Modeling and Simulation of Appearance

Lab #0 - Introduction to C++

Julio Marco - juliom@unizar.es
Néstor Monzón - nmonzon@unizar.es

2024 - 2025

About this assignment

- Brief overview of the basics of object-oriented programming in C++.

- We just have two hours, thus no in-depth tutorial can be done.

- You may want to (should) (must) refer to some more advanced tutorials:

  - W3Schools: https://www.w3schools.com/cpp/default.asp

  - C++ reference: https://en.cppreference.com/w/, https://cplusplus.com/reference/

  - Others (Google is your friend, copy-paste the error message you are getting)

Requirements for this lab

- Install a C++ compiler.
    - Linux/Mac: `GCC` (`g++`) or `Clang`
    - Windows: `MinGW`

- This lab can be done using just a terminal/CMD/PowerShell
    - You can also set up a graphical IDE (VSCode, XCode, Visual Studio…)
    - There are many tools that can help you (auto-format, error diagnosis, etc.)

A glimpse of this lab…

1/ Compiling and running C++ code.

A glimpse of this lab…

1/ Compiling and running C++ code.

2/ Reading and writing variables. `p0_0.cpp`

A glimpse of this lab…

1/ Compiling and running C++ code.

2/ Reading and writing variables. `p0_0.cpp`

3/ Operations with variables. `p0_1.cpp, p0_2.cpp`

A glimpse of this lab…

1/ Compiling and running C++ code.

2/ Reading and writing variables. `p0_0.cpp`

3/ Operations with variables. `p0_1.cpp, p0_2.cpp`

4/ Arrays. `p0_3.cpp`

A glimpse of this lab… - **Structs**

- Composite data structure that aggregates **variables** of arbitrary types.

A glimpse of this lab… - **Structs**

- Composite data structure that aggregates **variables** of arbitrary types.

```
// Data type Vertex2
struct Vertex2 {
  float x;
  float y;
};
// Define the variable
Vertex2 my_vertex;
my_vertex.x = 0.23;
my_vertex.y = -1.67;
```

```
// Directly define the variable
struct {
  float x;
  float y;
} my_vertex;

my_vertex.x = 0.23;
my_vertex.y = -1.67;
```

A glimpse of this lab…

1/ Compiling and running C++ code.

2/ Reading and writing variables. `p0_0.cpp`

3/ Operations with variables. `p0_1.cpp, p0_2.cpp`

4/ Arrays. `p0_3.cpp`

5/ Structs. `p0_4.cpp`

A glimpse of this lab… - **Classes**

- Building block that allows you to aggregate both **variables** and functions.

- They help represent the behavior of some **high-level element** (e.g., a sphere).

A glimpse of this lab… - **Classes**

- Building block that allows you to aggregate both **variables** and functions.

- They help represent the behavior of some **high-level element** (e.g., a sphere).

```
struct Vertex2 {
  float x;
  float y;
};

// my_vertex is a struct of
// type Vertex2
Vertex2 my_vertex;
my_vertex.x = 0.23;
my_vertex.y = -1.67;
```

```
class Vertex2 {
  public:
    float x,
    float y;
};
// my_vertex is an object
// of class Vertex2
Vertex2 my_vertex;
my_vertex.x = 0.23;
my_vertex.y = -1.67;
```

A glimpse of this lab… - **Classes**

- Building block that allows you to aggregate both **variables** and functions.
- They help represent the behavior of some **high-level element** (e.g., a sphere).
- Classes offer higher **control** over the visibility of the properties of an object.
    - Variables can be private.
    - Functions, which are public, are implemented to ensure *correct manipulations* on variables.

A glimpse of this lab…

1/ Compiling and running C++ code.

2/ Reading and writing variables. `p0_0.cpp`

3/ Operations with variables. `p0_1.cpp, p0_2.cpp`

4/ Arrays. `p0_3.cpp`

5/ Structs. `p0_4.cpp`

6/ Classes. `p0_5.cpp, p0_6.cpp`

A glimpse of this lab… - **Inheritance**

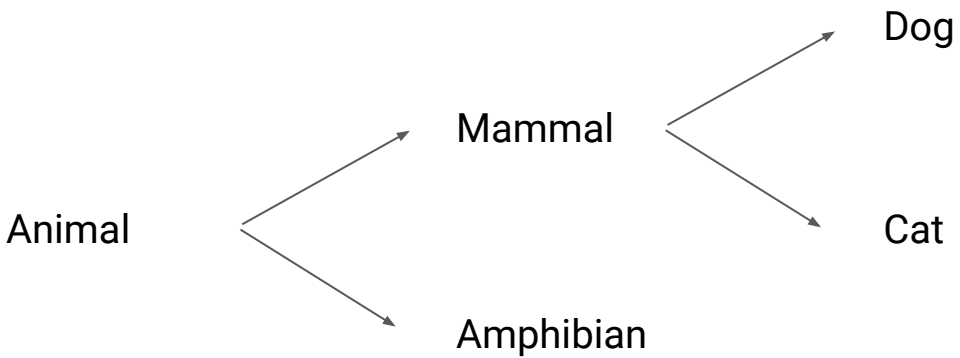
- Mechanism of C++ to re-use, re-define, and specify class definitions.

- Some class may (or may not) **retain** part of the mechanisms of their *parent*.

A glimpse of this lab… - **Inheritance**

- Mechanism of C++ to re-use, re-define, and specify class definitions.

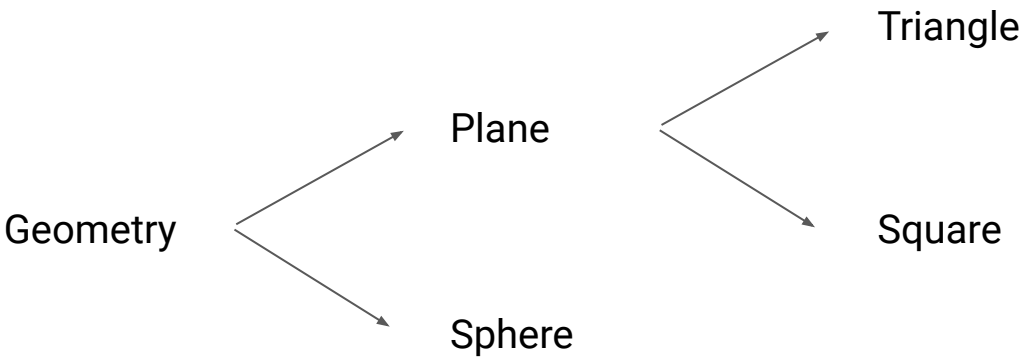
- Some class may (or may not) **retain** part of the mechanisms of their *parent*.



A glimpse of this lab… - **Inheritance**

- Mechanism of C++ to re-use, re-define, and specify class definitions.

- Some class may (or may not) **retain** part of the mechanisms of their *parent*.



A glimpse of this lab…

1/ Compiling and running C++ code.

2/ Reading and writing variables. `p0_0.cpp`

3/ Operations with variables. `p0_1.cpp, p0_2.cpp`

4/ Arrays. `p0_3.cpp`

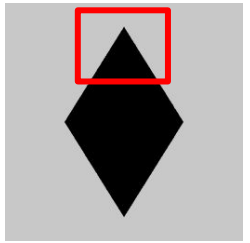
5/ Structs. `p0_4.cpp`

6/ Classes. `p0_5.cpp, p0_6.cpp`

7/ Inheritance. `rhombus_aa.cc, julia.cc`

A glimpse of this lab… - **AntiAliasing**

- Aliasing: when a smooth line appears pixelated due to the display (pixels)
- In the rhombus example, using more samples per pixel alleviates this issue (i.e. it has an anti-aliasing effect)

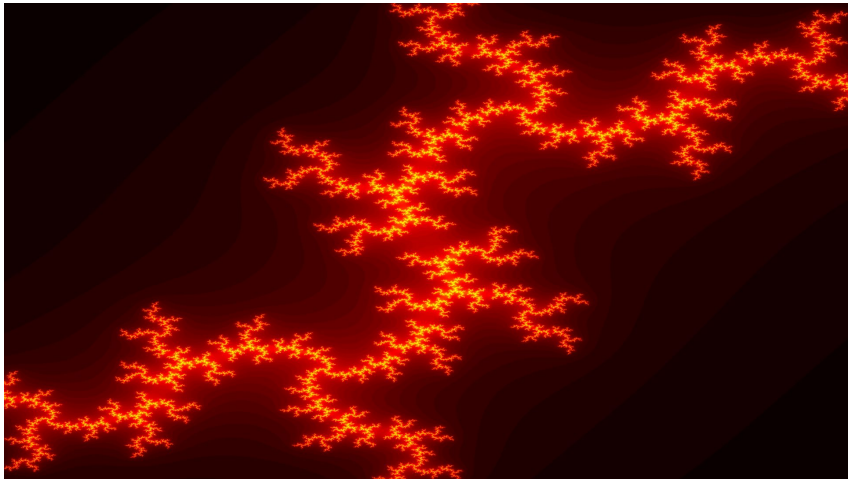


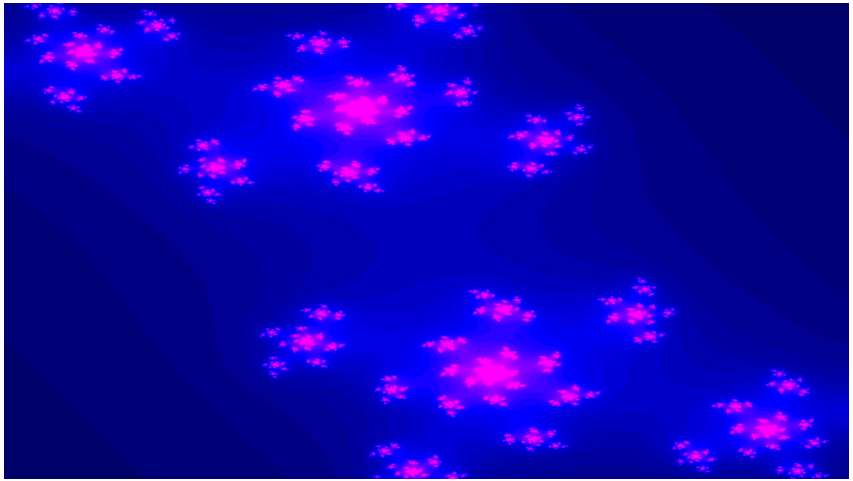
2 samples per pixel

64 samples per pixel

A glimpse of this lab… - **Fractals**

- Use pixel coordinates as input to mathematical formula, and apply it iteratively
- Paint each pixel depending on whether the result diverges to infinity or not
- Produces cool patterns on the computed image
- More info: https://paulbourke.net/fractals/juliaset/





A glimpse of this lab…

1/ Compiling and running C++ code.	
2/ Reading and writing variables.	`p0_0.cpp`
3/ Operations with variables.	`p0_1.cpp, p0_2.cpp`
4/ Arrays.	`p0_3.cpp`
5/ Structs.	`p0_4.cpp`
6/ Classes.	`p0_5.cpp, p0_6.cpp`
7/ Inheritance.	`rhombus_aa.cc, julia.cc`

No submission is required, but we *highly recommend* you to complete all the steps.

Disclaimer (again)

- Remember to read some additional tutorial until you know this kind of things:

What's the difference between these lines?

```
void function (int value)

void function (int* value)

void function (int& value)

void function (const int& value)
```

Disclaimer (again)

- Remember to read some additional tutorial until you know this kind of things:

What's the difference between these lines?

```
void function (int value)

void function (int* value)

void function (int& value)

void function (const int& value)
```

What's the difference between these lines?

```
int number = 64;

int* x_1 = &number;

int& x_2 = *x_1;

int** new_number = &x_1;

x_2 = **new_number + x_2;

std::cout << number << std::endl;
```