项目开发环境配置和编译

建议你使用IDE来编写本课程的作业，而不是使用Sublime Text或未配置的Visual Studio Code等文本编辑器。许多作业需要非常认真的调试。与文本编辑器相比，IDE让你可以更轻松地进行调试。建议你使用：

- Visual Studio（适用于Windows）
- Visual Studio Code 可以按如下方式配置（仅限 MacOS/Linux）将其配置为功能齐全的 c++ IDE。
- Xcode（MacOS）
- CLion（所有平台），Jetbrains 学生许可证免费

最后，你要构建计算机图形学作业，需要执行以下步骤：

1. 为你的机器设置代码环境（只需做**一次**）
2. 编译并运行作业（**每一项作业都**需要完成）

1. 在你的机器上设置 C++ 和依赖项

注意：你只需执行**一次**这些步骤。

平台：Windows、MacOS、Linux等。

Windows

这是在 Windows 上安装 IDE 和构建项目的视频演示（youtube）。

1. 安装 Visual Studio 2019 或更高版本
   - 安装Visual Studio时选择“使用C++进行桌面开发”
2. 在 Windows 上安装 Git
3. 安装 Vcpkg
   - 在要安装vcpkg的文件夹中打开PowerShell / CMD / Git Bash，（例如 `C:/` ）
   - Vcpkg 是一个包管理软件，用于下载和构建所需的软件依赖项
   - `git clone https://github.com/microsoft/vcpkg`
   - 进入 `vcpkg` 目录
   - `.\bootstrap-vcpkg.bat`
   - vcpkg安装完成后，安装本项目的依赖
     - 在64位系统上（目前的大多数系统） `.\vcpkg.exe install freetype:x64-windows`
     - 在32位系统上（可能无法正常工作） `.\vcpkg.exe install freetype`

苹果系统

1. 打开命令行终端并运行 `xcode-select --install` 。如果出现提示，请接受 XCode 许可证。如果在后续步骤中无法构建，请尝试通过 Mac App Store 安装 XCode。
2. 现在你应该可以 `g++ --version` 顺利运行了。

3. 我们将使用CMake来构建作业。如果你的Mac上没有CMake，你可以先安装HomeBrew（适用于MacOS的第三方管理器），然后运行，轻松安装它 `brew install cmake` 。或者，你可以直接从CMake网站下载CMake。
4. 你还需要安装 freetype 库。一种方便的方法是获取HomeBrew，然后运行 `brew install freetype` 。

常见错误：

- “CXX编译器识别未知”：尝试运行 `sudo xcode-select --reset; sudo xcodebuild -license accept` 。

Linux（Ubuntu/Debian/apt）

使用以下命令安装C++构建工具。假设你使用Ubuntu/Debian。以下是安装说明：

- `build-essential` ：包含编译C/C++项目所需的大部分组件的包
- `cmake` ：用于创建作业的工具
- `xorg-dev` ：包含许多与X Window系统相关的依赖项的源码
- `libfreetype6-dev` ：图形学课程作业使用的依赖项
- `mesa-common-dev libgl1-mesa-dev libglu1-mesa-dev` ：OpenGL开发库

```
sudo apt update
sudo apt install build-essential cmake xorg-dev libfreetype6-dev
sudo apt install mesa-common-dev libgl1-mesa-dev libglu1-mesa-dev
```

2. 编译并运行作业

注意：对于每个作业，你必须至少执行一次这些步骤。

程序：Visual Studio、XCode、CLion、命令行和Visual Studio Code，

Visual Studio（Windows）

1. 在VS里打开项目，列出项目文件后，右键单击 `CMakeLists.txt` ，然后单击“CMake Settings for xxx”。
    - 在左侧点击添加按钮，添加“x64-Release”（或32位机器上为“x86-Release”）配置。
    - 然后，找到字段“CMake toolchain file”，输入 `[location of vcpkg]/scripts/buildsystems/vcpkg.cmake` 。（例如 `C:/vcpkg/scripts/buildsystems/vcpkg.cmake` ）
2. “Build”->“Build All”
    - 然后，打开代码文件夹下的src文件夹并右键单击该项目的主文件（例如，main.cpp，然后选择“设置为启动项”。
3. 然后你就可以运行你的项目了
4. 要设置可执行文件的参数（如文件路径等）：
    i. 选择 `meshedit.exe` 作为运行目标（绿色“运行”按钮右侧的选择框）
5. 进入菜单->调试->调试并启动设置
6. 然后 `"name": "meshedit.exe"` ，添加注释并选择一个新行：（ `"args": ["../../../svg/basic/"]` 或你的任何 svg 文件）
7. 你的整个调试和启动配置应如下所示：

```
{
  "version": "0.2.1",
  "defaults": {},
  "configurations": [
    {
      "type": "default",
      "project": "CMakeLists.txt",
      "projectTarget": "meshedit.exe",
      "name": "meshedit.exe",
      "args": ["../../../svg/basic/"]
    }
  ]
}
```

请参阅调试部分以了解如何在 Visual Studio 中使用调试器。

XCode（MacOS）

Ren Ng(吴恩达) 教授制作了一个有用的 XCode 视频教程，与此处的说明类似。该视频还包含有关如何使用 Xcode 进行调试的更多详细信息。

1. 确保安装了Xcode.app
2. 打开命令行终端，输入 `cd` 将目更改为作业文件夹
3. 创建一个名为 `xcode` 的文件夹，然后使用 CD 进入该文件夹： `mkdir xcode; cd xcode`
4. 为XCode设置cmake（在 `xcode` 文件夹内)： `cmake -G Xcode ..`
5. 打开 `Finder` 并导航至 `xcode` 文件夹
6. 双击该 `<Homework Name>.xcodeproj` 文件可以在 XCode 中打开项目
7. 点击屏幕顶部的 `Product > Build` 来构建项目
8. 单击 `ALL_BUILD`，位于 Xcode 右上角（有关更多详细信息，请参阅前面的视频）
9. 点击 `Edit Scheme`
10. 侧面选择 `Run`
11. 在页面 `Run` 顶部栏上选择 `Info`
12. 设置 `Executable` 为执行文件（对于作业2来说，是的 `meshedit` ）
13. 选择 `Arguments` （在 `Edit Scheme > Run` 页面）来指定参数（对于作业2来讲，需要设定曲面和曲面的文件路径）（请参见Ren的视频帮助）
14. 选择 `Options` （在 `Edit Scheme > Run` 页面）并指定工作目录为 `xcode` 文件夹。
15. 按执行（play)按钮编译并启动程序

常见错误：

- 如果运行cmake（上面的第4步）时找不到C/C++编译器，请尝试使用环境变量指定编译器： `CC=gcc CXX=g++ cmake -G Xcode ..`

CLion（Windows / MacOS / Linux）

要在 CLion 中成功编译，需要遵循一组特定的说明。请务必仔细阅读这些说明。如果你使用的是 Windows，你仍然需要遵循设置说明并安装 Visual Studio，因为 C++ 与 Visual Studio 捆绑在一起。

1. 打开CLion（可免费使用使用Jetbrains学生许可证可免费使用）

2. 停留在`Welcome to CLion`窗口处。这是默认打开的窗口。要到达此处，请关闭所有 CLion 窗口并重新打开 CLion。
3. 点击`Open`
4. 在刚才弹出的文件资源管理器中导航到你的作业文件夹
5. 选择`CMakeLists.txt`并点击打开（只有选择此特殊文件时才能打开）
6. 点击`Open as Project`
7. 找到Build按键开始构建
8. 要指定程序参数，请找到Run菜单，然后使用`Edit Configurations`（对于作业2来说需要指定文件路径）
9. 通过 CLion GUI 或命令行运行项目（可执行文件/二进制文件将位于`cmake-build-debug`文件夹内）

命令行（MacOS / Linux / Windows）

我们建议使用配置好的文本编辑器（例如Visual Studio Code或 neovim）来运行命令行进行项目Build，文本编辑器可使用 clangd等配置来启用语法突出显示和高级调试等功能。

1. 打开命令行终端，输入`cd`将目录更改为作业文件夹
2. 创建一个名为`build`的文件夹，然后使用 CD 进入该文件夹：`mkdir build; cd build`
3. 设置cmake（在`build`文件夹内）：`cmake ../`
4. 构建/编译项目（在`build`文件夹内）：`make`
5. 运行项目（在`build`文件夹内）：`./meshedit`

`cmake ../`和`make`之间的区别

`cmake`和`make`做完全不同的事情：

- `cmake`从项目根文件夹中的配置文件（CMakeLists.txt）**生成一个 c++ 项目**
- `make`将项目中的所有C++代码**编译为二进制文件。**

除非您希望更改项目结构（例如，添加新库、创建新的 .cpp 文件或在调试和发布版本之间切换），否则每个项目`cmake ../`只需运行**一次。**

但是，每次修改代码时都需要运行`make`，以重新编译应用程序。

使用 Ninja 执行编译（可选）

构建大型C++项目时，你可以使用“Ninja”代替“Make”来显着加快重建时间（超过10倍！）。

请先安装ninja，在macOS上执行`brew install ninja`，或在ubuntu-linux上运行`sudo apt install ninja`。

要使用 Ninja，应将步骤 3 的命令替换为以下命令：

```
cmake -G "Ninja" ../
```

对于步骤`4`，运行`ninja`而不是`make`

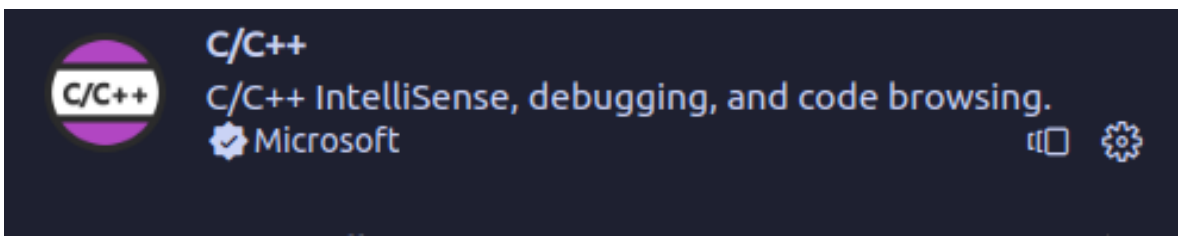
Visual Studio Code（MacOS / Linux）

当配置了 LLVM Clang 后，Visual Studio Code（VSC）就会变成强大且性能卓越的 C++ IDE。

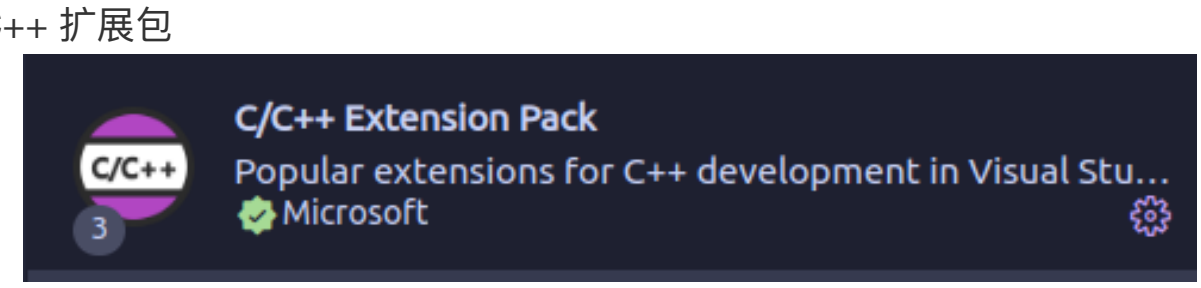
请注意，如果您是 Windows 或 macOS 用户，您可能仍需要在安装 Visual Studio/XCode 之后为您的机器设置代码环境。这是为了确保您的机器具有所需的 C++ 依赖项（GDB、CMake 和 g++）。通过其他方式安装依赖项是可行的，但不建议这样做，因为它们更复杂。

要实现语法高亮和调试等功能，请安装以下 VSC 插件：

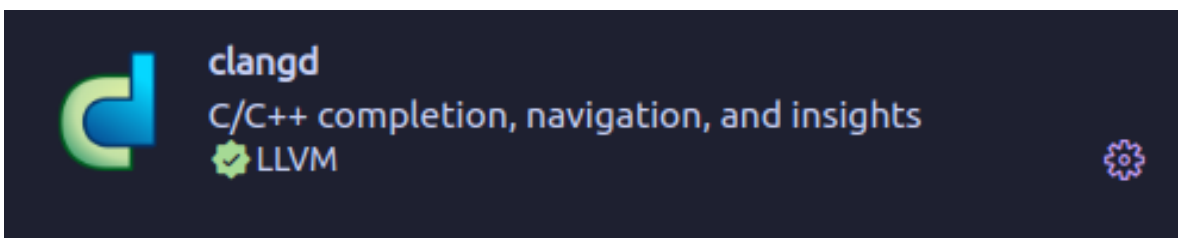
- C/C++



- C/C++ 扩展包



- clangd



完成后，请参考命令行部分来构建项目。

在运行 `cmake ..` 命令生成项目文件后，clangd 应该提供适当的语法高亮。

	
语法高亮未正确运行	语法高亮正常显示； 当鼠标悬停在类型上可正确显示其来源

格式化代码

对c++代码进行格式化/规范化显示，请使用组合键 `Ctrl + k + f` (windows/linux) 或 `Cmd + k + f` (macOS)。Clang 会自动对当前文件进行格式化。

如果您不喜欢默认的缩进级别（每个缩进2个空格），您可以通过创建`.clang-format`文本文件，放置在项目根目录下，来配置自定义格式。例如，设置`IndentWidth: 8`将缩进宽度调整为更容易阅读的格式。

请随意使用以下模板配置：

```
BasedOnStyle: LLVM
# Sets the base style for formatting. The LLVM style is used here.
IndentWidth: 8
# Specifies the number of spaces used for indentation.
ContinuationIndentWidth: 8
# Sets the width of the continuation line indentation.
PointerAlignment: Left
# Aligns the pointer symbol to the left with the type (e.g., int* a).
ColumnLimit: 120
```

完整的格式配置手册可以在这里访问。

调试

你可能经常使用普通的 printf() 语句来调试代码，但是如果使用调试器（debugger)，你将对代码有更深的洞察和控制。设置调试器配置可能需要一些时间，不过值得！

Visual Studio

在Visual Studio中调试：

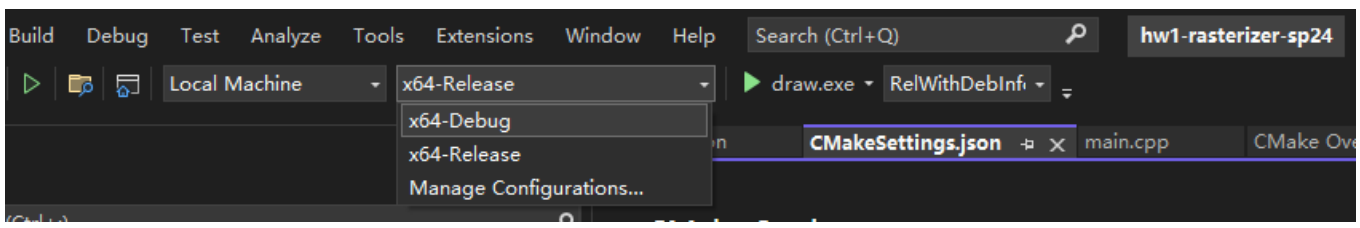
1. 设置调试配置（每个作业做一次）：

   右键单击`CMakeLists.txt`，然后单击“作业2 的 CMake 设置”

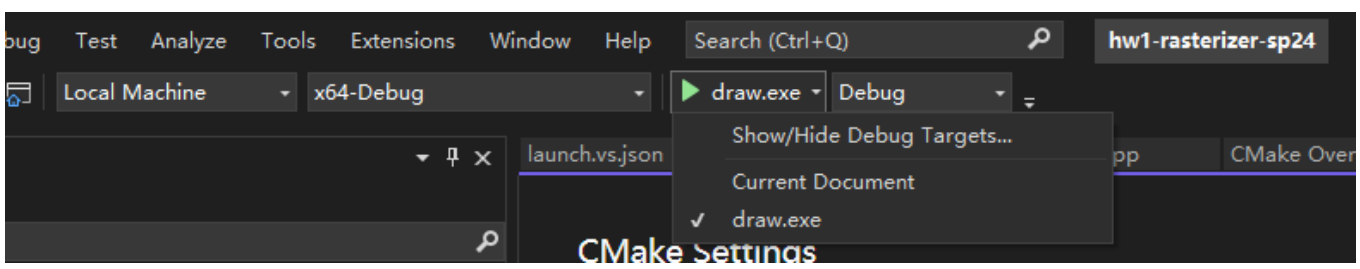
   - 在左侧中，选择“X64-Debug”
   - 然后，找到字段“CMake toolchain file”，输入`[location of vcpkg]/scripts/buildsystems/vcpkg.cmake`。（例如`C:/vcpkg/scripts/buildsystems/vcpkg.cmake`）：

2. 转到调试配置：

   在顶部，单击如下所示的下拉菜单，然后“X64-Debug”

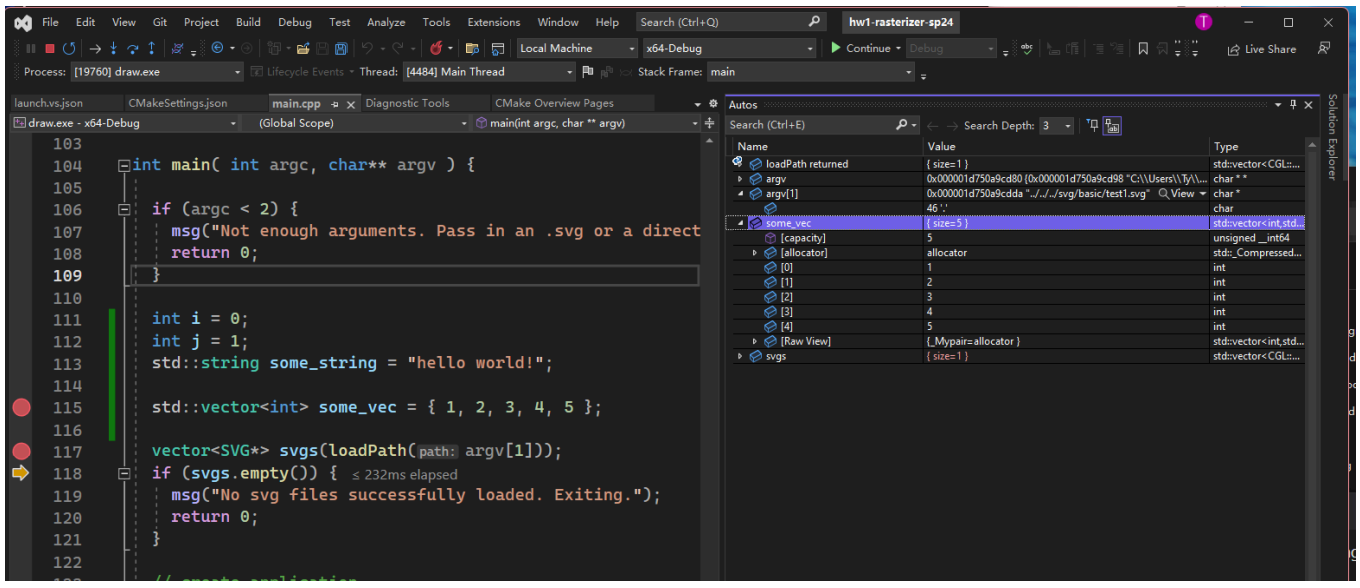
   

   切换后，“启动项”可能会重置。请确保输入正确的执行文件，在我们的例子中，执行文件是“meshedit.exe”：

3. 设置断点，并通过VS运行二进制文件

在要调试的代码部分设置断点，然后通过Visual Studio运行二进制运行文件。程序将继续执行，直到到达断点。然后，就可以执行变量检查等调试操作。



要停止程序，请按 `shift + F5` 。

调试完成后，记得通过下拉菜单切换回“Release”配置。由于编译器进行了一系列积极的优化，“Release”配置中的程序运行速度较快，但与调试器配合得不太好。

Visual Studio 代码

Visual Studio Code 与 GDB/LLDB 交互提供调试功能。

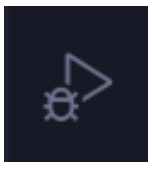
1. 将项目Rebuild为可调试的二进制文件

   使用调试参数，重新生成cmake文件，具体将 `DCMAKE_BUILD_TYPE` 其设置为 `debug` 。通常情况下，clang和g++编译器都会优化大量c++代码；优化使得默认编译的二进制文件不适合调试。将Build设置为调试将会禁止所有优化。

   ```
   cd build && cmake -DCMAKE_BUILD_TYPE=Debug ../
   ```

2. 设置调试配置（每个项目只需做一次）。

   i. 点击左栏上的“运行并调试”符号。

   ii. 单击 `create a launch.json` 文件，将提示你进入新创建的 `launch.json` 文件，该配置文件用于 VSC 调试。

   iii. 根据你的操作系统选择以下调试配置，粘贴到文件中：

MacOS：

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "(lldb) Launch",
      "type": "cppdbg",
      "request": "launch",
      "program": "${workspaceFolder}/build/quad_test",
      "args": [],
      "stopAtEntry": false,
      "cwd": "${fileDirname}",
      "environment": [],
      "externalConsole": false,
```

```
      "MIMode": "lldb"
    }
  ]
}
```

Linux：

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "(gdb) Launch",
      "type": "cppdbg",
      "request": "launch",
      "program": "${workspaceFolder}/build/quad_test",
      "args": [],
      "stopAtEntry": false,
      "cwd": "${fileDirname}",
      "environment": [],
      "externalConsole": false,
      "MIMode": "gdb",
      "setupCommands": [
        {
          "description": "Enable pretty-printing for gdb",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        },
        {
          "description": "Set Disassembly Flavor to Intel",
          "text": "-gdb-set disassembly-flavor intel",
          "ignoreFailures": true
        }
      ]
    }
  ]
}
```

对于不同的项目，配置`program`以更改二进制文件的名称；配置`args`可以像通过命令行一样输入不同的参数。

3. 返回调试面板并单击新建`(gdb) launch`或`(lldb)launch`按钮，设置断点并开始调试。VSC将运行程序、与gdb/lldb交互，并让你执行调试操作。