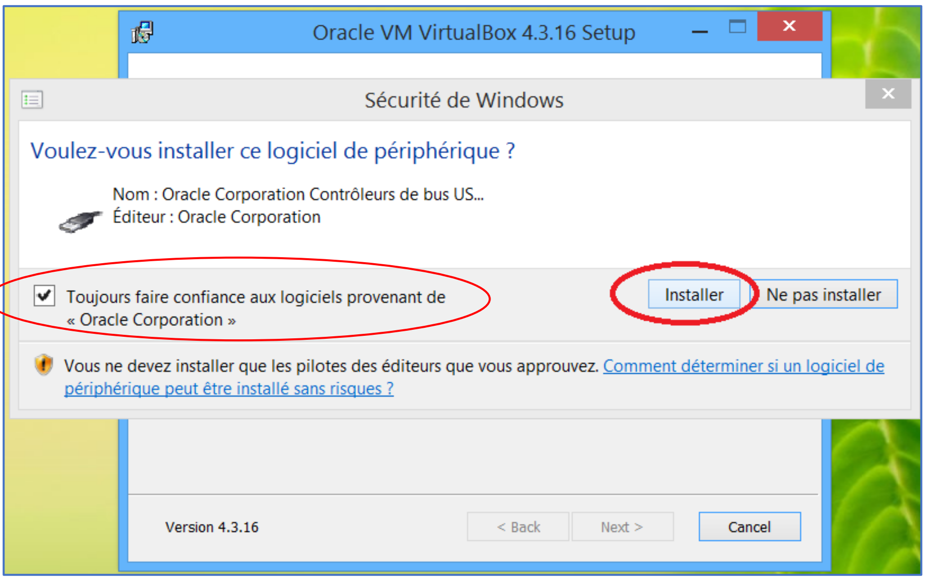
CADCG practical – Installing Ubuntu virtual machine for gnurbs and compiling

Install Virtual Box

- Download Virtual box for Windows or Mac from: https://www.virtualbox.org/
- Install Virtual Box. Keep the default options.
- If the install process asks you for installing a device software, click "install" (figure below) Be sure that the option "Always trust software from Oracle Corporation" is checked.


Oracle VM VirtualBox 4.3.16 Setup

Sécurité de Windows

Voulez-vous installer ce logiciel de périphérique ?

Nom : Oracle Corporation Contrôleurs de bus US...
Éditeur : Oracle Corporation

☑ Toujours faire confiance aux logiciels provenant de « Oracle Corporation »

Installer | Ne pas installer

Vous ne devez installer que les pilotes des éditeurs que vous approuvez. Comment déterminer si un logiciel de périphérique peut être installé sans risques ?

Version 4.3.16

< Back | Next > | Cancel

- When the install process is finished, do not start Virtual Box yet.
- Also, download and install the Extension Pack.
- Copy the file "CADCG.zip" to your computer and extract the archive.
- Note: Put the file "CADCG.vmdk" in a convenient repository, e.g., "CADCG/TPS/VM/CADCG.vmdk".
- **Be aware that all your files created within the virtual machine will be saved inside the file "CADCG.vmdk". Indeed, this file will behave as a hard-disk for the virtual machine.**

Creating an Ubuntu 22.04 machine on Virtual Box

- Launch Virtual Box and click the button "New" (Figure 1)
- In the new window, fill-in the fields as showed in Figure 2 :
    - Name: CADCG
    - Folder: Select the folder where the vmdk file is.
    - Type: Linux
    - Version: Ubuntu 22.04 (64-bit)

- Click then “Next” and choose at least 3000 Mb for the RAM memory (a higher RAM allows you to set a better resolution for instance, but you should not exceed the ¾ of the RAM memory of your own computer). Set the number of processors: 2 should be enough. See Figure 3.
- For the hard disk (Figure 4), choose the option “Use an existing hard disk” and locate the file “CADCG.vmdk”. A new window with the list of available hard disk appears (Figure 5): click on “Add” in the top left corner and find the file. After that validate the choice with “Choose”.
- Then, click on “Next”. Virtualbox provides you a summary. Verify it then validate the creation with “Finish”. Your virtual machine is almost ready. **Do not start it for the moment.**

Linking the virtual machine to the host machine

For more convenience, it is possible to transfer files from the host machine to the virtual machine and vice-versa. In order to enable this transfer, we must specify a folder on the host machine that will be shared with the virtual machine.

- Select the machine in the left panel and click on the button “Configure” (Figure 6).
- On the left panel, select “Shared folders” (Figure 7). Then, add a shared folder by clicking on the button “Add a new shared folder”.
- Select the folder of the host machine you want to share, tick the box “Auto-mount”, and click “OK”.
- The selected folder should appear in the list of shared folders. Click “OK” to finish.

Running Ubuntu 22.04

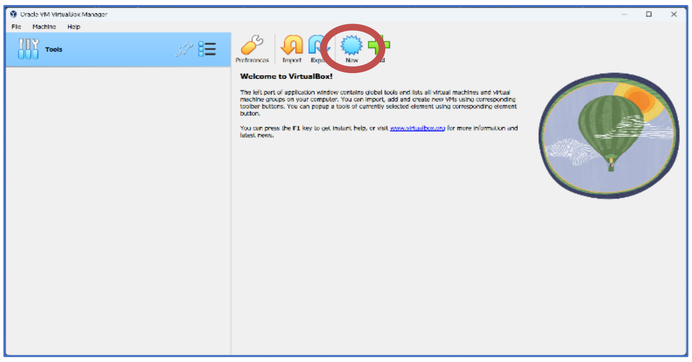
- Click on the button “Run” of the tab menu for running the machine. You should get something as shown in Figure 8. The grey panel on the right informs you that your mouse and keyboard will be automatically detected, click on the barred bubbles if you do not want to be shown again. **Note 1**: if the virtual machine does not launch and get an error like “VERR_VMX_MSR_VMXON_DISABLED”, you must enable the virtualization process in your bios options. This is machine dependent; refer to the documentation of your machine.
- **Remarks**:
  - **The user id of the virtual machine is:** ***student***
  - **The “super user” password is:** *1234*. This password will be asked if you try to update the system or install new software.
- **Full screen mode**: the virtual machine is by default running in windowed mode. If you want it in full screen mode, hit the keyboard keys “right ctrl + f” (Figure 9). Hit again these keys to exit the full screen mode.

  Note 2: if the virtual machine seems slow, you can try to accelerate it by checking the box “Configure→Display→Enable 3D acceleration”.

Note 3: If your screen turns black when resizing it, it certainly means that you do not have enough video memory. Allow more RAM to compensate in the configure menu or enable 3D acceleration.

- **Where is my shared folder?** Double-click on the "Home" folder on the desktop of the virtual machine. And then on "Other Locations" and "Computer" (Figure 10). The shared folder is "media/sf_[name-of-shared-folder]". (Note: the "1234" password will be required every time you want to access the shared folder).

Your Linux virtual machine is now operational. All you need for compiling and running your codes is already installed. Now we can build our first CodeBlocks project for our first code in the next section.

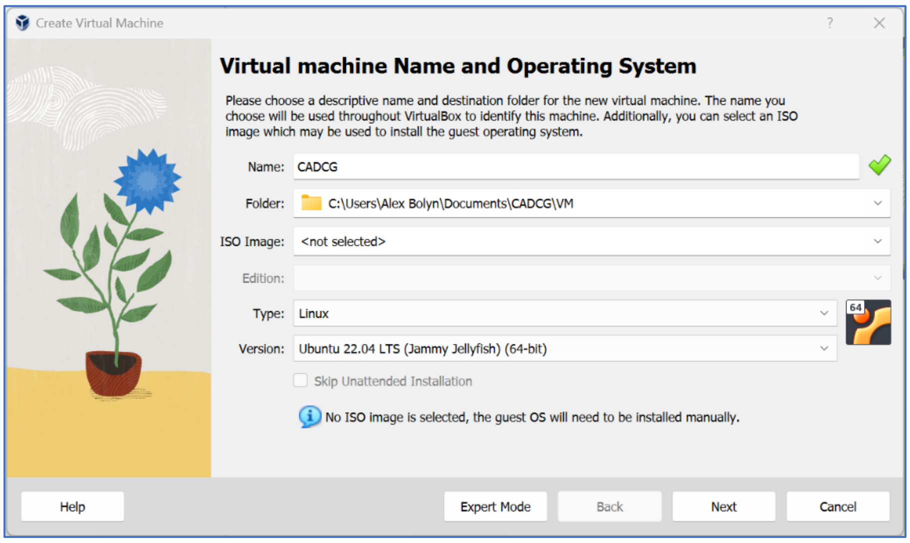


Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the F1 key to get instant help, or visit www.virtualbox.org for more information and latest news.

Figure 1



Virtual machine Name and Operating System

Please choose a descriptive name and destination folder for the new virtual machine. The name you choose will be used throughout VirtualBox to identify this machine. Additionally, you can select an ISO image which may be used to install the guest operating system.

Name: CADCG

Folder: C:\Users\Alex Bolyn\Documents\CADCG\VM

ISO Image: <not selected>

Edition:

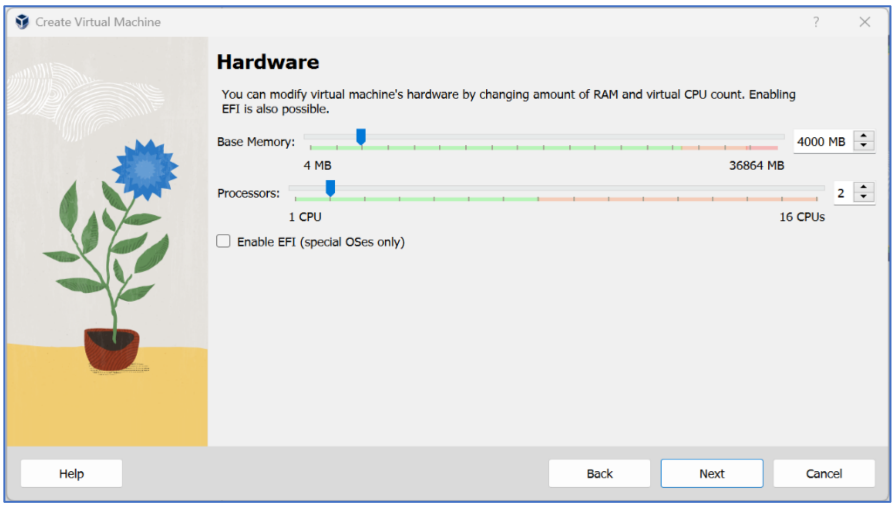
Type: Linux

Version: Ubuntu 22.04 LTS (Jammy Jellyfish) (64-bit)

Skip Unattended Installation

No ISO image is selected, the guest OS will need to be installed manually.

Figure 2


Create Virtual Machine

Hardware

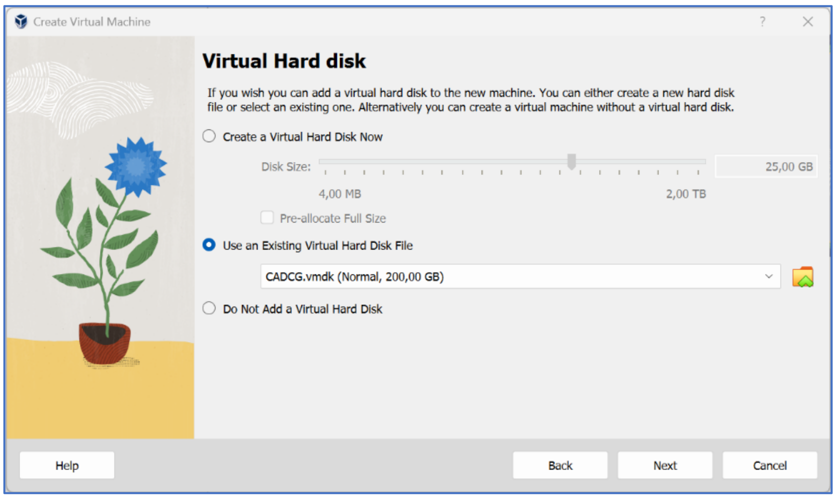
You can modify virtual machine's hardware by changing amount of RAM and virtual CPU count. Enabling EFI is also possible.

Base Memory: 4000 MB (4 MB – 36864 MB)

Processors: 2 (1 CPU – 16 CPUs)

☐ Enable EFI (special OSes only)

Figure 3


Create Virtual Machine

Virtual Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select an existing one. Alternatively you can create a virtual machine without a virtual hard disk.

○ Create a Virtual Hard Disk Now

Disk Size: 25,00 GB (4,00 MB – 2,00 TB)

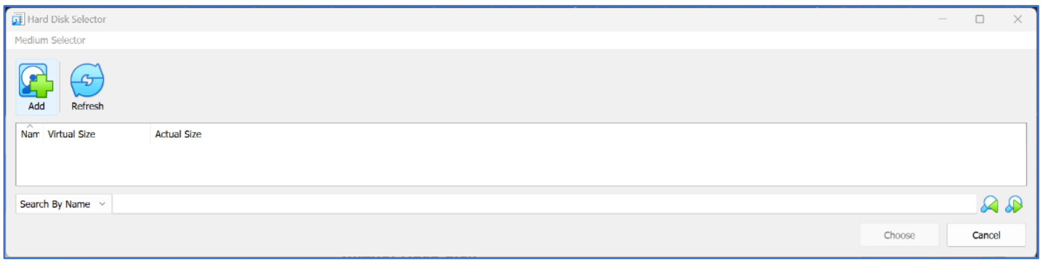
☐ Pre-allocate Full Size

● Use an Existing Virtual Hard Disk File

CADCG.vmdk (Normal, 200,00 GB)

○ Do Not Add a Virtual Hard Disk

Figure 4

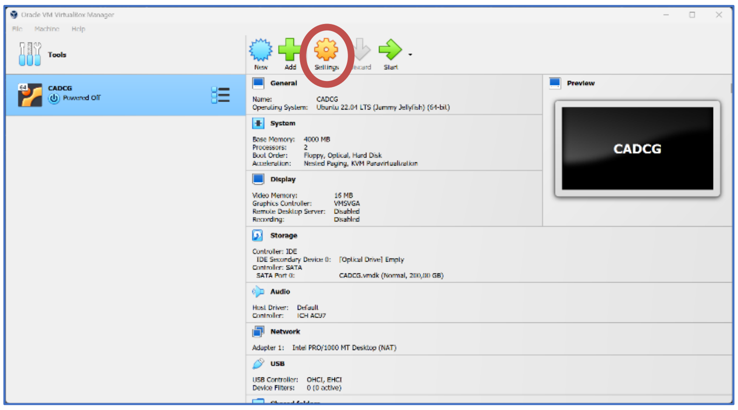

Hard Disk Selector

Medium Selector

Nam	Virtual Size	Actual Size

Search By Name

Figure 5



General

```
Name:              CADCG
Operating System:  Ubuntu 22.04 LTS (Jammy Jellyfish) (64-bit)
```

System

```
Base Memory:   4000 MB
Processors:    2
Boot Order:    Floppy, Optical, Hard Disk
Acceleration:  Nested Paging, KVM Paravirtualization
```

Display

```
Video Memory:            16 MB
Graphics Controller:     VMSVGA
Remote Desktop Server:   Disabled
Recording:               Disabled
```

Storage

```
Controller: IDE
  IDE Secondary Device 0:   [Optical Drive] Empty
Controller: SATA
  SATA Port 0:              CADCG.vmdk (Normal, 200,00 GB)
```

Audio

```
Host Driver:  Default
Controller:   ICH AC97
```

Network

```
Adapter 1:   Intel PRO/1000 MT Desktop (NAT)
```

USB

```
USB Controller:  OHCI, EHCI
Device Filters:  0 (0 active)
```

Figure 6

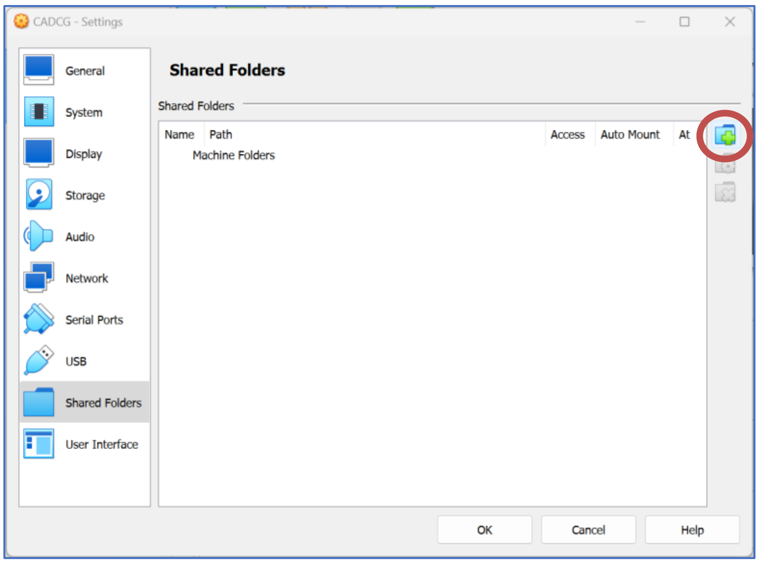


Figure 7

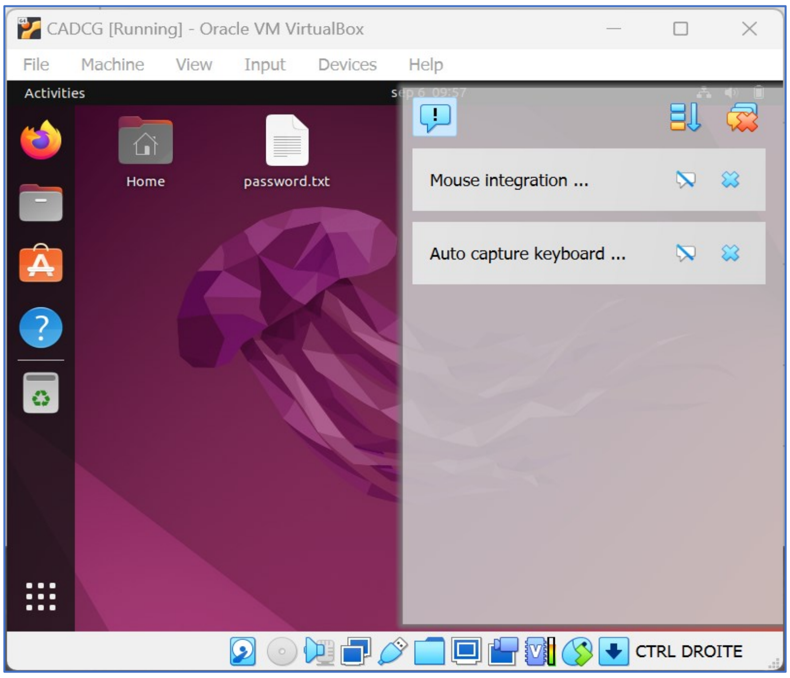


Figure 8

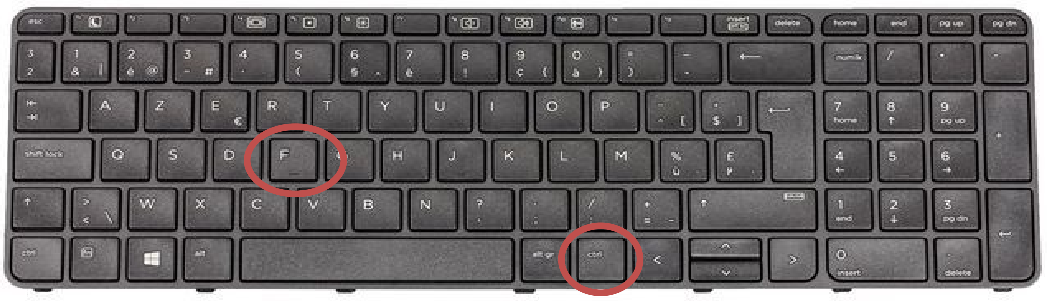


Figure 9

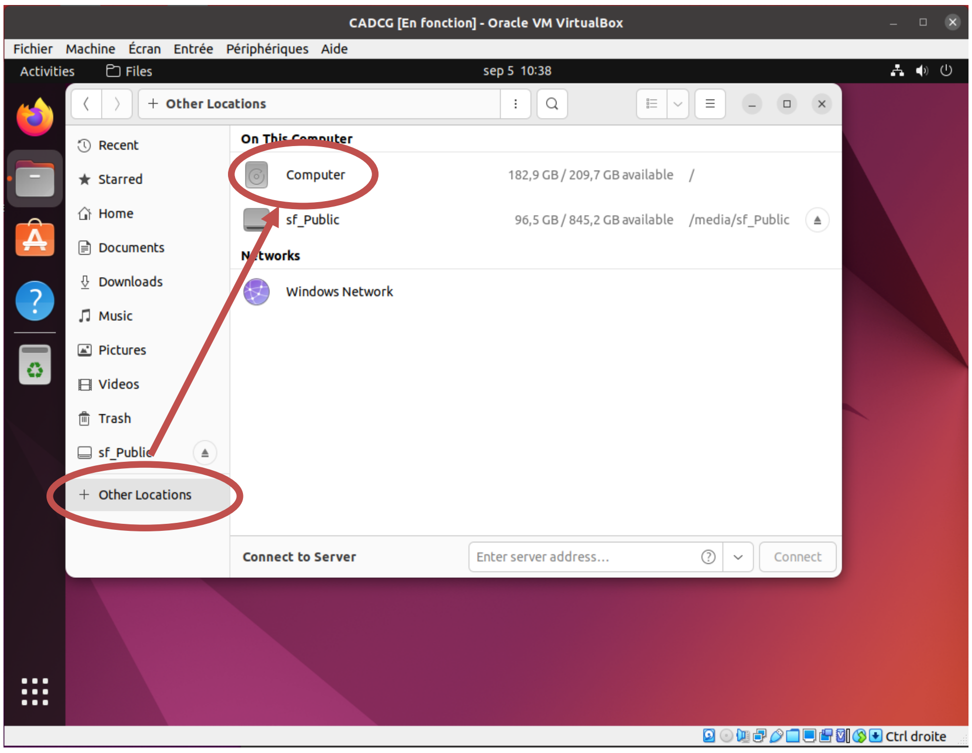


Figure 10

Compiling Gnurbs

Source code of Gnurbs (TP1)

1. From your virtual machine (Firefox), download the test source code on the webpage of the CADCG course (CADCG_TP1.zip): https://www.cgeo.uliege.be/CADCG/
2. Unzip the file into a folder, e.g. « Home/Documents/CADCG/TP1 ».
3. The zip file contains several folders and files. Here, we will be interested in mainly two folders:
   - The **test** folder contains files for generating the executables. Here, the file we will mainly work with is “test/curve/main.cc”.
   - The **api** folder contains files related to the **Gnurbs** library. The file “api/nlagrange.cc” which will compute Lagrange polynomials is actually the subject of the first session. The code in this file will be called by the executable generated from the file “test/curve/main.cc”.

You can download the file outside your virtual machine as well, but you need to pass it with the shared file. Since this shared file is protected, you need to extract the files outside the shared file to be able to work with.

Generating the project

As it is the case here, a C++ code rarely consists of a single file. More often, it consists in multiples files that are organized into multiple folders and sub-folders. Some files may contain code for a *library* which contains some defined functionalities.

For instance, the “api” folder contains files for the *Gnurbs* library which implements polynomials, splines, and nurbs curves and surfaces[1]. The “nutil” folder contains files for the *nutil* library implementing basic linear algebra and graphic display functionalities. In general, a library can be considered as a toolbox providing different services for a given usage (e.g. image analysis, solving differential equations, minimizing functionals, etc.)

Libraries are *compiled* and *linked* against an executable. Here the executable is called “curve”.

- *compiled* means that the human-readable C++ code[2] is translated into *assembly* code and then into machine code (i.e. instructions that can directly be fed to the CPU).
- *linked* means that two or more libraries or executables can call instructions and get data from the other.

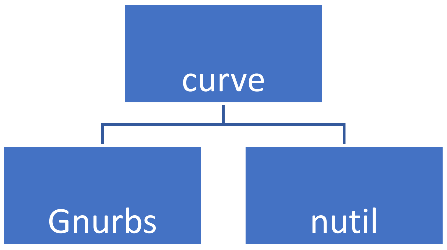
There are two kinds of *link*: *Static linking* and *dynamic linking*.

- *Static linking* means that the libraries and executable machine code are stacked into a single file. Such a file can run independently on its own but may be huge and requiring a lot of RAM (depending on the application).

[1] Note: for this first practical course this folder is pretty empty, but it will be increasingly extended with more files for the next practical courses.
[2] Some people argue that C++ is barely human, but the Perl language is utterly from another universe.

- *Dynamic linking* means that the libraries and executable are separated machine code files which have the information about each-other locations in memory. This result in much smaller executables and RAM resources are spared as several dynamic linked executables can use the same dynamic linked libraries[3]. However, these executables are no more independent and will not run if their dynamic linked libraries are absent (or the link is broken by, e.g., a change in the name of the library).

For instance, here libraries and executable are linked as follow:



Usually, C++ files for a library are of two kinds:

- *Header* files which *declare* existing functionalities (usual file extensions: h, hh, hpp, hxx).
- *Source* files which *implement* these functionalities (usual file extensions: c, cc, cpp, cxx).

This separation into *header* and *source* files allow any user of a given library to focus only on *how to use* the library and not be bothered about *how things are actually computed* inside the library (as you used `printf` for the `stdio` library in C without looking after how it works).

All of this is fine, but how to tell the compiler which source and header files are intended for library A, which for library B, which are for executables, etc ?
One option is to directly enter commands to the compiler. But it may lead to unbearable command lines like:

```
g++ file1.cpp file2.cpp file3.cpp \
  -std=c++14  -o app.bin -O3 -g \
  -Wall -Wextra -pendantic \
  -lpthread -lblas -lboost_system -lboost_filesystem \
  -I./include/path1/with/headers1 -I./include2 -L./path/lib1 -
L./pathLib2
```

To solve this problem, we use here the software *CMake*. This software takes as input script files called *CMakeLists* that describe how code files should compiled for which libraries, how to link them, and against which executables.

CMake allows to a lot of flexibilities. For instance, it allows choosing which libraries to use following the target operating system on which the executable will run[4]. It also allows to build different versions of a same executable for considering the presence of a graphic card or not,

[3] In windows you can see files with the “dll” extension. This means *dynamic linked library*.
[4] For instance, graphic libraries are different between Windows, Linux and Mac.

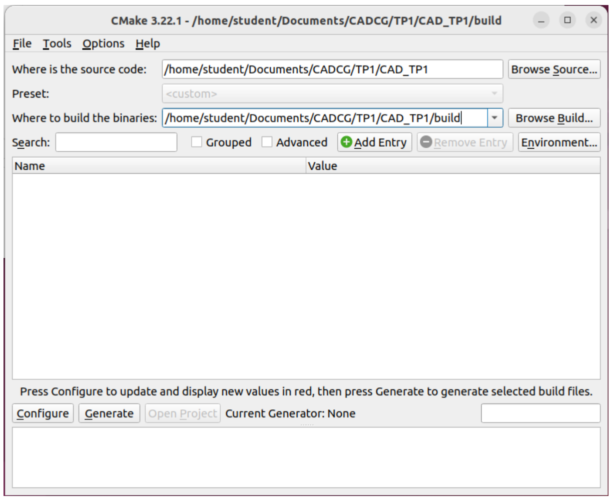
and even the presence of some CPU feature, like SSE capabilities[5]. In fact, it is even possible to compile and link libraries written with different languages, as, e.g., Fortran, C and C++. Finally, it is possible to build different versions of the same executable (debug version, optimized version, statically or dynamically linked version, etc). Finally, it also allows generating project files for given compilers and IDEs[6]. In short, the possibilities are endless.

Time to use CMake

Here, the CMakeLists files have already been written. It was decided to use the IDE CodeBlocks for the classes thus the files were edited to generate a CodeBlocks project. Hence, we only need to execute CMake on them.

1. Click on the "Show Applications" button located on the left of the screen. And then on the "CMake" application. The window presented in Figure 12 appears.
2. In the field "Where is the source code", enter the path to the folder "CAD_TP1" by clicking on "**Browse Source...**"
3. In the field "Where to build the binaries", enter the same path and add "/build" at the end, or choose another path. It is recommended to **NOT** build the binaries into the same path as the source code, as the build will generate a lot of intermediate files which will mess up with the source files[7]. If the folder "build" does not exist for moment, CMake will create it for us later. For now, your window should look as presented in Figure 12.
4. Click on the "**Configure**" button located on the bottom left of the window. If CMake asks you to create a directory, choose "Yes".
5. Next, in the "CMakeSetup" window (Figure 11), select as "generator for this project" the "**CodeBlocks – Unix Makefiles**". And then click on "Finish".
6. CMake will look the needed system libraries and show a list of available options (Figure 13). Here, the virtual machine has been configured to provide all the needed libraries, so no options need to be changed. Skip this step by clicking on "**Configure**" once more to accept the current options. Check that the log prompts at the bottom of the window shows "Configuring done" (Figure 14).
7. Now, click on "**Generate**" to generate the CodeBlocks project file. The log prompts should read "Generating done". If so, close CMake.
8. The "Gnurbs" project has been now fully configured for being compiled on an Ubuntu 64bits system using CodeBlocks and the GNU GCC[8] compiler.

[5] Streaming SIMD Extensions is a set of CPU instructions accelerating the computation of mathematical operations on vectors.
[6] Integrated Development environment: a software with a graphical interface for writing and compiling code in one or several languages.
[7] For the computer there will be no problem, but for the user (a.k.a. you) retrieving source files among a bunch of binary files will be painful. This kind of punishment should only be reserved to your worst enemy.
[8] GNU: *GNU is Not Unix* (note the recursive acronym). Unix-like operating system, but not Unix, kernel of the Ubuntu operating system.
GCC: *the GNU Compiler Collection*. A collection of compilers for C, C++, Fortran, Ada, etc. for the GNU operating systems.


CMake 3.22.1 - /home/student/Documents/CADCG/TP1/CAD_TP1/build

Where is the source code: /home/student/Documents/CADCG/TP1/CAD_TP1

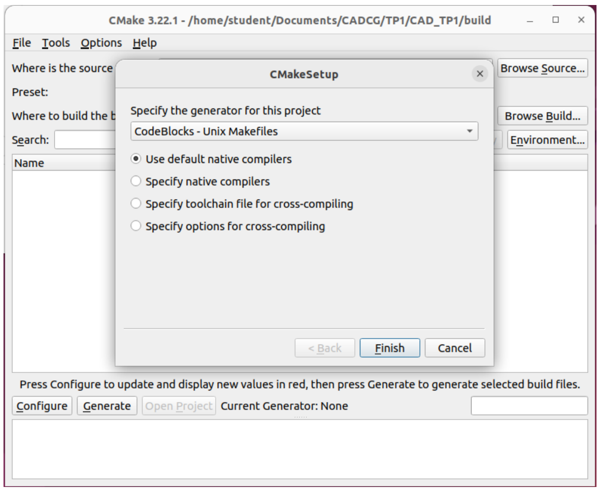
Preset: <custom>

Where to build the binaries: /home/student/Documents/CADCG/TP1/CAD_TP1/build

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: None

Figure 12


CMake 3.22.1 - /home/student/Documents/CADCG/TP1/CAD_TP1/build

CMakeSetup

Specify the generator for this project

CodeBlocks - Unix Makefiles

[✓ Use default native compilers]

Specify native compilers

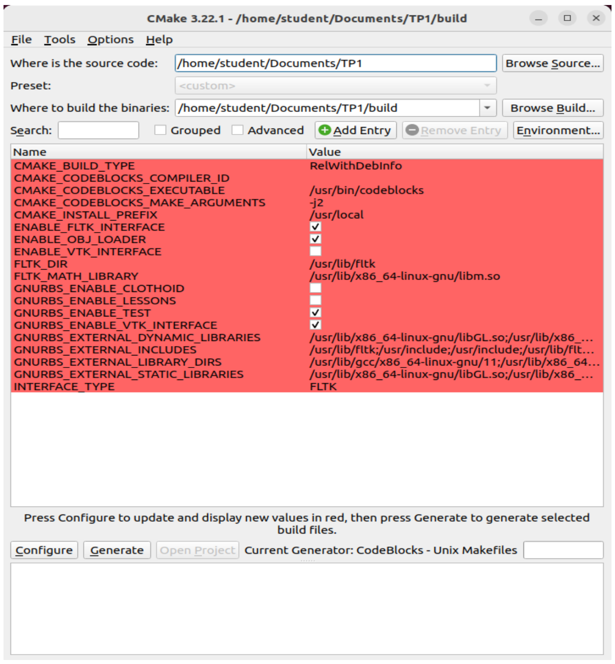
Specify toolchain file for cross-compiling

Specify options for cross-compiling

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: None

Figure 11


CMake 3.22.1 - /home/student/Documents/TP1/build

Where is the source code: /home/student/Documents/TP1

Preset: <custom>

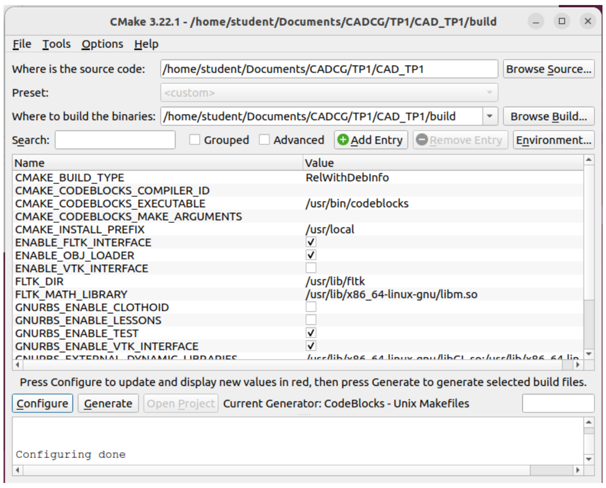
Where to build the binaries: /home/student/Documents/TP1/build

Name	Value
CMAKE_BUILD_TYPE	RelWithDebInfo
CMAKE_CODEBLOCKS_COMPILER_ID	
CMAKE_CODEBLOCKS_EXECUTABLE	/usr/bin/codeblocks
CMAKE_CODEBLOCKS_MAKE_ARGUMENTS	-j2
CMAKE_INSTALL_PREFIX	/usr/local
ENABLE_FLTK_INTERFACE	✓
ENABLE_OBJ_LOADER	✓
ENABLE_VTK_INTERFACE	☐
FLTK_DIR	/usr/lib/fltk
FLTK_MATH_LIBRARY	/usr/lib/x86_64-linux-gnu/libm.so
GNURBS_ENABLE_CLOTHOID	☐
GNURBS_ENABLE_LESSONS	☐
GNURBS_ENABLE_TEST	✓
GNURBS_ENABLE_VTK_INTERFACE	✓
GNURBS_EXTERNAL_DYNAMIC_LIBRARIES	/usr/lib/x86_64-linux-gnu/libGL.so;/usr/lib/x86_...
GNURBS_EXTERNAL_INCLUDES	/usr/lib/fltk;/usr/include;/usr/include;/usr/lib/flt...
GNURBS_EXTERNAL_LIBRARY_DIRS	/usr/lib/gcc/x86_64-linux-gnu/11;/usr/lib/x86_64...
GNURBS_EXTERNAL_STATIC_LIBRARIES	/usr/lib/x86_64-linux-gnu/libGL.so;/usr/lib/x86_...
INTERFACE_TYPE	FLTK

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: CodeBlocks - Unix Makefiles

Figure 14


CMake 3.22.1 - /home/student/Documents/CADCG/TP1/CAD_TP1/build

Where is the source code: /home/student/Documents/CADCG/TP1/CAD_TP1

Preset: <custom>

Where to build the binaries: /home/student/Documents/CADCG/TP1/CAD_TP1/build

Name	Value
CMAKE_BUILD_TYPE	RelWithDebInfo
CMAKE_CODEBLOCKS_COMPILER_ID	
CMAKE_CODEBLOCKS_EXECUTABLE	/usr/bin/codeblocks
CMAKE_CODEBLOCKS_MAKE_ARGUMENTS	
CMAKE_INSTALL_PREFIX	/usr/local
ENABLE_FLTK_INTERFACE	✓
ENABLE_OBJ_LOADER	✓
ENABLE_VTK_INTERFACE	☐
FLTK_DIR	/usr/lib/fltk
FLTK_MATH_LIBRARY	/usr/lib/x86_64-linux-gnu/libm.so
GNURBS_ENABLE_CLOTHOID	☐
GNURBS_ENABLE_LESSONS	☐
GNURBS_ENABLE_TEST	✓
GNURBS_ENABLE_VTK_INTERFACE	✓

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: CodeBlocks - Unix Makefiles

```
Configuring done
```

Figure 13