或预编译库），无复杂依赖 |
| 错误处理 | 明确返回初始化状态，避免崩溃 |

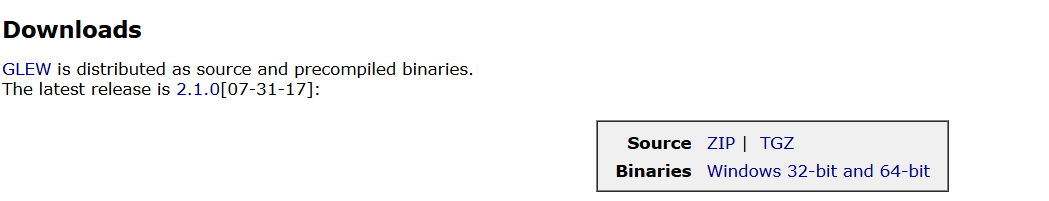
GLEW 是OpenGL开发的“钥匙扣”：

它把散落在显卡驱动中的函数指针（钥匙）统一收集管理，让你无需手动寻找即可直接调用 gl\* 系列函数。虽然现代替代品（如GLAD）更灵活，但理解GLEW仍能加深你对OpenGL底层机制的认识。

## 下载GLEW

访问GLEW官网：<https://glew.sourceforge.net/>进行下载。

如果不想看源码，直接下载二进制Binaries版本：



## VS2022链接GLEW

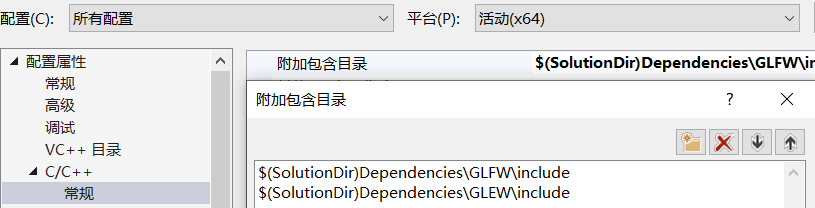
### 和之前链接GLFW一样，我们采用静态链接

1.把下载好的文件解压，也放在Dependencies文件夹内，改名为GLEW：



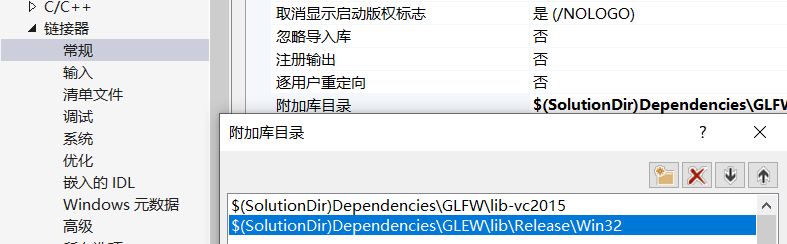
doc文件夹内包含了需要的各种文档。

2.同样在C/C++，常规，附加包含目录中加入include的相对路径：

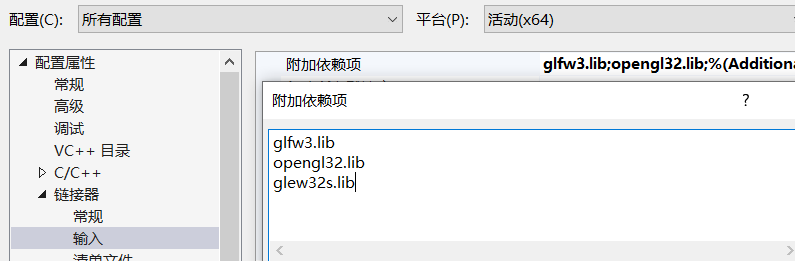


在链接器，常规，附加库目录中加入glew32s.lib的相对路径：

注意：32位在Win32文件夹，64位在x86文件夹



3.在链接器，输入中加入glew32s.lib：



至此就配置完成了。

### 使用现代OpenGL的函数

1.添加头文件：

为什么要GL/？因为我们的包含目录到glew.h之间还有一个GL文件夹。

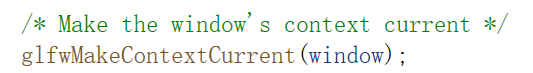
2.初始化glew：

但是在我们运行之后，发现报错。这是因为头文件glew.h要放在glfw3.h之前。所以我们要把glew.h放在所有OpenGL头文件之前。

3.这样右键生成，会发现编译没错了。但是有链接错误。在C/C++，预处理器，预处理器定义中添加：GLEW\_STATIC



自此点击生成成功生成。

注意：使用要先初始化OpenGL上下文，也就是该函数要放在后面。

我们使用GLEW就可以访问所有的OpenGL函数了，可以使用任何版本。我们浏览GL/glew.h文件中的内容，就会发现它里面全是OpenGL的函数指针。

### 打印OpenGL当前版本

在初始化上OpenGL下文之后就可以打印OpenGL当前版本：

std::cout << glGetString(GL\_VERSION) << std::endl;



