# 欢迎来到OpenGL的世界

欢迎来到OpenGL的世界。这个工程只是我([Joey de Vries](http://joeydevries.com/))的一次小小的尝试，希望能够建立起一个完善的OpenGL教学平台。无论你学习OpenGL是为了学业，找工作，或仅仅是因为兴趣，这个网站都将能够教会你**现代**(Core-profile) OpenGL从基础，中级，到高级的知识。LearnOpenGL的目标是使用易于理解的形式，使用清晰的例子，展现现代OpenGL的所有知识点，并与此同时为你以后的学习提供有用的参考。

如果您喜欢这个系列教程的话，不妨向Joey de Vries的[Paypal](https://www.paypal.me/learnopengl/)进行捐赠，支持一下作者，让这个教程能够持续完善并更新。

## 为什么要阅读这些教程呢？

在互联网上，有关学习OpenGL的有成千上万的文档与资源，然而其中大部分的资源仅仅讨论了OpenGL的立即渲染模式（Immediate Mode，通常会说旧OpenGL），亦或是不完整，缺少适当的文档，甚至是仅仅不适合你的口味。所以，我的目标是提供一个既完整，又易懂的平台供人们学习。

如果你很享受那些提供手把手指导的教程，那些提供清晰例子的教程，以及那些不会一下将你淹没在细节中的教程，那么我的这些教程很可能就很适合你。我的教程旨在让那些没有图形编程经验的人们能够理解，又让那些有经验的读者有阅读下去的兴趣。我的教程同样也讨论了一些常用的概念，只需要你再有一点创造力，就能将你的想法变成真正的3D程序。如果如果你觉得前面这些讲的都是你，欢迎继续阅读我的教程。

## 你将学会什么呢？

我这些教程的核心是现代OpenGL。学习（和使用）现代OpenGL需要用户对图形编程以及OpenGL的幕后运作有非常好的理解才能在编程中有很好的发挥。所以，我们会首先讨论核心的图形学概念，OpenGL怎样将像素绘制到屏幕上，以及如何利用黑科技做出一些很酷的效果。

除了核心概念之外，我们还会讨论许多有用的技巧，它们都可以用在你的程序中，比如说在场景中移动，做出漂亮的光照，加载建模软件导出的自定义模型，做一些很酷的后期处理技巧等。最后，我们也将会使用我们已学的知识从头开始做一个小游戏，让你真正体验一把图形编程的魅力。

## 关于中文翻译

这里是LearnOpenGL教程的中文翻译，英文版的地址为：<https://learnopengl.com/>

由于翻译可能无法做到精确表达原文意思，我们推荐您在对问题有疑惑的时候去阅读一下英文版的教程。如果您对翻译有更好的建议，可以去我们的[GitHub工程](https://github.com/LearnOpenGL-CN/LearnOpenGL-CN)上提交Issue或者Pull Request。如果是对教程的内容有问题，请先查看原文，如果不是翻译错误的话，请直接在原网站评论区向作者（JoeyDeVries）反馈。

如果教程中的源码无法打开的话，可以到教程的[GitHub页面](https://github.com/JoeyDeVries/LearnOpenGL)上去寻找所需的代码，每一节教程的源码以及练习都位于 src 目录下的对应章节目录中。

# 简介

你到这里来可能是想学习计算机图形的工作原理，并且自己做一些很酷的东西。自己做东西是非常有趣的，同样也能给你带来对图形编程的兴趣。然而，在你开始学习旅程之前，有这么几点注意事项。

## 前置知识

由于OpenGL是一个图形API，并不是一个独立的平台，它需要一个编程语言来工作，在这里我们使用的是C++。所以，对C++的熟练掌握在学习这个教程中是必不可少的。当然，我仍将尝试解释大部分用到的概念，包括一些高级的C++话题，所以，你并不一定要是一个C++专家才能来学习。不过，请确保你至少应该能写个比 ‘Hello World’ 复杂的程序。如果你对C++不是很熟悉，我推荐您学习一下[www.learncpp.com](http://www.learncpp.com/)上的免费教程。

除此之外，我们也将用到一些数学知识（线性代数、几何、三角学），同样我也会尝试解释所有的必备的数学概念。但是，毕竟我不是一个数学家，即使我的解释可能会很容易理解，但是这些解释都是不全面的。所以，在必须的时候我会链接一些不错的资料，它们会将这些概念解释的更加全面。不要被必须的数学知识吓到了，几乎所有的概念只要有基础的数学背景都可以理解。我也会将数学的内容压缩至极限。大部分的功能甚至都不需要你理解所有的数学知识，只要你会使用就行。

## 结构

LearnOpenGL被分解成了许多大的主题。每个主题包括一些小节，每个小节中会对不同的概念进行详细的解释。所有的主题都可以在目录中找到。这些主题是按照线性来学习的（所以建议从上到下来读，除非有特殊指示），每个页面将会解释每个概念的背景理论和实际操作。

为了让教程更容易理解，结构更鲜明，本站采用了**方框**和**代码块**。

## 方框

**绿色**方框是一些注释或者是对于OpenGL或讨论主题有用的特性/提示。

**红色**方框是一些警告或者一些你需要特别注意的特性。

**译注**

**蓝色**方框是翻译时为了帮助读者理解附加的一些信息。

## 代码

你在网站中将会看到很多小片的代码，它们将会在下面这样的代码框中：

// 这个方框是代码

由于这样只提供了代码的片段，当需要的时候我会提供个链接到当前工程的源代码的。

## 颜色标记

有一些词语会以不同颜色显示出来，用来表示这些词语有不同的意义：

* 定义：绿色的字是定义，即一个重要的概念或名称，这些词语你能经常见到。
* 程序逻辑：红色的字是函数的名称或者是类名。
* 变量：蓝色的字是变量，包括所有的OpenGL常量。

现在你应该对这个网站的结构有一些了解了，你现在可以进入「入门」这一章，开始你的OpenGL学习生涯吧！

# OpenGL

| 原文 | [OpenGL](http://learnopengl.com/#!Getting-started/OpenGL) |
| --- | --- |
| 作者 | JoeyDeVries |
| 翻译 | gjy\_1992, Krasjet |
| 校对 | 暂未校对 |

在开始这段旅程之前我们先了解一下OpenGL到底是什么。一般它被认为是一个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。然而，OpenGL本身并不是一个API，它仅仅是一个由[Khronos组织](http://www.khronos.org/)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。至于内部具体每个函数是如何实现(Implement)的，将由OpenGL库的开发者自行决定（译注：这里开发者是指编写OpenGL库的人）。因为OpenGL规范并没有规定实现的细节，具体的OpenGL库允许使用不同的实现，只要其功能和结果与规范相匹配（亦即，作为用户不会感受到功能上的差异）。

实际的OpenGL库的开发者通常是显卡的生产商。你购买的显卡所支持的OpenGL版本都为这个系列的显卡专门开发的。当你使用Apple系统的时候，OpenGL库是由Apple自身维护的。在Linux下，有显卡生产商提供的OpenGL库，也有一些爱好者改编的版本。这也意味着任何时候OpenGL库表现的行为与规范规定的不一致时，基本都是库的开发者留下的bug。

由于OpenGL的大多数实现都是由显卡厂商编写的，当产生一个bug时通常可以通过升级显卡驱动来解决。这些驱动会包括你的显卡能支持的最新版本的OpenGL，这也是为什么总是建议你偶尔更新一下显卡驱动。

所有版本的OpenGL规范文档都被公开的寄存在Khronos那里。有兴趣的读者可以找到OpenGL3.3（我们将要使用的版本）的[规范文档](https://www.opengl.org/registry/doc/glspec33.core.20100311.withchanges.pdf)。如果你想深入到OpenGL的细节（只关心函数功能的描述而不是函数的实现），这是个很好的选择。如果你想知道每个函数**具体的**运作方式，这个规范也是一个很棒的参考。

## 核心模式与立即渲染模式

早期的OpenGL使用立即渲染模式（Immediate mode，也就是固定渲染管线），这个模式下绘制图形很方便。OpenGL的大多数功能都被库隐藏起来，开发者很少有控制OpenGL如何进行计算的自由。而开发者迫切希望能有更多的灵活性。随着时间推移，规范越来越灵活，开发者对绘图细节有了更多的掌控。立即渲染模式确实容易使用和理解，但是效率太低。因此从OpenGL3.2开始，规范文档开始废弃立即渲染模式，并鼓励开发者在OpenGL的核心模式(Core-profile)下进行开发，这个分支的规范完全移除了旧的特性。

当使用OpenGL的核心模式时，OpenGL迫使我们使用现代的函数。当我们试图使用一个已废弃的函数时，OpenGL会抛出一个错误并终止绘图。现代函数的优势是更高的灵活性和效率，然而也更难于学习。立即渲染模式从OpenGL**实际**运作中抽象掉了很多细节，因此它在易于学习的同时，也很难让人去把握OpenGL具体是如何运作的。现代函数要求使用者真正理解OpenGL和图形编程，它有一些难度，然而提供了更多的灵活性，更高的效率，更重要的是可以更深入的理解图形编程。

这也是为什么我们的教程面向OpenGL3.3的核心模式。虽然上手更困难，但这份努力是值得的。

现今，更高版本的OpenGL已经发布（写作时最新版本为4.5），你可能会问：既然OpenGL 4.5 都出来了，为什么我们还要学习OpenGL 3.3？答案很简单，所有OpenGL的更高的版本都是在3.3的基础上，引入了额外的功能，并没有改动核心架构。新版本只是引入了一些更有效率或更有用的方式去完成同样的功能。因此，所有的概念和技术在现代OpenGL版本里都保持一致。当你的经验足够，你可以轻松使用来自更高版本OpenGL的新特性。

当使用新版本的OpenGL特性时，只有新一代的显卡能够支持你的应用程序。这也是为什么大多数开发者基于较低版本的OpenGL编写程序，并只提供选项启用新版本的特性。

在有些教程里你会看见更现代的特性，它们同样会以这种红色注释方式标明。

## 扩展

OpenGL的一大特性就是对扩展(Extension)的支持，当一个显卡公司提出一个新特性或者渲染上的大优化，通常会以扩展的方式在驱动中实现。如果一个程序在支持这个扩展的显卡上运行，开发者可以使用这个扩展提供的一些更先进更有效的图形功能。通过这种方式，开发者不必等待一个新的OpenGL规范面世，就可以使用这些新的渲染特性了，只需要简单地检查一下显卡是否支持此扩展。通常，当一个扩展非常流行或者非常有用的时候，它将最终成为未来的OpenGL规范的一部分。

使用扩展的代码大多看上去如下：

if(GL\_ARB\_extension\_name)

{

// 使用硬件支持的全新的现代特性

}

else

{

// 不支持此扩展: 用旧的方式去做

}

使用OpenGL3.3时，我们很少需要使用扩展来完成大多数功能，当需要的时候，本教程将提供适当的指示。

## 状态机

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的状态通常被称为OpenGL上下文(Context)。我们通常使用如下途径去更改OpenGL状态：设置选项，操作缓冲。最后，我们使用当前OpenGL上下文来渲染。

假设当我们想告诉OpenGL去画线段而不是三角形的时候，我们通过改变一些上下文变量来改变OpenGL状态，从而告诉OpenGL如何去绘图。一旦我们改变了OpenGL的状态为绘制线段，下一个绘制命令就会画出线段而不是三角形。

当使用OpenGL的时候，我们会遇到一些状态设置函数(State-changing Function)，这类函数将会改变上下文。以及状态使用函数(State-using Function)，这类函数会根据当前OpenGL的状态执行一些操作。只要你记住OpenGL本质上是个大状态机，就能更容易理解它的大部分特性。

## 对象

OpenGL库是用C语言写的，同时也支持多种语言的派生，但其内核仍是一个C库。由于C的一些语言结构不易被翻译到其它的高级语言，因此OpenGL开发的时候引入了一些抽象层。“对象(Object)”就是其中一个。

在OpenGL中一个对象是指一些选项的集合，它代表OpenGL状态的一个子集。比如，我们可以用一个对象来代表绘图窗口的设置，之后我们就可以设置它的大小、支持的颜色位数等等。可以把对象看做一个C风格的结构体(Struct)：

struct object\_name {

float option1;

int option2;

char[] name;

};

译注: 在更新前的教程中一直使用的都是OpenGL的基本类型，但由于作者觉得在本教程系列中并没有一个必须使用它们的原因，所有的类型都改为了自带类型。但是请仍然记住，使用OpenGL的类型的好处是保证了在各平台中每一种类型的大小都是统一的。你也可以使用其它的定宽类型(Fixed-width Type)来实现这一点。

当我们使用一个对象时，通常看起来像如下一样（把OpenGL上下文看作一个大的结构体）：

// OpenGL的状态

struct OpenGL\_Context {

...

object\* object\_Window\_Target;

...

};

// 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);

这一小段代码展现了你以后使用OpenGL时常见的工作流。我们首先创建一个对象，然后用一个id保存它的引用（实际数据被储存在后台）。然后我们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）。接下来我们设置窗口的选项。最后我们将目标位置的对象id设回0，解绑这个对象。设置的选项将被保存在objectId所引用的对象中，一旦我们重新绑定这个对象到GL\_WINDOW\_TARGET位置，这些选项就会重新生效。

目前提供的示例代码只是OpenGL如何操作的一个大致描述，通过阅读以后的教程你会遇到很多实际的例子。

使用对象的一个好处是在程序中，我们不止可以定义一个对象，并设置它们的选项，每个对象都可以是不同的设置。在我们执行一个使用OpenGL状态的操作的时候，只需要绑定含有需要的设置的对象即可。比如说我们有一些作为3D模型数据（一栋房子或一个人物）的容器对象，在我们想绘制其中任何一个模型的时候，只需绑定一个包含对应模型数据的对象就可以了（当然，我们需要先创建并设置对象的选项）。拥有数个这样的对象允许我们指定多个模型，在想画其中任何一个的时候，直接将对应的对象绑定上去，便不需要再重复设置选项了。

## 让我们开始吧

你现在已经知道一些OpenGL的相关知识了，OpenGL规范和库，OpenGL幕后大致的运作流程，以及OpenGL使用的一些传统技巧。不要担心你还没有完全消化它们，后面的教程我们会仔细地讲解每一个步骤，你会通过足够的例子来真正掌握OpenGL。如果你已经做好了开始下一步的准备，我们可以在[这里](https://learnopengl-cn.github.io/01%20Getting%20started/02%20Creating%20a%20window/)开始创建OpenGL上下文以及我们的第一个窗口了。

## 附加资源

[opengl.org](https://www.opengl.org/)：OpenGL官方网站。

[OpenGL registry](https://www.opengl.org/registry/)：包含OpenGL各版本的规范和扩展。

# 创建窗口

| 原文 | [Creating a window](http://learnopengl.com/#!Getting-started/Creating-a-window) |
| --- | --- |
| 作者 | JoeyDeVries |
| 翻译 | gjy\_1992, Krasjet |
| 校对 | 暂未校对 |

**译注**

注意，由于作者对教程做出了更新，之前本节使用的是GLEW库，但现在改为了使用GLAD库，关于GLEW配置的部分现在已经被修改，但我仍决定将这部分教程保留起来，放到一个历史存档中，如果有需要的话可以到[这里](https://learnopengl-cn.github.io/legacy/)来查看。

在我们画出出色的效果之前，首先要做的就是创建一个OpenGL上下文(Context)和一个用于显示的窗口。然而，这些操作在每个系统上都是不一样的，OpenGL有目的地将这些操作抽象(Abstract)出去。这意味着我们不得不自己处理创建窗口，定义OpenGL上下文以及处理用户输入。

幸运的是，有一些库已经提供了我们所需的功能，其中一部分是特别针对OpenGL的。这些库节省了我们书写操作系统相关代码的时间，提供给我们一个窗口和上下文用来渲染。最流行的几个库有GLUT，SDL，SFML和GLFW。在教程里我们将使用**GLFW**。

## GLFW

GLFW是一个专门针对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。它允许用户创建OpenGL上下文，定义窗口参数以及处理用户输入，这正是我们需要的。

本节和下一节的目标是建立GLFW环境，并保证它恰当地创建OpenGL上下文并显示窗口。这篇教程会一步步从获取、编译、链接GLFW库讲起。我们使用的是Microsoft Visual Studio 2015 IDE（操作过程在新版的Visual Studio都是相同的）。如果你用的不是Visual Studio（或者用的是它的旧版本）请不要担心，大多数IDE上的操作都是类似的。

## 构建GLFW

GLFW可以从它官方网站的[下载页](http://www.glfw.org/download.html)上获取。GLFW已经有针对Visual Studio 2013/2015的预编译的二进制版本和相应的头文件，但是为了完整性我们将从编译源代码开始。所以我们需要下载**源代码包**。

如果你要使用预编译的二进制版本的话，请下载32位的版本而不是64位的（除非你清楚你在做什么）。大部分读者反映64位版本会出现很多奇怪的问题。

下载源码包之后，将其解压并打开。我们只需要里面的这些内容：

* 编译生成的库
* **include**文件夹

从源代码编译库可以保证生成的库是兼容你的操作系统和CPU的，而预编译的二进制文件可能会出现兼容问题（甚至有时候没提供支持你系统的文件）。提供源代码所产生的一个问题在于不是每个人都用相同的IDE开发程序，因而提供的工程/解决方案文件可能和一些人的IDE不兼容。所以人们只能从.c/.cpp和.h/.hpp文件来自己建立工程/解决方案，这是一项枯燥的工作。但因此也诞生了一个叫做CMake的工具。

## CMake

CMake是一个工程文件生成工具。用户可以使用预定义好的CMake脚本，根据自己的选择（像是Visual Studio, Code::Blocks, Eclipse）生成不同IDE的工程文件。这允许我们从GLFW源码里创建一个Visual Studio 2015工程文件，之后进行编译。首先，我们需要从[这里](http://www.cmake.org/cmake/resources/software.html)下载安装CMake。我选择的是Win32安装程序。

当CMake安装成功后，你可以选择从命令行或者GUI启动CMake，由于我不想让事情变得太过复杂，我们选择用GUI。CMake需要一个源代码目录和一个存放编译结果的目标文件目录。源代码目录我们选择GLFW的源代码的根目录，然后我们新建一个 *build* 文件夹，选中作为目标目录。

在设置完源代码目录和目标目录之后，点击**Configure(设置)**按钮，让CMake读取设置和源代码。我们接下来需要选择工程的生成器，由于我们使用的是Visual Studio 2015，我们选择 **Visual Studio 14** 选项（因为Visual Studio 2015的内部版本号是14）。CMake会显示可选的编译选项用来配置最终生成的库。这里我们使用默认设置，并再次点击**Configure(设置)**按钮保存设置。保存之后，点击**Generate(生成)**按钮，生成的工程文件会在你的**build**文件夹中。

编译

在**build**文件夹里可以找到**GLFW.sln**文件，用Visual Studio 2015打开。因为CMake已经配置好了项目，所以我们直接点击**Build Solution(生成解决方案)**按钮，然后编译的库**glfw3.lib**（注意我们用的是第3版）就会出现在**src/Debug**文件夹内。

库生成完毕之后，我们需要让IDE知道库和头文件的位置。有两种方法：

1. 找到IDE或者编译器的**/lib**和**/include**文件夹，添加GLFW的**include**文件夹里的文件到IDE的**/include**文件夹里去。用类似的方法，将**glfw3.lib**添加到**/lib**文件夹里去。虽然这样能工作，但这不是推荐的方式，因为这样会让你很难去管理库和include文件，而且重新安装IDE或编译器可能会导致这些文件丢失。
2. 推荐的方式是建立一个新的目录包含所有的第三方库文件和头文件，并且在你的IDE或编译器中指定这些文件夹。我个人会使用一个单独的文件夹，里面包含**Libs**和**Include**文件夹，在这里存放OpenGL工程用到的所有第三方库和头文件。这样我的所有第三方库都在同一个位置（并且可以共享至多台电脑）。然而这要求你每次新建一个工程时都需要告诉IDE/编译器在哪能找到这些目录。

完成上面步骤后，我们就可以使用GLFW创建我们的第一个OpenGL工程了！

## 我们的第一个工程

首先，打开Visual Studio，创建一个新的项目。如果VS提供了多个选项，选择Visual C++，然后选择**Empty Project(空项目)**（别忘了给你的项目起一个合适的名字）。现在我们终于有一个空的工作空间了，开始创建我们第一个OpenGL程序吧！

## 链接

为了使我们的程序使用GLFW，我们需要把GLFW库链接(Link)进工程。这可以通过在链接器的设置里指定我们要使用**glfw3.lib**来完成，但是由于我们将第三方库放在另外的目录中，我们的工程还不知道在哪寻找这个文件。于是我们首先需要将我们放第三方库的目录添加进设置。

要添加这些目录（需要VS搜索库和include文件的地方），我们首先进入Project Properties(工程属性，在解决方案窗口里右键项目)，然后选择**VC++ Directories(VC++ 目录)**选项卡（如下图）。在下面的两栏添加目录：

这里你可以把自己的目录加进去，让工程知道到哪去搜索。你需要手动把目录加在后面，也可以点击需要的位置字符串，选择选项，之后会出现类似下面这幅图的界面，图是选择**Include Directories(包含目录)**时的界面：

这里可以添加任意多个目录，IDE会从这些目录里寻找头文件。所以只要你将GLFW的**Include**文件夹加进路径中，你就可以使用<GLFW/..>来引用头文件。库文件夹也是一样的。

现在VS可以找到所需的所有文件了。最后需要在**Linker(链接器)**选项卡里的**Input(输入)**选项卡里添加**glfw3.lib**这个文件：

要链接一个库我们必须告诉链接器它的文件名。库名字是**glfw3.lib**，我们把它加到**Additional Dependencies(附加依赖项)**字段中(手动或者使用选项都可以)。这样GLFW在编译的时候就会被链接进来了。除了GLFW之外，你还需要添加一个链接条目链接到OpenGL的库，但是这个库可能因为系统的不同而有一些差别。

## Windows上的OpenGL库

如果你是Windows平台，**opengl32.lib**已经包含在Microsoft SDK里了，它在Visual Studio安装的时候就默认安装了。由于这篇教程用的是VS编译器，并且是在Windows操作系统上，我们只需将**opengl32.lib**添加进连接器设置里就行了。

## Linux上的OpenGL库

在Linux下你需要链接**libGL.so**库文件，这需要添加-lGL到你的链接器设置中。如果找不到这个库你可能需要安装Mesa，NVidia或AMD的开发包，这部分因平台而异（而且我也不熟悉Linux）就不仔细讲解了。

接下来，如果你已经添加GLFW和OpenGL库到连接器设置中，你可以用如下方式添加GLFW头文件：

#include <GLFW\glfw3.h>

对于用GCC编译的Linux用户建议使用这个命令行选项-lGLEW -lglfw3 -lGL -lX11 -lpthread -lXrandr -lXi。没有正确链接相应的库会产生 *undefined reference* (未定义的引用) 这个错误。

GLFW的安装与配置就到此为止。

## GLAD

到这里还没有结束，我们仍然还有一件事要做。因为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。由于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。所以任务就落在了开发者身上，开发者需要在运行时获取函数地址并将其保存在一个函数指针中供以后使用。取得地址的方法[因平台而异](https://www.khronos.org/opengl/wiki/Load_OpenGL_Functions)，在Windows上会是类似这样：

// 定义函数原型

typedef void (\*GL\_GENBUFFERS) (GLsizei, GLuint\*);

// 找到正确的函数并赋值给函数指针

GL\_GENBUFFERS glGenBuffers = (GL\_GENBUFFERS)wglGetProcAddress("glGenBuffers");

// 现在函数可以被正常调用了

GLuint buffer;

glGenBuffers(1, &buffer);

你可以看到代码非常复杂，而且很繁琐，我们需要对每个可能使用的函数都要重复这个过程。幸运的是，有些库能简化此过程，其中**GLAD**是目前最新，也是最流行的库。

## 配置GLAD

GLAD是一个[开源](https://github.com/Dav1dde/glad)的库，它能解决我们上面提到的那个繁琐的问题。GLAD的配置与大多数的开源库有些许的不同，GLAD使用了一个[在线服务](http://glad.dav1d.de/)。在这里我们能够告诉GLAD需要定义的OpenGL版本，并且根据这个版本加载所有相关的OpenGL函数。

打开GLAD的[在线服务](http://glad.dav1d.de/)，将语言(Language)设置为**C/C++**，在API选项中，选择**3.3**以上的OpenGL(gl)版本（我们的教程中将使用3.3版本，但更新的版本也能正常工作）。之后将模式(Profile)设置为**Core**，并且保证**生成加载器**(Generate a loader)的选项是选中的。现在可以先（暂时）忽略拓展(Extensions)中的内容。都选择完之后，点击**生成**(Generate)按钮来生成库文件。

GLAD现在应该提供给你了一个zip压缩文件，包含两个头文件目录，和一个**glad.c**文件。将两个头文件目录（**glad**和**KHR**）复制到你的**Include**文件夹中（或者增加一个额外的项目指向这些目录），并添加**glad.c**文件到你的工程中。

经过前面的这些步骤之后，你就应该可以将以下的指令加到你的文件顶部了：

#include <glad/glad.h>

点击编译按钮应该不会给你提示任何的错误，到这里我们就已经准备好继续学习[下一节](https://learnopengl-cn.github.io/01%20Getting%20started/03%20Hello%20Window/)去真正使用GLFW和GLAD来设置OpenGL上下文并创建一个窗口了。记得确保你的头文件和库文件的目录设置正确，以及链接器里引用的库文件名正确。如果仍然遇到错误，可以先看一下评论有没有人遇到类似的问题，请参考额外资源中的例子或者在下面的评论区提问。

## 附加资源

[GLFW: Window Guide](http://www.glfw.org/docs/latest/window_guide.html)：GLFW官方的配置GLFW窗口的指南。

[Building applications](http://www.opengl-tutorial.org/miscellaneous/building-your-own-c-application/)：提供了很多编译或链接相关的信息和一大列错误及对应的解决方案。

[GLFW with Code::Blocks](http://wiki.codeblocks.org/index.php?title=Using_GLFW_with_Code::Blocks)：使用Code::Blocks IDE编译GLFW。

[Running CMake](http://www.cmake.org/runningcmake/)：简要的介绍如何在Windows和Linux上使用CMake。

[Writing a build system under Linux](http://learnopengl.com/demo/autotools_tutorial.txt)：Wouter Verholst写的一个autotools的教程，讲的是如何在Linux上编写构建系统，尤其是针对这些教程。

[Polytonic/Glitter](https://github.com/Polytonic/Glitter)：一个简单的样板项目，它已经提前配置了所有相关的库；如果你想要很方便地搞到一个LearnOpenGL教程的范例工程，这也是很不错的。