

# CGRA C++ Setup Guide

Joshua Scott and Ben Allen

## 1 Introduction

This guide is provided to assist you with setting up the CGRA C++ assignments on ECS and Windows computers. This guide is officially supported; if you are having problems with it please contact one of the tutors

The base code for the CGRA C++ assignments and projects uses a CMake build system for simplicity and cross-platform development. We would prefer you didn't change this, but you can if you must. If you do significantly alter the build process, please submit a `README.txt` must make it abundantly clear how to build and run your project on the ECS Linux systems.

## 2 Directory Structure

Figure 3 is an example of the directory structure for a CGRA C++ assignment. This structure has been designed for ease of use and platform independence.

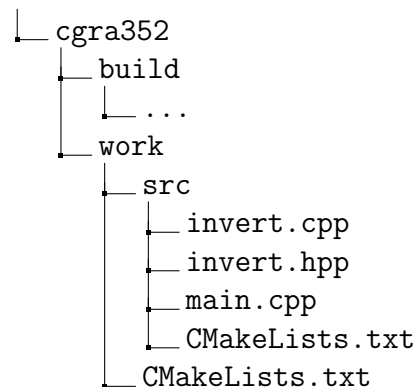


Figure 1: Assignment directory structure

<b>build</b>	This is where you'll run CMake and build your executable.
<b>work</b>	The working directory for the project. This is the one to submit for each assignment.
<b>CMake</b>	This folder holds CMake magic, no need to modify.
<b>src</b>	Source files ( <b>.hpp</b> and <b>.cpp</b> ).
<b>CMakeLists.txt</b>	This file holds CMake magic. You will not need to modify the one in the <b>work/</b> directory, but you may need to modify one or more in <b>work/src/</b> if you want to add files to the project.

## 3 CMake

### 3.1 ECS

You will need at least CMake 3.1 installed on your system and on your **PATH**. For the ECS systems there is no need to install CMake, it is already on the system for you. For other systems you can get the latest version from <http://www.cmake.org/>. The CMake installer should have an option to add itself to **PATH** for you.

## 4 ECS Setup

### 4.1 Text Editor

Completing the assignments with a text editor has an easier setup with less potential disasters but often takes longer to debug programs. If pursuing this route we recommend acquiring a more powerful text editor like 'Sublime Text 3' (x64-bit tarball for Linux) to work with. To build your project, first navigate to the assignment directory from a terminal. If the build directory is missing, create it with:

```
mkdir build
```

Then execute:

```
cd build
cmake ../work
make
cd ..
```

If the project builds without errors the executable should be located in the **build/bin/** directory and can be run with:

```
./build/bin/cgra352
```

Or if you need to add additional arguments, such as the provided `vgc-logo.png`:

```
./build/bin/cgra352 work/res/vgc-logo.png
```

All of your changes should be made in `work/src/` in the base directory, **NOT** the `build/src/` directory. For most changes you'll simply need to run `make` in the `build/` directory to compile your changes. If you add additional source files make sure you list them in `work/src/CMakeLists.txt`; when you run `make` CMake will detect the change, regenerate the Makefile and re-run `make` for you.

## 4.2 Eclipse

Eclipse is an IDE installed on the ECS machines and although the setup is more complicated, but once it has been setup it offers benefits like error checking ahead of compiling. Eclipse wasn't built with CMake in mind so it does have some issues and if you encounter any you can't resolve, please contact the tutors.

Running CMake is very similar, but with the additional argument (`-G <tool>`) specifying how the project should be built.

```
cd build
cmake -G "Eclipse CDT4 - Unix Makefiles" ../work
```

Start Eclipse and go to `File > Import > Existing Projects into Workspace`, browse to and select the `build/` directory as the project. Make sure the box `Copy Projects into Workspace` is unchecked.

Once you've imported the project, you should be able to run it straight away. If you can't run it, do the following:

- Go to `Run > Run Configurations`. On the left side, select `C/C++ Application`, then in the `Main` tab, make sure your `C/C++ Application` field contains `./bin/cgra352` and `Enable auto build` is checked.
- On your project, right click `> Run As > C/C++ Application`. This should setup the default way to start the program, so you can simply run the project anytime after that.

If you need to run with arguments (and you will with some projects) go to `Run > Run Configurations > Arguments` and enter your arguments there. For example:

```
work/res/vgc-logo.png
```

If you add additional files to the source code, create them manually, **NOT** with Eclipse. You must also update the `CMakeLists.txt` in the `work/src/` directory to include (or exclude) the files, and re-run the `cmake` command to update the Eclipse project.

If you have problems, you can always try not using Eclipse.

## 5 Windows (Visual Studio) Setup

We recommend using Visual Studio 2017. The free ‘Community’ edition is also suitable or you can obtain a free academic licence for the ‘Enterprise’ edition from Microsoft through DreamSpark courtesy of your ECS enrolment; see <http://ecs.victoria.ac.nz/Support/TechNoteMsdnaaSoftware> for details. Visual Studio is licensed on a per-user basis, so you will be able to install it on several computers with the same key.

The installer for Visual Studio 2017 does not install the C++ toolkit by default; you must manually select the “Desktop development with C++” workload for installation.

Older versions of Visual Studio may be unable to build the assignments, as the provided assignment code may make use of previously unsupported C++11/14/17 features. We will endeavour to ensure the provided code builds in Visual Studio 2017.

### 5.1 OpenCV

To install OpenCV on Windows, download the latest OpenCV Windows release from <https://opencv.org/releases.html> and extract it to your local drive. Add the `bin\` directory of the appropriate binaries to your `PATH`, something like this: `C:\opencv\build\x64\vc15\bin`.

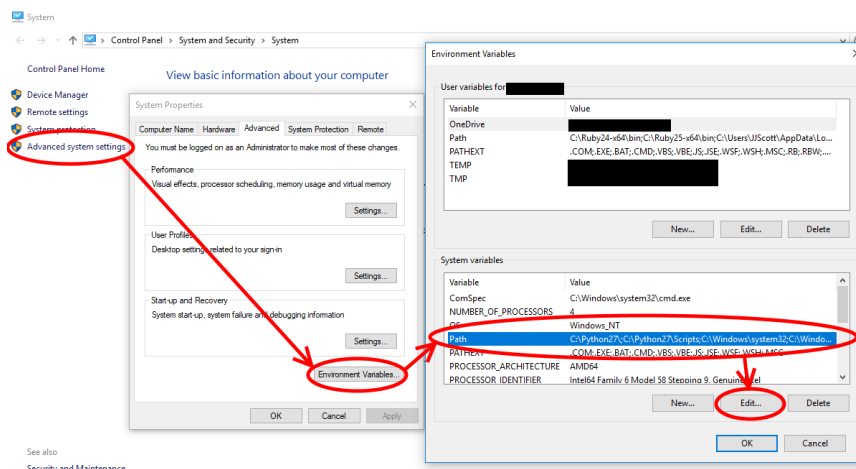


Figure 2: Setting an environment variable using windows.

### 5.2 Running CMake

Open `cmd.exe` and `cd` to the assignment root directory, the structure of which will look something like figure 3.

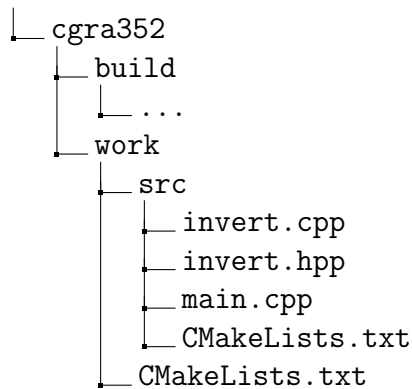


Figure 3: Assignment directory structure

If the `build\` directory doesn't exist, make it with `mkdir build`. Then `cd build` to enter it, and use one of the commands below to generate the Visual Studio solution. `XX` is your Visual Studio version, as in this table:

Product	Version
Visual Studio 2013	12
Visual Studio 2015	14
Visual Studio 2017	15

### 5.2.1 Building for x64

```
> cmake -G "Visual Studio XX Win64" -DOpenCV_DIR="C:/opencv/
    build/x64/vc14/lib" ..\work
```

You will only be able to run an assignment built this way on a x64 system. The extra argument (`-DOpenCV_DIR`) allows your project to find and use the OpenCV library on your machine. If your path to the file differs make sure you change it appropriately.

### 5.2.2 Building for x86

```
> cmake -G "Visual Studio XX" ..\work
```



## 5.3 Solution Preparation

Before you can run the assignment, there are a couple of changes you need to make.

### 5.3.1 Opening the Solution

In the `build\` directory there will be a `.sln` file, which is a Visual Studio *solution*. This serves as a container for projects. Double-click on this file to open it in Visual Studio, or open it from an already running Visual Studio instance.

### 5.3.2 StartUp Project

Set which project to run when you start debugging.

- Solution Explorer > a1 > right click > Set as StartUp Project

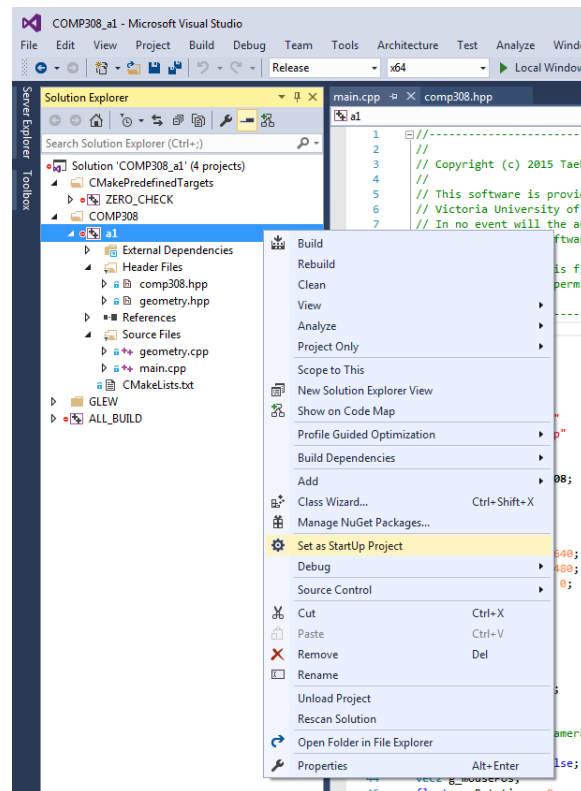


Figure 4: Setting the StartUp project

### 5.3.3 Working Directory and Command Arguments

Set the initial working directory and the command-line arguments for your program.

- Solution Explorer > cgra352 > right click > Properties > Configuration Properties > Debugging
- Select All Configurations from the configuration drop-down
- Set Working Directory to `$(SolutionDir)...`

- Set Command Arguments to whatever is required by your program

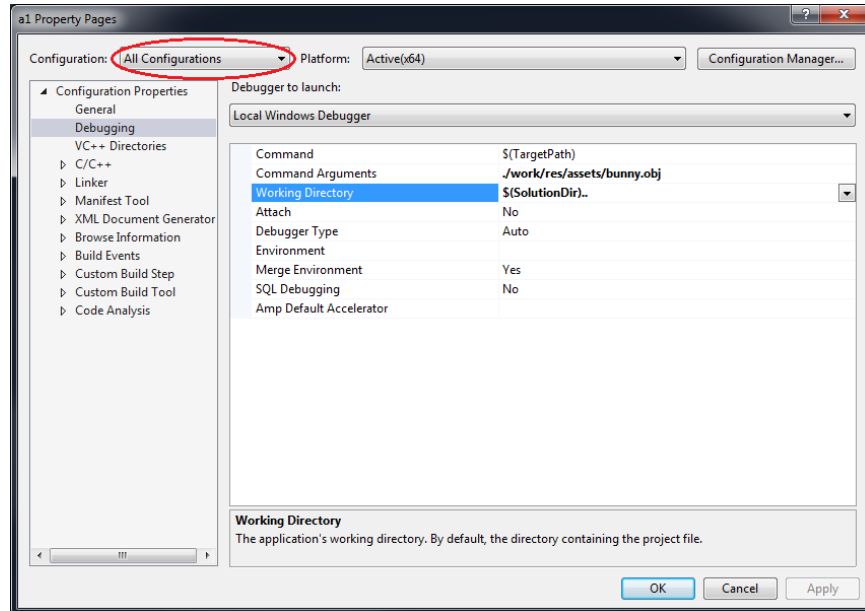


Figure 5: Setting the Working Directory

## 5.4 Running

Pressing the toolbar button with the green arrow (or just F5) will launch the StartUp project with Visual Studio attached as debugger.

### 5.4.1 Solution Configurations

CMake prepares several solution configurations for you, which can be selected from the dropdown on the toolbar.

<b>Debug</b>	Unoptimized with debug symbols, best for debugging
<b>RelWithDebInfo</b>	Optimized with debug symbols, fast but still debuggable
<b>Release</b>	Optimized for speed with no debug symbols, undebuggable
<b>MinSizeRel</b>	Optimized for size with no debug symbols, undebuggable

### 5.4.2 Adding Files

If you need to add additional source files, add them manually to the `work\src\` directory and list them in `work\src\CMakeLists.txt`. The next time you build or run through Visual Studio, CMake will detect the change and regenerate the solution, after which you will be prompted to reload it. If you have problems, you can try cleaning the solution (**Build > Clean Solution**) or re-running `cmake` from the command line.