

# COMP5822M

## Software Setup

### Contents

1	Overview	1
2	Vulkan Implementation	2
3	The Vulkan SDK	2
4	C++ compiler / IDE	2
5	Renderdoc (optional)	3

Quick links:

- Vulkan SDK: <https://vulkan.lunarg.com/sdk/home>
- Visual Studio: <https://visualstudio.microsoft.com/vs/community/>
- RenderDoc: <https://renderdoc.org/>

## 1 Overview

This guide is mainly for people who wish to develop on their own personal machines (laptops/desktops). ~~Machines in the School of Computing facilities should be preconfigured already (2021/22. Machines in the School of Computing facilities are not useable at the time of writing; details will follow. (2022/23) The state of the machines in the School of Computing is entirely unknown at the moment and require further investigation (2023/24).~~ However, it might be a good idea to briefly skim through this document regardless.

You will need the following software to complete the practical work in COMP5822M:

- A Vulkan implementation
- The Vulkan SDK
- A (modern-ish) C++ compiler and/or development environment
- RenderDoc (optional)

COMP5822M assumes - and has been tested with - recent 64-bit Linux versions or 64-bit Windows 10 or 11 installations. You will additionally need a sufficiently recent GPU. Other OSs and/or 32-bit builds are untested and unsupported. You might be able to get it to work if the system has support for Vulkan, but things may not work out of the box.

While MacOS does not support Vulkan natively, Vulkan version 1.2 can be used through MoltenVK (which is part of the VulkanSDK these days). However, without consistent access to a Mac, it's not possible to test the exercises in this environment (unlike, for example, Windows, where Microsoft provides OS images for testing). It *should* be possible to complete most exercises on a Mac, but it may involve some additional work on your part.



It is further recommended that you use source control (e.g., git or similar) for your practical work. However, **do not share/publish your solutions to either exercises or courseworks online!** (For example, use private repositories on Github!)

## 2 Vulkan Implementation

The Vulkan implementation is provided by the graphics drivers for your graphics card (standalone software implementations of Vulkan do exist, but I would recommend against trying to use one for COMP5822M). The Vulkan documentation frequently refers to such drivers as ICDs (installable client drivers). Chances are you already have one installed (your graphics drivers). However, it is a good idea to update your graphics drivers if you have not done so recently. The drivers will also provide a Vulkan loader, which is needed to get access to the Vulkan API.

COMP5822M mainly targets Vulkan version 1.2 (but you can use features from Vulkan version 1.3 if you wish). Most desktop-class GPUs and drivers seem to support this. Check [vulkan.gpuinfo.org](https://vulkan.gpuinfo.org) to see what Vulkan versions have been reported for your GPU – or wait for Exercise 1.1, in which you will learn how to enumerate available devices using Vulkan.

MacOS will likely require MoltenVK to be installed. I would assume that it is installed by the Vulkan SDK (next section). It also seems to be installable via [Homebrew](https://brew.sh/): `brew install molten-vk`. This is mostly untested, though, so do either on your own risk.



## 3 The Vulkan SDK

Grab the Vulkan SDK at <https://vulkan.lunarg.com/sdk/home>. Pick the one that is appropriate for your system. (Linux users might want to check their package manager to see if the SDK is available that way.)

COMP5822M has been tested with SDK version 1.3.268 from October 24, 2023. Make sure you get this version or a newer one. None of the optional components from the installer are required, but can choose to install them if you so wish.

The Vulkan SDK provides a few different components (although, for convenience, some are provided by the exercises and coursework code as well). The main components that we want are:

- the validation layer, `VK_LAYER_KHRONOS_validation`
- the `vkconfig[.exe]` tool that allows us to conveniently configure the validation layer above

## 4 C++ compiler / IDE

The code provided with COMP5822M uses (parts of) C++17, so you will need a C++ compiler that has a functioning C++17 implementation. For Windows, Visual Studio 2022 is recommended. For Linux, a recent GCC (probably  $\geq$  version 11) is recommended, though most of the testing was done with GCC 13.x. MacOS will likely use the Clang toolset, of which recent versions should work as well.

At University of Leeds, you can use GCC 11.2 on the machines in the 24h teaching lab (Room 2.15 in Bragg). Use the command `module load gcc` to load the newer GCC version (otherwise, you will be stuck with the default GCC 8.5).

The machines in the Visualization Teaching Lab (2.16 in Bragg) have Visual Studio 2022 pre-installed, along with the other software mentioned in this document.



### Linux - GCC

Consult your system's package manager for installation of GCC and related software. If necessary, make sure to install the C++ components as well (`g++`).

Furthermore, you might want to make sure that you have a decent text editor with syntax highlighting for C++ and (modern) GLSL as well as auto-completion. The latter is especially useful when writing Vulkan code. Additionally, make sure that you have a debugger installed. If you know how to use `gdb`, it will do just fine.

### Windows - Visual Studio

If you do not have Visual Studio installed, you can get the Visual Studio Community 2022 edition:

- <https://visualstudio.microsoft.com/vs/community/>.

The community edition is free for individual and academic use – see information on the linked page for details. The code has been tested with both VS 2019 and 2022, so either will work. If you already have Visual Studio installed, make sure that the required C++ tooling is available and, if necessary, update to the latest version of the tools for your VS version.

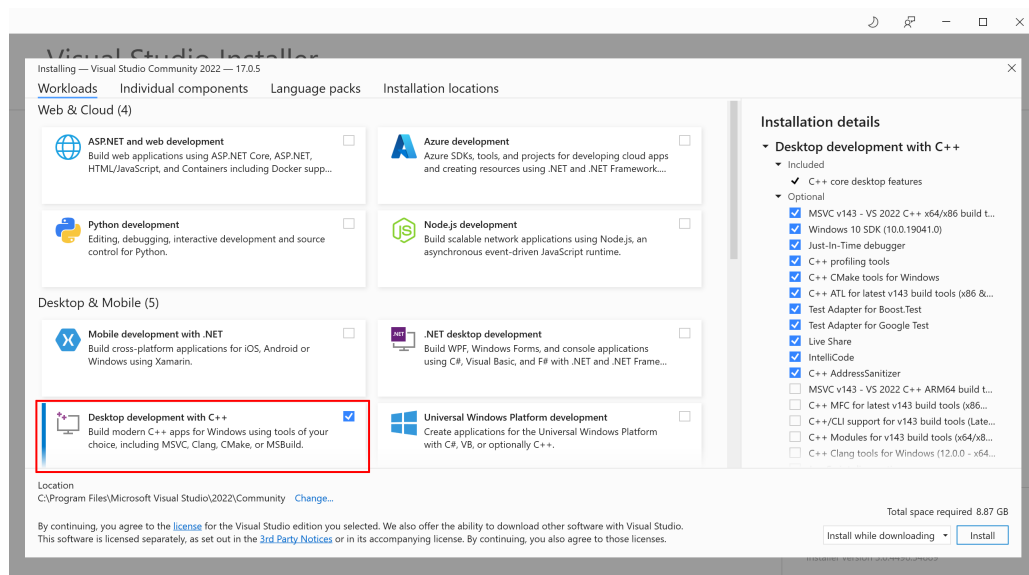


Figure 1: Visual Studio Community 2022 installer, workload selection.

For a fresh install, first select *Desktop development with C++* in the category *Workloads* (see Figure 1). In the category *Individual components* (see Figure 2), further select the following:

- *Compilers, build tools, runtimes* → *Windows Universal CRT SDK*
- *SDKs, libraries, and frameworks* → *Windows Universal C Runtime*
- (optional) *Code tools* → *Git for Windows*

The total installation seems to require about 9 GB of disk space.

The Visual Studio install has occasionally given some trouble, where some key components were missing and it has been impossible to build standard C++ programs (despite selecting the C++ workload). The additional selections under *Individual Components* attempts to account for this. Nevertheless, I've been unable to get a fresh Windows image to install and test VS 2022 from scratch. With VS 2022, the additional components *might* no longer be necessary. You may try this - if you do, please let me know if it worked.

You will be working with GLSL code quite a bit, but Visual Studio does not natively support GLSL source code. It is recommended that you install an extension that provides syntax highlight for GLSL code. To do so, go to *Extensions* → *Manage Extensions* in the Visual Studio menu. This should bring up a popup (Figure 3) where you can manage and install extensions. Here, select *Online* in the left hand menu, and search for `glsl` in the top-right search field. You should find an extension called *GLSL language integration (for VS2022) [Preview]*. Press *Download* to install the extension. You will likely need to restart VS to complete the installation.

Unfortunately, the extension is not installed on the machines in the Visualization Teaching Lab. Installation of the extension requires elevated privileges, so it is not possible to install it ourselves. I have asked IT to install the extension, and ~~hopefully it will become available within the next weeks~~ (nevermind, this was last year, and they never did – chances are not good).

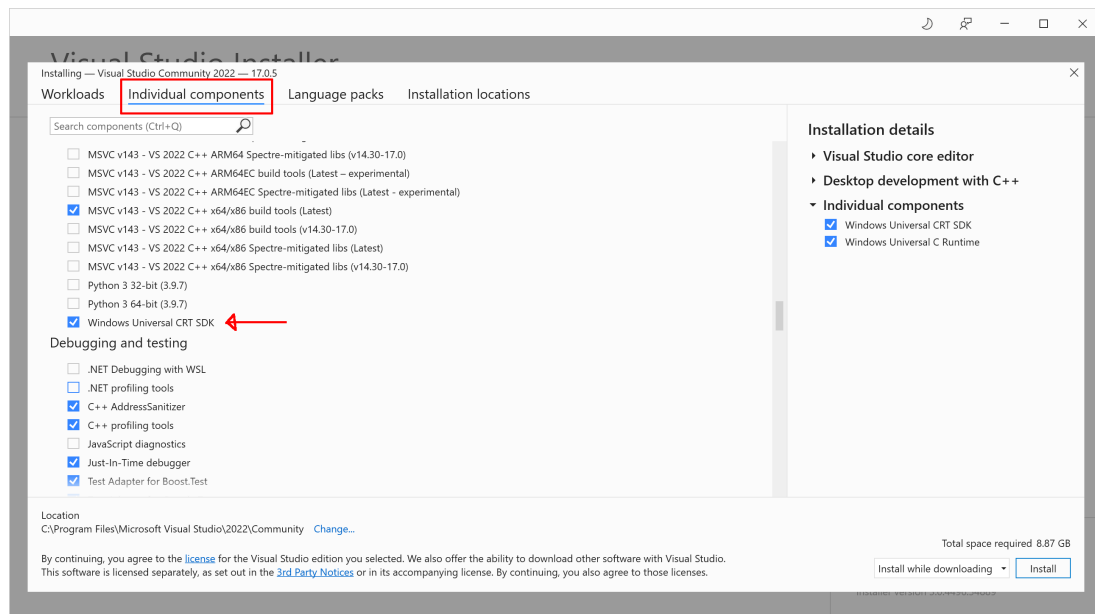
## MacOS

“Instructions” are less detailed and more of a guesswork for MacOS. You will have to install C++ development tools, probably via Xcode.

I've tested building the applications on a MacOS headless image on Github with the Linux command line instructions. These seem to mostly work.

## 5 Renderdoc (optional)

RenderDoc is a stand-alone graphics debugger, specifically designed to help debug graphics code using e.g. Vulkan, OpenGL or DirectX. RenderDoc can capture the commands that your application submits and allows you to view these after the fact. You can inspect the parameters that went into the commands, and in some cases view intermediate results and similar.



**Figure 2:** Visual Studio Community 2022 installer, component selection. The Windows Universal SDK is shown as an example. Scroll through the list or use the Search components field to find the relevant components.

This does not replace the normal debugger, such as the visual debugger in Visual Studio or `gdb` in Linux. Those help you debug your C++ code, whereas RenderDoc specifically helps with looking at and analyzing e.g. the Vulkan commands and resources that you execute and use.







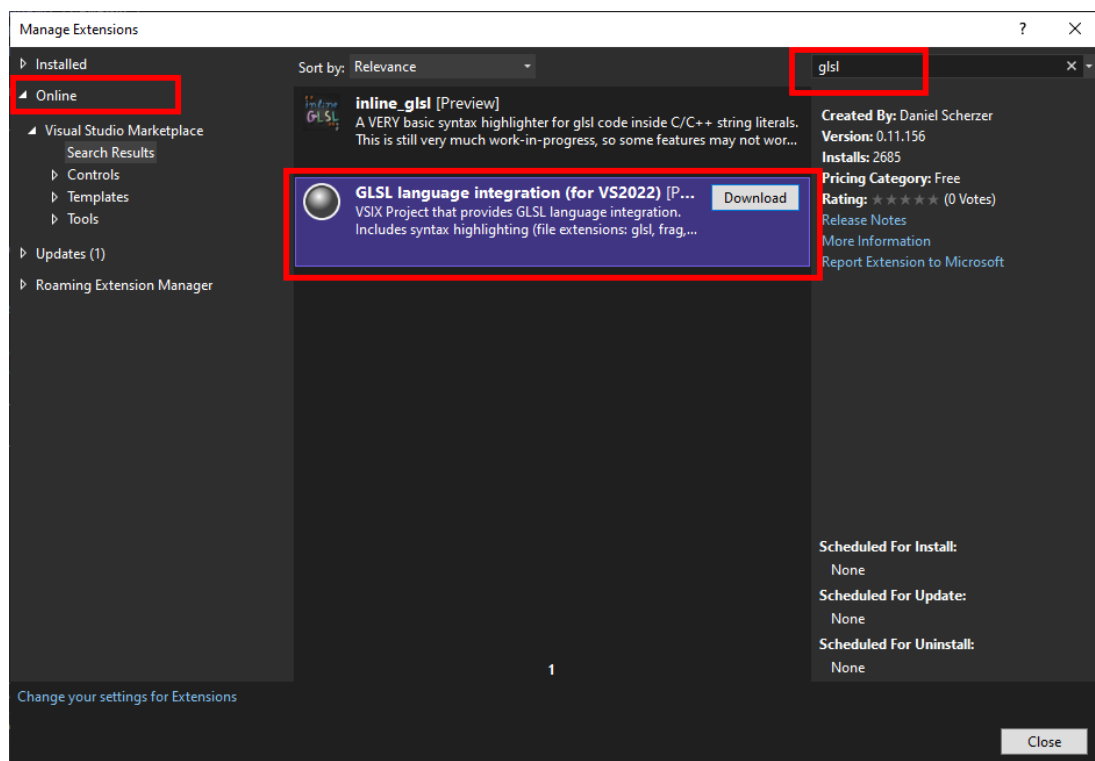
RenderDoc is available from <https://renderdoc.org/>. (Linux: as always, check your package manager.)

## Next steps

You may now start with Exercise 1.1, which will also help verify that the required software was indeed installed successfully.

### Acknowledgements

The document uses icons from <https://icons8.com>:    . The “free” license requires attribution in documents that use the icons. Note that each icon is a link to the original source thereof.



**Figure 3:** Visual Studio Extensions Manager (I'm using the dark theme). Note that you will need to select Online on the left hand side before searching for the extension.