Assignment 1: Transformations
CS180 Winter 2021

Professor: Lingqi Yan

University of California, Santa Barbara

Assigned on Jan 14, 2021 (Thursday)

Due at 11:59PM Anywhere on Earth (AOE) on Jan 20, 2021 (Wednesday)

Notes:

- Be sure to read the university's Academic Integrity policy.
- Any updates or correction will be posted on piazza, so check there occasionally.
- You must do your own work independently.

---

1 Usage of Virtual Machine

This course will be using virtual machines (VM). You can write, compile, and run your codes in the virtual machine. By mounting the provided Virtual Hard Disk File, everyone is guaranteed to have an identical and sufficient coding environment. You do not need to set it up on your own.

1.1 Installation

We use Oracle VM VirtualBox in this course. You can download it using the following link:

- Windows:
  `https://download.virtualbox.org/virtualbox/6.1.16/VirtualBox-6.1.16-140961-Win.exe`
- Mac OS:
  `https://download.virtualbox.org/virtualbox/6.1.16/VirtualBox-6.1.16-140961-OSX.dmg`
- Linux: You can find the right version corresponding to your own kernel at
  `https://www.virtualbox.org/wiki/Linux_Downloads`.

1.2 Download the Virtual Hard Disk File (.vdi)

We have packed all the software you need and developing environment into one single `vdi` file: `https://ucsb.box.com/s/gyhsyc4q7b6se9wof03740qratbgjakm`. You can download and unzip it. The unzipped file (`Ubuntu 18.04.2 (64bit).vdi`) can be loaded by Virtual Box directly (see next subsection).

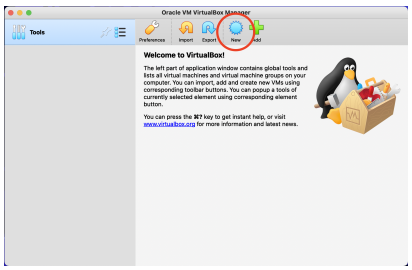
1.3 Virtual Box Configuration

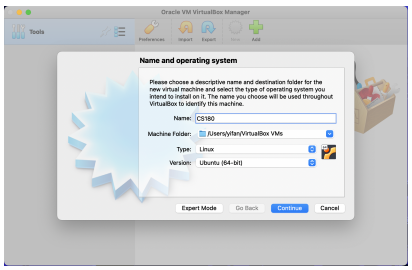
Open Virtual Box, click **New**. Fill in the blanks using the following information:

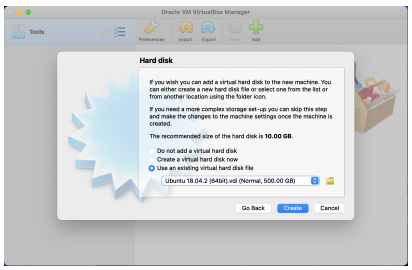
- Name: whatever you want;
- Machine Folder: use default settings or whatever you want;
- Type: Linux;
- Version: Ubuntu (64-bit).

Then click **Continue**. On the next page, we recommend you to set the memory size to no less than 2GB.

In the **Hard disk** page, choose **Use an existing virtual hard disk file**, and select the `vdi` file you just downloaded. Click **Create**, and you are done.



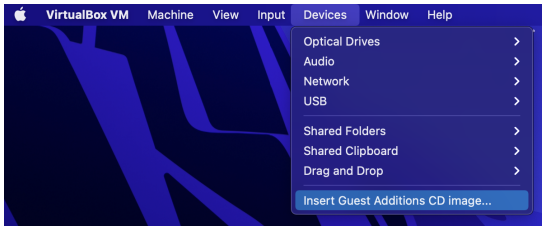




The password of the given system is **Ilovegraphics**.

1.4 Install Guest Additions

After you enter the system, please install guest additions, by clicking **Devices** → **Insert Guest Additions CD image...**.



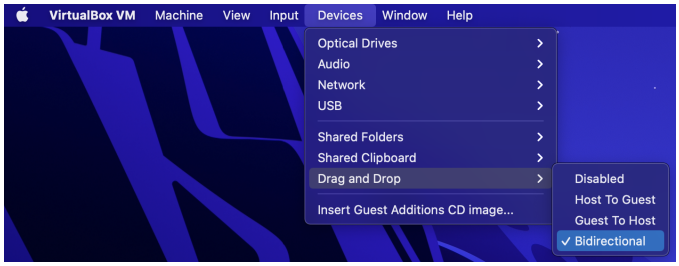
If this method doesn't work for you, you can also try to install using command line: open terminal (Ctrl-Alt-t), then execute the following commands:

```
sudo mkdir -p /media/cdrom
sudo mount -t auto /dev/cdrom /media/cdrom/
cd /media/cdrom/
sudo sh VBoxLinuxAdditions.run
```

After doing so, restart VM to finish the installation.

1.5 Import codes

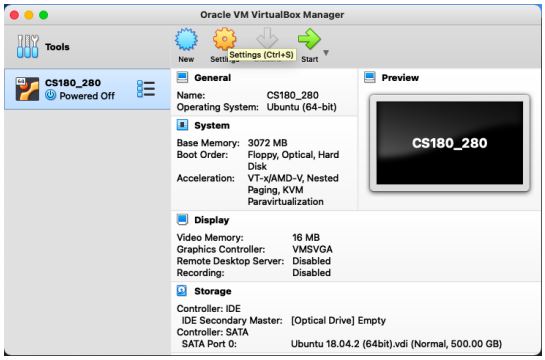
There are many ways to do so. Here is one example: you can directly drag your files from your host operating system into the VM. Make sure the **Drag and Drop** feature is set to **Bidirectional**:

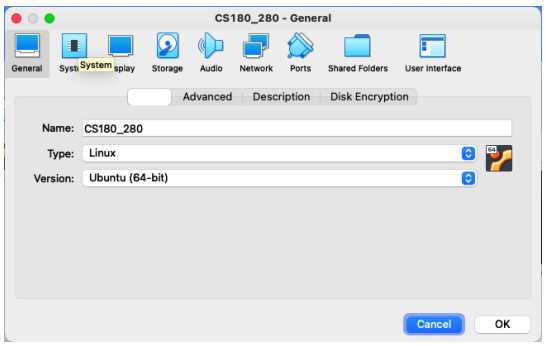


When the code skeleton is imported into the VM, you can view and edit it using Visual Studio Code (VS Code). This should be the default editor of `cpp` files.

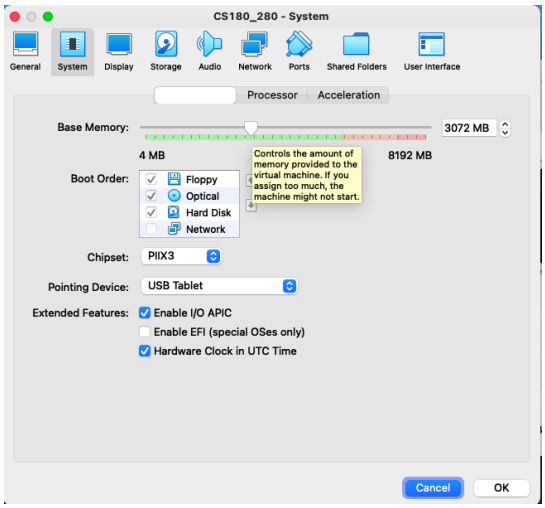
1.6 VM Performance Settings

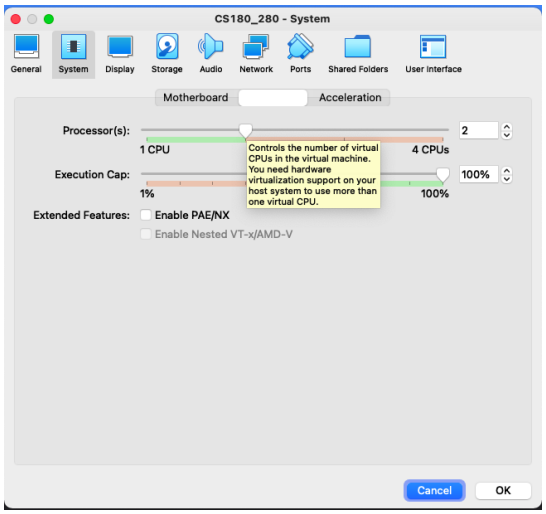
To improve the performance of the VM, power off the VM and then select go to **Settings** (the Gear Icon) in the VirtualBox Manager.





Then in the settings window go to **System** tab then go to **Motherboard** Tab. Increase the base memory to any desired value but be sure not to set it in the red region.





After changing the available memory, increase the available cores by going to the **Processor** tab. Set the Processor to any desired value but again be sure not to set any value in the red region.

2 Code Skeleton

The code skeleton is provided in the sample code `main.cpp`.

2.1 Developing Tools

We recommend you to use Visual Studio Code (VS Code) as the editor and compile and run your codes in the terminal.

After you drag the code into your VM, open VS Code, select **File** → **Open Folder** to open the folder containing the code.

2.2 C++ 101

This subsection provides some basic knowledge about C++ that is relevant to the course assignments. If you want to learn more, please go to `https://devdocs.io/cpp/` or Stack Overflow.

2.2.1 Headers

C++ adopts the convention of using header files to contain declarations. You make the declarations in a header file, then use the `#include` directive in every `cpp` file or other header files that require that declaration. The `#include` directive inserts a copy of the header file directly into the `cpp` file before compilation.

In practice, you can include additional libraries by using `#include`:

```
#include <cmath>
#include <iostream>
```

The above codes include the necessary libraries for C++ input/output and mathematical calculations.

2.2.2 Functions

A function is a block of code that only runs when it is called. The function named with **main** is the entry point of a program.

```
int main() {
    float a = 1.0, b = 2.0;
    std::cout << a << std::endl;
    std::cout << a/b << std::endl;
    std::cout << std::sqrt(a) << std::endl;
    std::cout << std::acos(-1) << std::endl;
    std::cout << std::sin(30.0/180.0*acos(-1)) << std::endl;
    return 0;
}
```

The above program outputs the following calculation results: a, $\frac{a}{b}$, $\sqrt{a}$, $\arccos(-1)$, $\sin(30°)$, where $a = 1$ and $b = 2$, and exits safely.

2.2.3 Common Errors

- Compile Error: try to solve it based on the error message. If you cannot solve it by yourself, you can search the error message on Stack Overflow to find similar cases.
- undefined reference to xxx: usually linking errors. Check if the function is declared in the header file, but has not been implemented in the `cpp` file. Or check the linking configurations in `CMakeLists.txt`.
- Segmentation Fault: usually caused by index out of bounds, too much stack usage.
- Bus Error: the causes are usually similar to the causes of the segmentation fault.
- Math Error: usually caused by dividing it by 0.

2.3 Eigen

This course uses Eigen as the C++ library for linear algebra. Its official documentation can be found at `http://eigen.tuxfamily.org`.

2.3.1 Headers

In order to use Eigen in your project, it needs to be included:

```
#include <eigen3/Eigen/Core>
```

2.3.2 Vectors and Matrices

This part only provides an overview of vectors and matrices operations in Eigen. For a more thorough explanation, please refer to `https://eigen.tuxfamily.org/dox/group__TutorialMatrixArithmetic.html`.

```
    // Example of vectors
    std::cout << "Example of vectors \n";
    // vectors definition
    Eigen::Vector3f v(1.0f,2.0f,3.0f);
    Eigen::Vector3f w(1.0f,0.0f,0.0f);
    // vectors output
    std::cout << "Example of output \n";
    std::cout << v << std::endl;
    // vectors addition
    std::cout << "Example of addition \n";
    std::cout << v + w << std::endl;
    // vectors scalar multiplication
    std::cout << "Example of scalar multiplication \n";
    std::cout << v * 3.0f << std::endl;
    std::cout << 2.0f * v << std::endl;
```

The above code shows the definition, output, addition, and scalar multiplication of 3D floating-point vectors. Please try computing the dot product of two vectors on your own.

```
    // Example of matrices
    std::cout << "Example of matrices \n";
    // matrices definition
    Eigen::Matrix3f i,j;
    i << 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0;
    j << 2.0, 3.0, 1.0, 4.0, 6.0, 5.0, 9.0, 7.0, 8.0;
    // matrices output
    std::cout << "Example of output \n";
    std::cout << i << std::endl;
    // matrices addition i + j
    // matrices scalar multiplication i * 2.0
    // matrices multiplication i * j
    // multiply a matrix and a vector i * v
```

This sample provides the definition and output of matrices. Please explore the usage described in the comments above.

2.4 Code Compilation and Execution

This course recommends you to use **cmake** to build your program. After finishing writing codes, go to the current folder in the terminal window, then run the following commands to compile and run your

program:

```
# Make a folder named build, and enter it. This is the folder that stores all the compiling and linking results.
mkdir build; cd build

# .. means the parent directory. It's the directory containing CMakeLists.txt. This step generates a Makefile using cmake.
cmake ..

# Compile your program. 'Make' will read the Makefile by default. Errors will show up in the terminal if there is something wrong.
make

# Run your program. 'assignment1' is the name of the executable file. You can change it in the CMakeLists.txt.
./assignment1

# Delete all the compiled results before submission.
cd ..
rm -r build
```

3 Problem Set

For problems 2, 3 and 4 go to `main.cpp` provided with the assignment. Add codes in appropriate places.

1. (5 points) Describe what this 2D homogeneous transform matrix does for a point: $\begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

2. (5 points) Write the 3×3 transformation matrix for a $45°$ **clockwise** rotation in 2D (assuming homogeneous coordinates). Populate the `rot_45` matrix variable using `<<` operator in `main.cpp`.

3. (5 points) Write the 4×4 transform matrix to move a point by $(3, 4, 5)$. Populate the `translation` matrix variable using `<<` operator in `main.cpp`.

4. (20 points) In computer graphics, we often need to map points in one rectangle to a new rectangle area. Suppose the bottom left corner and top right corner of original rectangle are $(2, 3)$ and $(4, 5)$. The bottom left corner and top right corner of new rectangle are $(3, 2)$ and $(5, 4)$. This can be achieved by a sequence of three steps:

   (a) Move point $(2, 3)$ to the origin.

   (b) Scale the rectangle to be the same size as the target rectangle.

   (c) Move back points to new position.

   Populate the matrix for each step, and the multiplication result of these matrices in the correct order in `main.cpp`.

4 Submission

Please submit a zip file on gauchospace containing your project (specifically, `CMakeLists.txt` and `main.cpp`) and a report no more than one page.

The report should contain the answer to problem 1, and the matrices populated in problem 2, 3, and 4. You can either write them down, or show the screenshot of your program's output.

Make sure the zip file contains NO compiled results, for example, the files in the `build` folder and the folder itself.