ESAT
INNOVATION SCHOOL
ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

Asignatura 2019-2020

Práctica Programación Avanzada

Jose Maria Maestre Quiles & Hector Ochando
Programación Avanzada
Curso 2022/2023
Gustavo Aranda

Índice/Index

Asignatura 2019-2020

ESAT
INNOVATION SCHOOL
ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

1.- What is an algorithm

An algorithm is a sequence of instructions necessary to execute a process. Algorithms are necessary to program any type of application, since they define the flow of the program, the steps to follow, its performance, etc.

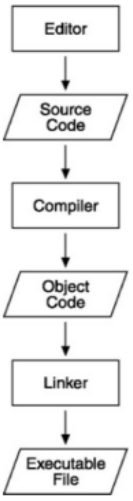
To build an application, we must program these algorithms and execute them in the corresponding order.

1.1.- Run an algorithm

In order to execute a previously written algorithm, we must compile it. In our case, to be able to compile an algorithm in C++ we will need the Visual Studio compiler.

This compiler will take care of converting our C++ code into an executable. To do this we must define the entry point, which will be the main function, which will mark the beginning of the program. Before compiling the code, the compiler will run the pre-processor, taking into account directives like #define, #include, etc. Next, it compiles each of the .cc files by transforming them into machine code, thus creating the files with the .obj extension.

With the object code, the linker is in charge of collecting the signatures of the functions and allowing them to be called from another translation unit, it is also in charge of detecting the main function and generating the executable through the entry point. In the event that some necessary function signatures cannot be found, the linker will give an error, although the compiler will have been executed correctly and we will have our object file.



1.2.- Bubble sort C++

We use the bubble sort algorithm to sort the spheres according to their distance from the camera in order to superimpose them on each other.

```
102  void GameLoop::orderSpheres(){
103
104  // A function to implement bubble sort
105
106      for (int i = 0; i < sphere_size_ - 1; i++){
107          // Last i elements are already in place
108          for (int j = 0; j < sphere_size_ - i - 1; j++){
109            if (spheres[j].sphereOrigin_.z > spheres[j + 1].sphereOrigin_.z){
110              Sphere temp = spheres[j];
111              spheres[j] = spheres[j + 1];
112              spheres[j + 1] = temp;
113
114            }
115          }
116      }
117  }
118
```

To do this, we use a for loop within a for loop, which runs through the entire array of Spheres, in which we check whether each position on the z-axis of the sphere is greater than its next position.

```
102  void GameLoop::orderSpheres(){
103
104  // A function to implement bubble sort
105
106      for (int i = 0; i < sphere_size_ - 1; i++){
107          // Last i elements are already in place
108          for (int j = 0; j < sphere_size_ - i - 1; j++){
109            if (spheres[j].sphereOrigin_.z > spheres[j + 1].sphereOrigin_.z){
110              Sphere temp = spheres[j];
111              spheres[j] = spheres[j + 1];
112              spheres[j + 1] = temp;
113
114            }
115          }
116      }
117  }
118
```

If so, we exchange these values by creating a temporary Sphere variable.

```
102  void GameLoop::orderSpheres(){
103
104  // A function to implement bubble sort
105
106      for (int i = 0; i < sphere_size_ - 1; i++){
107          // Last i elements are already in place
108          for (int j = 0; j < sphere_size_ - i - 1; j++){
109              if (spheres[j].sphereOrigin_.z > spheres[j + 1].sphereOrigin_.z){
110                  Sphere temp = spheres[j];
111                  spheres[j] = spheres[j + 1];
112                  spheres[j + 1] = temp;
113
114              }
115          }
116      }
117  }
118
```

In this way, each time we move a sphere, we will order them according to their proximity to the camera to render only the ones in front superimposed on the ones behind.

You can see a visual explanation of this algorithm on:

https://www.youtube.com/watch?v=Iv3vgjM8Pv4&ab_channel=kamalyassin

2.- Programming paradigms

There are several ways to guide programming depending on the desired objective. Some of these types are:

2.1.- Procedural programming

This type of programming starts at a specific point but as the code is executed it jumps to different parts of it.

For example, C++ starts execution in the main function, and as it progresses it jumps to the different programmed functions.

2.1.1.- Procedural Programming Example

In the case of procedural programming we can use the entire SDL library as an example, since it is programmed in the C language, although some of the functions take advantage of techniques that resemble object-oriented programming, such as passing by reference to a function a structure that is going to be modified. An example of procedural programming is for example "SDL_RenderCopy()" in this case generates a copy of the texture in the renderer on the entire screen.

```
void Texture::draw() {
  if (renderer_ ≠ nullptr && texture_ ≠ nullptr) {
    SDL_RenderCopy(renderer_, texture_, NULL, NULL);
  }
}
```

2.2.- Object-oriented programming

The concept of this type of programming is to have a class that contains base data, which we intelligently combine with other classes in order to save code.

A class is an abstract concept of an object to conceive programming as a more human idea.

An example would be the car class, which has base data such as number of wheels, maximum speed, acceleration, size, weight, etc..

From this class it can inherit another class such as car, which in addition to having these base data has others such as number of doors, cubic centimeters, type of fuel, etc.

2.2.1.- Object-oriented Programming Example

In the case of object oriented programming we have used different classes, in this case we are representing the RayTracer class, for example the traceRay method called in the "update" method of RayTrace is in charge of calculating the colour of the pixels depending on their position, the ray that would come out of them (in a 3D world) and if it collides with a sphere. It also takes into account the colour as a function of directional light. This function is called screen pixel.

```
for(int i = 0; i<GameLoop::Instance().sphere_size_ && !colisioned; i++){
  pixels_[x + ((height_ - 1) - y) * width_] = traceRay(ray, GameLoop::Instance().spheres[i], colisioned);
}
```

```
Uint32 RayTracer::traceRay(const Ray& ray, const Sphere& sphere, bool &colisioned) {

  oxml::Vec3 lightDir = GameLoop::Instance().globalLigt_.light_direction_;
  lightDir.Normalize();

  //oxml::Vec4 sphereColor(sphere.sphereColor_);
  oxml::Vec4 sphereColor = BlendColors(sphere.sphereColor_,GameLoop::Instance().globalLigt_.color_);
  oxml::Vec3 origin = ray.origin - sphere.sphereOrigin_;
  //oxml::Vec3 sphereOrigin(oxml::Vec3::zero);
  //float radius = 0.5f;
  //printf("Color→ %f %f %f\n",sphere.sphereColor_.x);


  float a = ray.direction.SqrMagnitude();
  float b = 2.0f * oxml::Vec3::Dot(origin, ray.direction);
  float c = origin.SqrMagnitude() - (sphere.radius_ * sphere.radius_);

  float discriminant = b * b - 4.0f * a * c;

  if (discriminant >= 0) {
    float t0 = (-b - sqrtf(discriminant)) / (2.0f * a);
    if (t0 < 0.0f) {
      colisioned = false;
      return 0xff808080;
    }
    //float t1 = (-b + sqrtf(discriminant)) / (2.0f * a);
    oxml::Vec3 hit_point = origin + ray.direction * t0;
    oxml::Vec3 normal = hit_point - sphere.sphereOrigin_;
    normal.Normalize();
    float light = fmaxf(oxml::Vec3::Dot(normal, -lightDir), 0.0f);
    sphereColor *= light;
    sphereColor.w = 1.0f;
    colisioned = true;
    return sphereColor.ToRGBA();
  }
  colisioned = false;
  return 0xff808080;
}
```

2.3.- Event Driven Programming

This type of programming is designed to create components that are continually waiting for something to happen, and when the something they are designed to do happens, they will execute the intended code.

An example of this would be web programming, in which we have components such as buttons, forms, links, etc., which are waiting to be clicked to execute their code.

2.3.1.- Event Driven Programming Example

In the case of our application we have used the events of the SDL library, these events are received through the function "SDL_PollEvent()", once we have the event it is removed from the event queue and we can handle it by finding out what type it is and what it contains inside as it is the union of all the SDL event structures.

```
SDL_Event event;
while (runing) {
  Time::last_time_ = (float) SDL_GetTicks64();
  while (SDL_PollEvent(&event)) {
    ImGui_ImplSDL2_ProcessEvent(&event);
    if (event.type == SDL_QUIT)
      runing = false;
    if (event.type == SDL_WINDOWEVENT && event.window.event == SDL_WINDOWEVENT_CLOSE &&
        event.window.windowID == SDL_GetWindowID(window_))
      runing = false;

      if(event.type == SDL_MOUSEBUTTONDOWN){
        printf("Click\n");
        Camera::mouse_clicked = true;
      }
      if(event.type == SDL_MOUSEBUTTONUP){
        Camera::mouse_clicked = false;
      }
  }
```

3.- Debug process

The debugging process is present in all stages of development of a project, and consists of detecting and solving the different errors that prevent the program from running properly, which may arise in the programming of the code, either by software or hardware.
Generally, a program contains a large number of bugs and glitches that are eventually fixed. In the debugging process, these unwanted events and disasters are removed from the code. In this process, entire programs are tested and run to correct the problem. To solve problems, the debugger allows us to create breakpoints, view and modify memory. This allows you to better identify what the code problems are.

3.1.- Origin

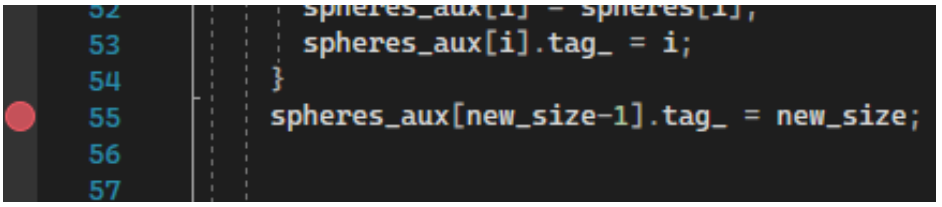
This name was given by the programmer Grace Hopper, who was working for the United States Navy when she discovered that the error she had in her computer was because she had a dead **bug** inside it, so Grace found the first bug in the history.

3.2.- IDE

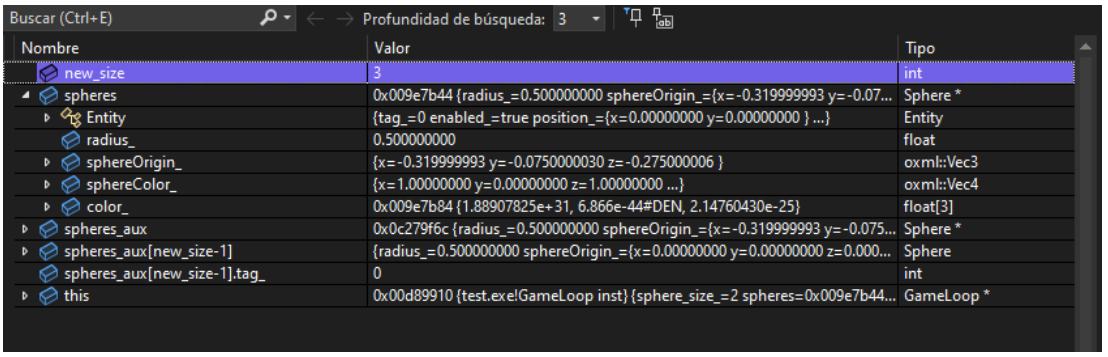
To facilitate the debugging process, there are several tools called IDEs (Integrated Development Environment).

The objective of these IDEs is to increase our productivity by accelerating the process of detecting errors in the code.

For this detection, the IDE allows us to execute our program line by line, being able to even mark interruption points where the program will be interrupted.



These breakpoints are marked by clicking to the left of the line where we want to mark it. When it reaches that part of the program, the IDE will pause the execution of the program and we will be able to see the values of the memory in process:



We can also modify the memory values on the fly to be able to see different results without having to recompile the entire program.

The difference between working with a debugger and not working with it is the time we save having to recompile all the code, how quickly we can find bugs, and the ability we have to see and modify memory in process. Without the debugger, we'll have to print to the screen the values we want to see, which will slow down performance and output speed.

The debugger is responsible for modifying the original binary (it is a copy that is loaded into memory). When a breakpoint is set, the debugger will place a special instruction at the address of the breakpoint. This special instruction must somehow allow the debugger to detect when it is running. This can be some statement that causes some kind of interrupt/exception, which the debugger can hook to, or some statement that handles the runtime control for debugging. This operating system, where you are debugging, must support modifying the running program, with something like poke/peek commands, these commands also allow memory display and modification. The disadvantage of breakpoints is that the debugger must be able to modify the running program.

4.- Code Standard

The programming style that we have used in this practice, in terms of nomenclature, format and good practices, is based on an adaptation of Google's nomenclature and common sense. This nomenclature can be found in
https://google.github.io/styleguide/cppguide.html [1]

This regulation is applicable to code written in C and C++ without external engines, since these have their own nomenclature.

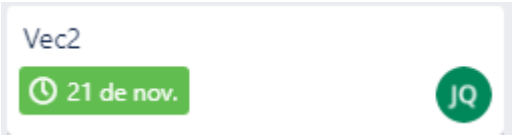
This nomenclature is common in companies, since it defines a common standard for all, making the coordination process easier.

If we did not use the standard in the company, new additions to the team would need time to adapt to the company's nomenclature, so apart from the time we could save, we would lose money. In addition, to communicate between team members, as we all work with the same nomenclature, there was no conflict when naming variables or methods, and the code of the others is much more readable than if each one did it on their own.

Also, when reading code written by other people, it is much easier to read it knowing the standard because you know where each variable comes from and what type it is, for example. Otherwise, you would have to spend more time understanding the code. In the same way, as an individual programmer it is better to follow the standard nomenclature because other people will find it easier to understand your code and may be able to help you more easily or even add some improvements.

5.- Task management system

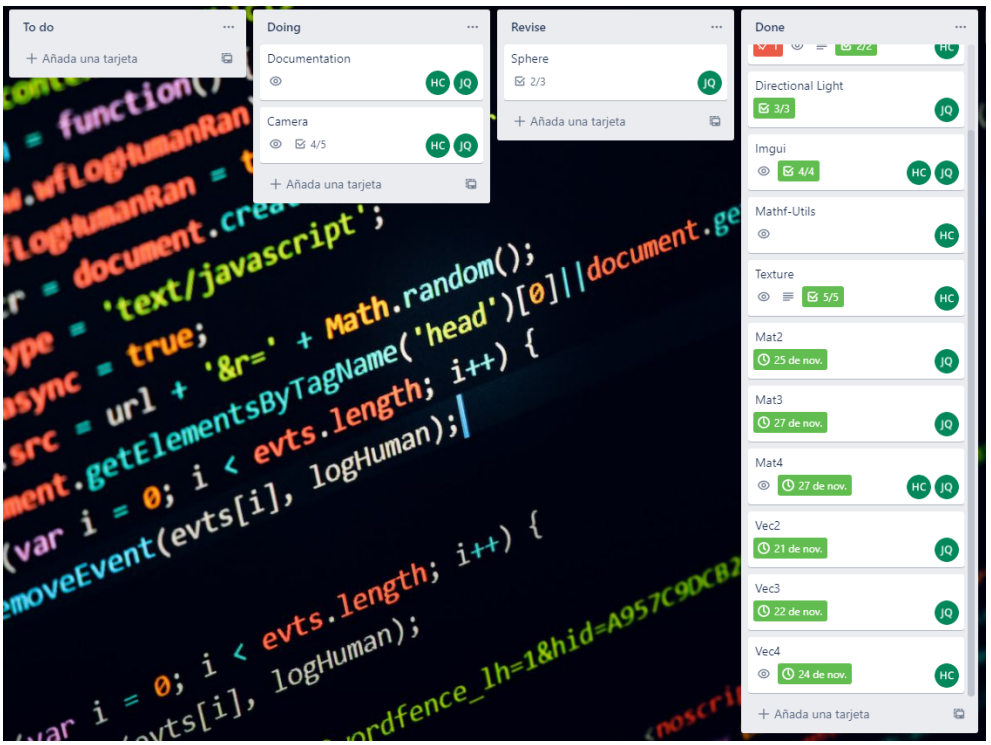
In this practice we have used Trello for task management.
In this Trello, tasks have been divided into the smallest work units under the same topic and assigned to one or more team members. For example "Vec2".



The purpose of each of these cards is to represent everything that has to be done to complete the section correctly. This can range from performing functions, methods, commenting or documenting.

For a better organisation we have classified the tasks in 4 groups:

- To do: In this group are the tasks that have to be done but are not yet done..
- Doing: In this group are the tasks that are being carried out and are not yet completely finished.
- Revise: In this group are the tasks that are finished but not yet checked to confirm that they are well done.
- Done: In this group are the tasks that are done and reviewed.



It has also helped us to organise our time as the tasks had an end date so that the tasks would be finished by that date.

Link to the board:
https://trello.com/invite/b/GeWjBMPX/ATTI2af042ea9aa4ac54f2ec6063298b034d08EE8932/2pa

6- Process and Challenges

All programming involves creating something that solves a problem. Different programming methodologies can be used to plan how the desired software will be developed. There are different types of programming methodologies, but they are all based on planning, design, development and implementation.

- Planning: In this phase, the problem to be solved or the objective is stated.
- Design: In this phase, we consider how the implementation is going to be carried out in order to solve the problem or reach the objective. In this section it is very useful to use pseudocode and/or flowcharts. Good planning is necessary for the easy development of the software.
- Development: In this phase, the software must be developed on the basis of the specifications argued in the design phase.
- Implementation: In this phase the software development should be finished and the implementation will proceed, this can be either the release or the use of the software.

The most important programming methodologies are:

- Waterfall Methodology: In this methodology it is developed in distinct stages and obeying a rigorous order. Before each stage, the code must be reviewed to see if it is ready to move on to the next phase. The initial requirements and specifications are not ready to be changed, so the results cannot be seen until the project is well advanced.
- Prototyping methodology: this is based on building a prototype of software that is built quickly so that users can test it and provide feedback. This allows developers to better understand what the software should do and how it should work, which in turn can help reduce development time and costs. That way you can fix what is wrong and include other requirements that may arise.
- Incremental methodology: In this methodology with each incremental step, new functionalities are added, which allows you to see the results faster compared to the waterfall model. The software can be started even before it is completely finished and is generally much more flexible than other methodologies.

When it comes to code development, many challenges can arise:

- Time Management: One of the biggest challenges in programming is time, with so many deadlines and deliverables, it can be difficult to keep up and finish the whole project on time.

- Communication: This is also one of the most common problems in programming as developers may have difficulties in understanding the needs and specification of the software to be developed. There can also be difficulties in communication within the team itself.

- Scope Creep: This refers to the change of requirements during a project, it is a common problem in development. It occurs when the project starts to evolve beyond what was initially planned. This can have significant impacts on project schedules, as well as generate additional costs and/or delay project completion.

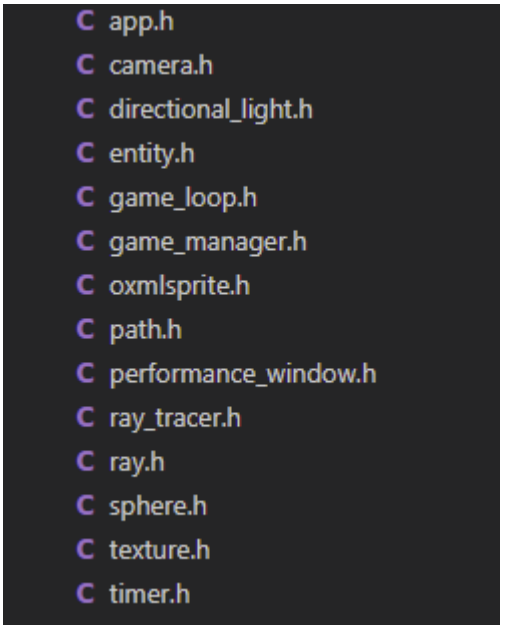
- Low-Quality Code: This problem is based on the fact that writing low-quality code can happen when there is little time, haste, or lack of knowledge of code standards or good practices. It can also happen due to lack of code review.

- Technology Change: This refers to the constant evolution of technologies, both software and hardware. This causes a problem for the developer as it makes it difficult to adapt to these new technologies. It can take time to adapt to new technologies.

7.- Procedural Object-Oriented code

7.1.- Objects

Our application is based on different classes that have base data used by other classes.



The most relevant classes are:

7.1.1- App

This class is a singleton, so there will only be one in memory.

```
8   class App final {
9   public:
10
11   static App& Instance();
12
13   ~App();
```

This class has the information necessary for the rendering of the screen, the creation of the screen and the correct flow of the application.

7.1.2.- Camera

This class contains the information for the correct functioning of the game camera. This camera is used to control the field of view and the rendering of the objects in the game, in this case, the spheres. It contains information about the position, rotation, forward, etc.

7.1.3.- Entity

The entity class contains base information for all objects that are part of the execution of the program. It contains methods for variable initialisation, changing the rotation or position of variables, and so on.

It also contains a tag variable to uniquely identify each entity.

This class is inherited by classes such as:

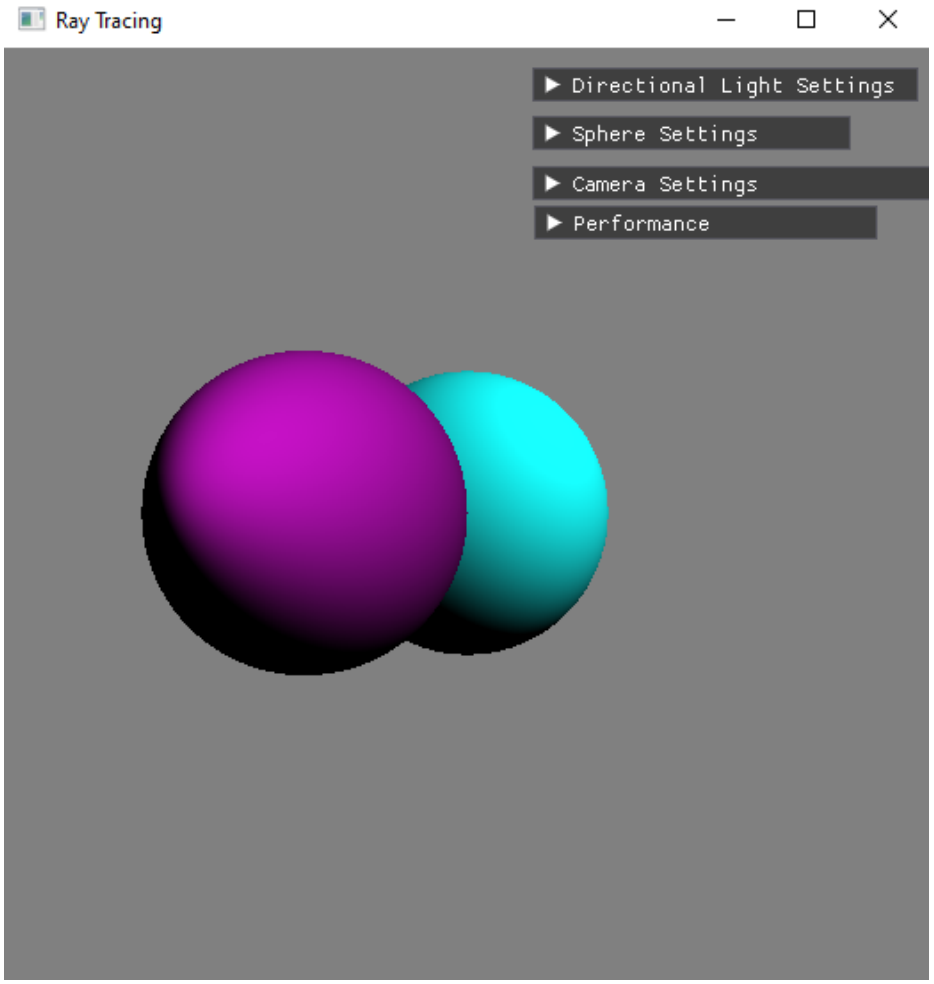
7.1.3.1. Directional light

This class contains the information of the light in the world, such as position, colour, direction of light, etc.

7.1.3.2 Sphere

This class contains information about the spheres to be rendered in the world. Apart from the base information, it contains object-specific information such as radius, colour, origin, etc.

This class is used to define the spheres to be painted:



7.1.4.- RayTracer

This class contains all the necessary information for ray tracing each pixel on the screen, as well as its size and colour. It also contains the onResize method for updating the screen size when resizing and the method for mixing different colours.

7.1.5.- GameLoop

This class contains information for the correct execution of the programme, such as the order of execution, the spheres, the position of the camera, etc.

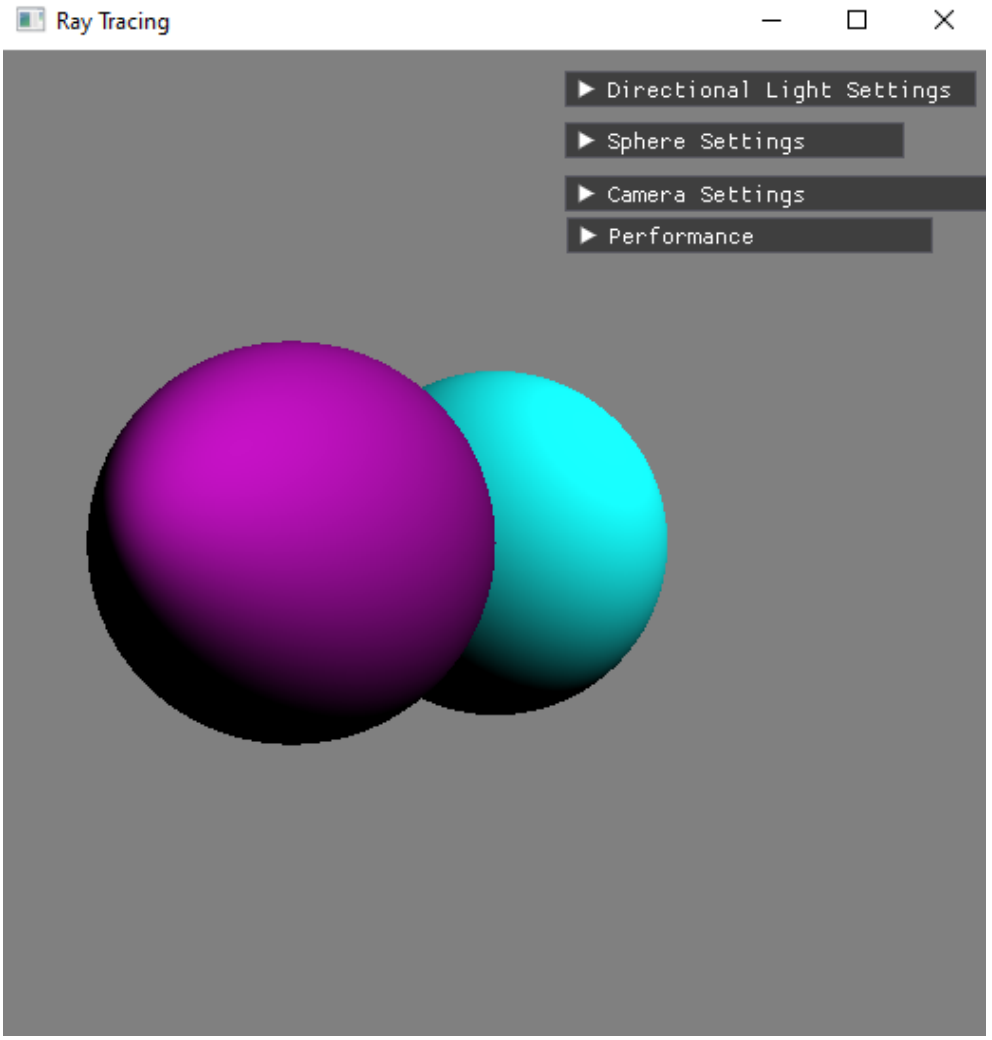
7.2.- Events

In our case, we use events to know the player's input, either to move the camera with the WASD keys or to detect if the player has pressed or released the mouse button.

SDL provides us with functions to detect these events, such as SDL_PollEvent or SDL_GetKeyboardState.

8.- User Manual

To start the application, we must run the executable called "test.exe". Once executed, a screen like this will open:



In it we can find two balls rendered by ray tracing. We can move the camera with WASD

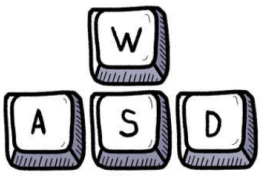
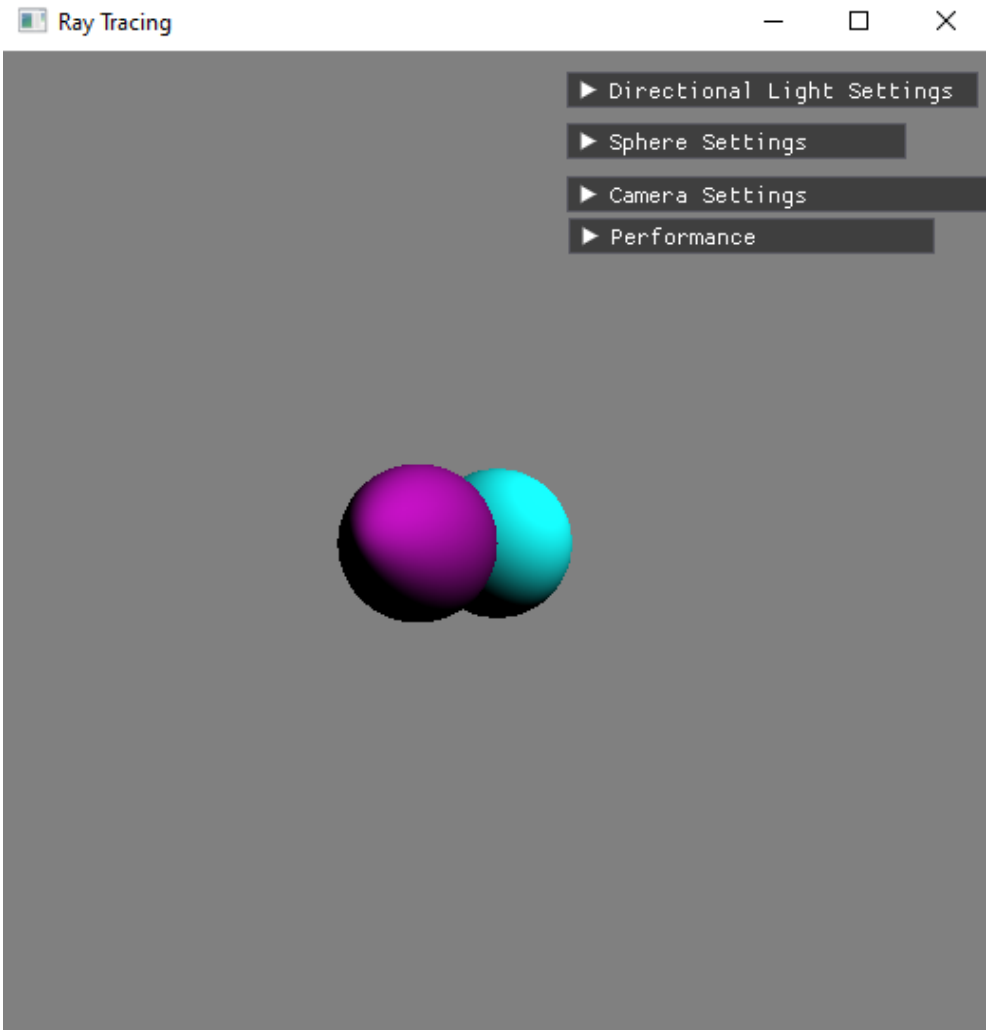


Fig1

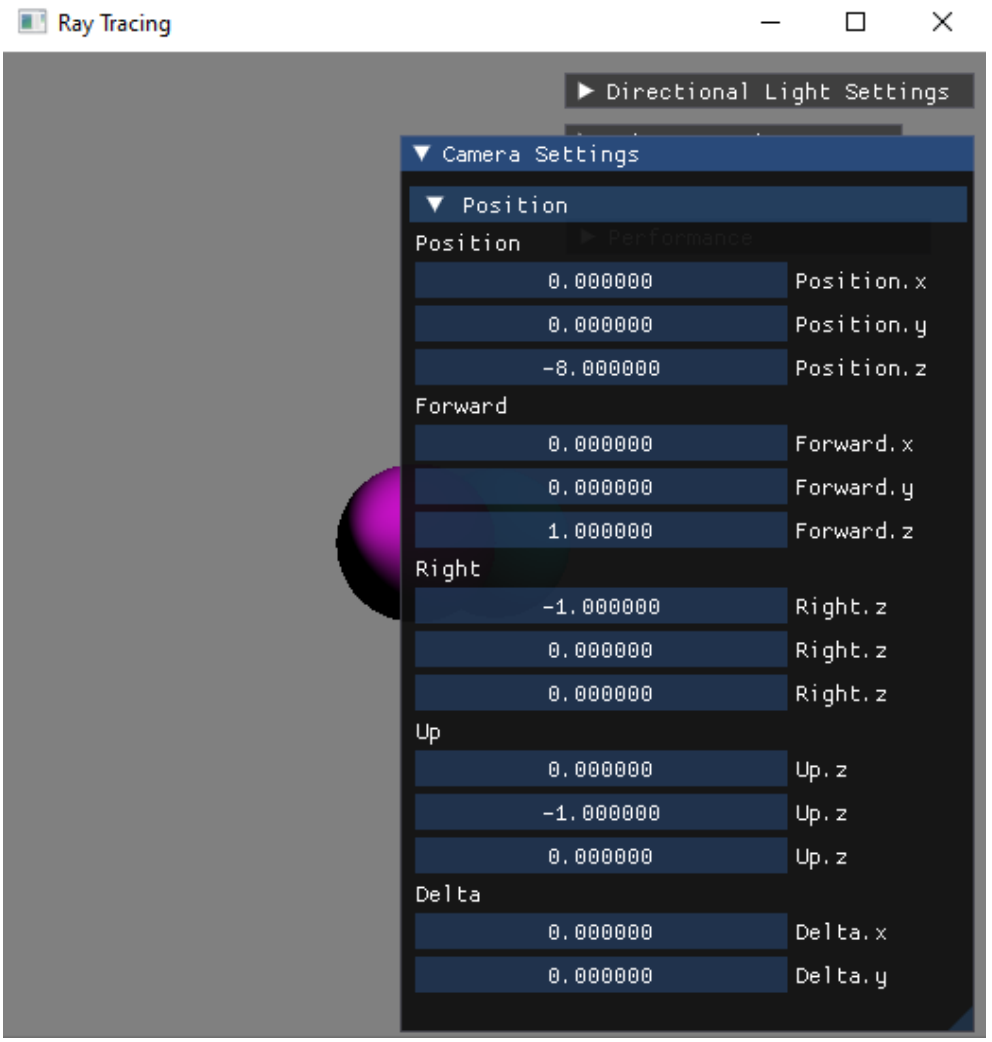
If we move the camera backwards, the balls will look smaller as a result.



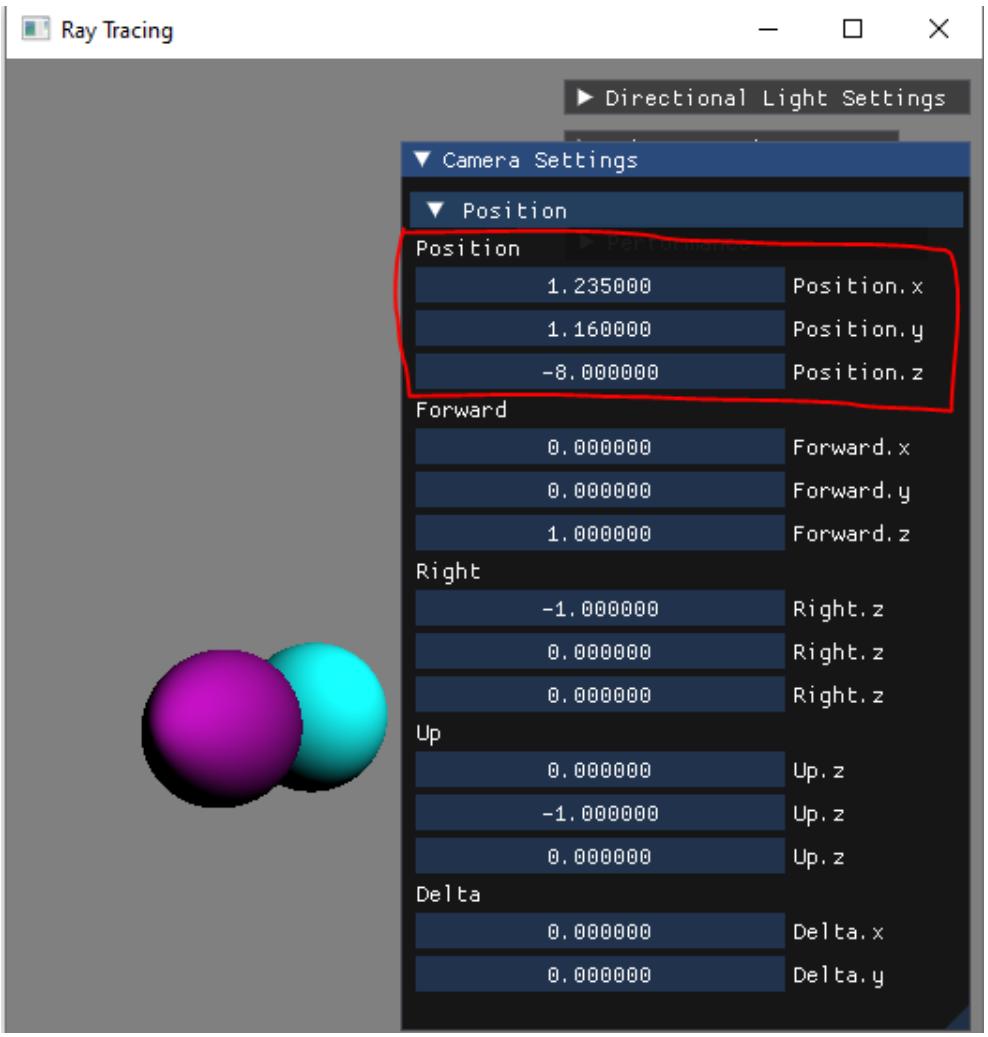
We can find several ImGui menus to view and change running parameters.

8.1- Camera Settings

We will have to click on the menu to display it, in it we will find another menu that we will also have to click on to see all the parameters of the camera:

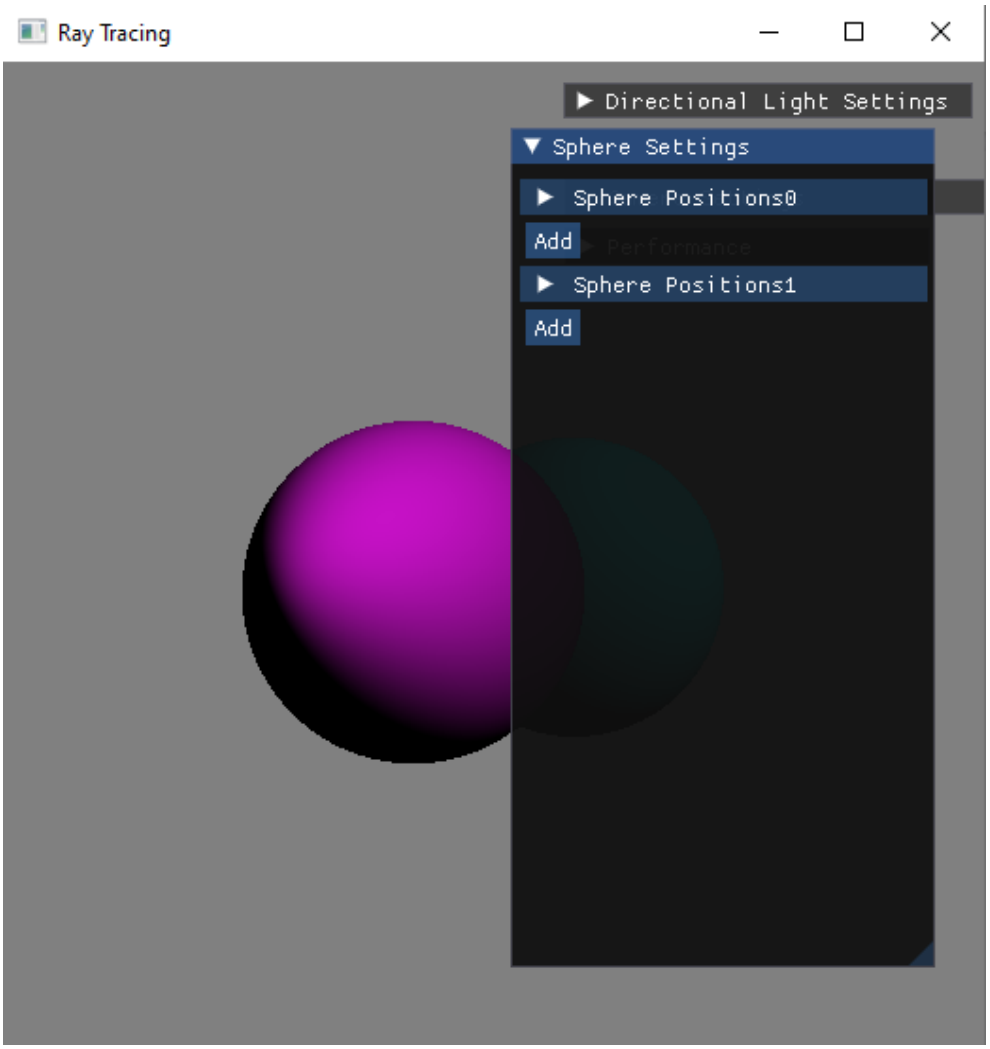


We can modify these parameters by clicking and dragging horizontally on the values.

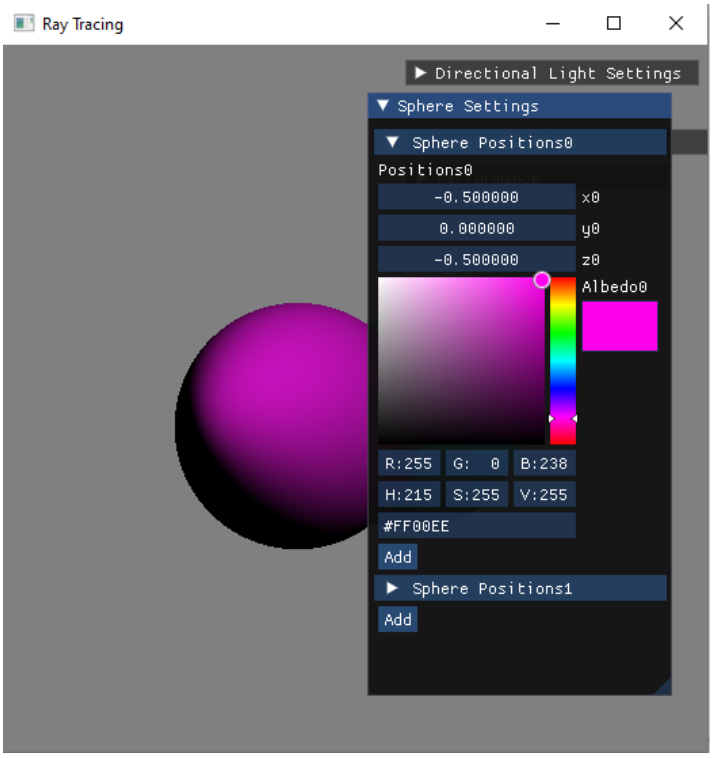


8.2- Sphere Settings

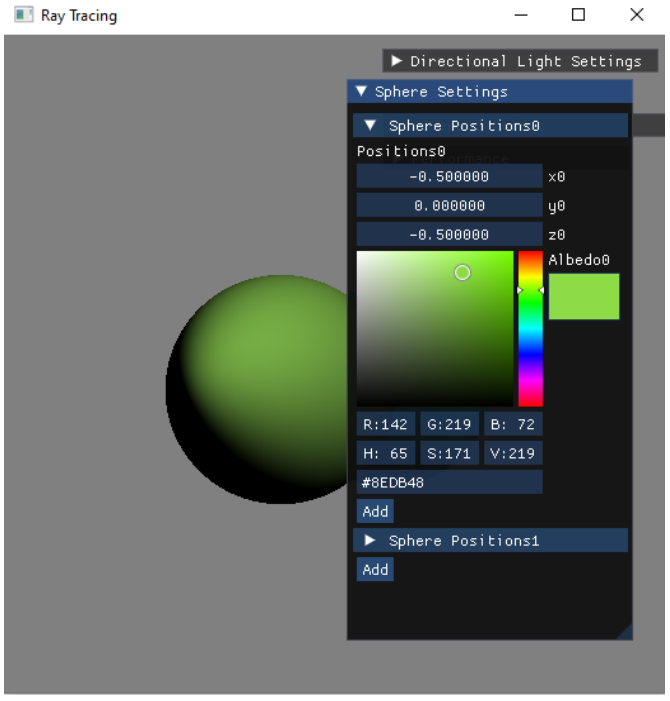
To configure the spheres, click on the "Sphere Settings" drop-down menu, which will open a window with a drop-down menu for each ball and an "Add" button to add a new ball.



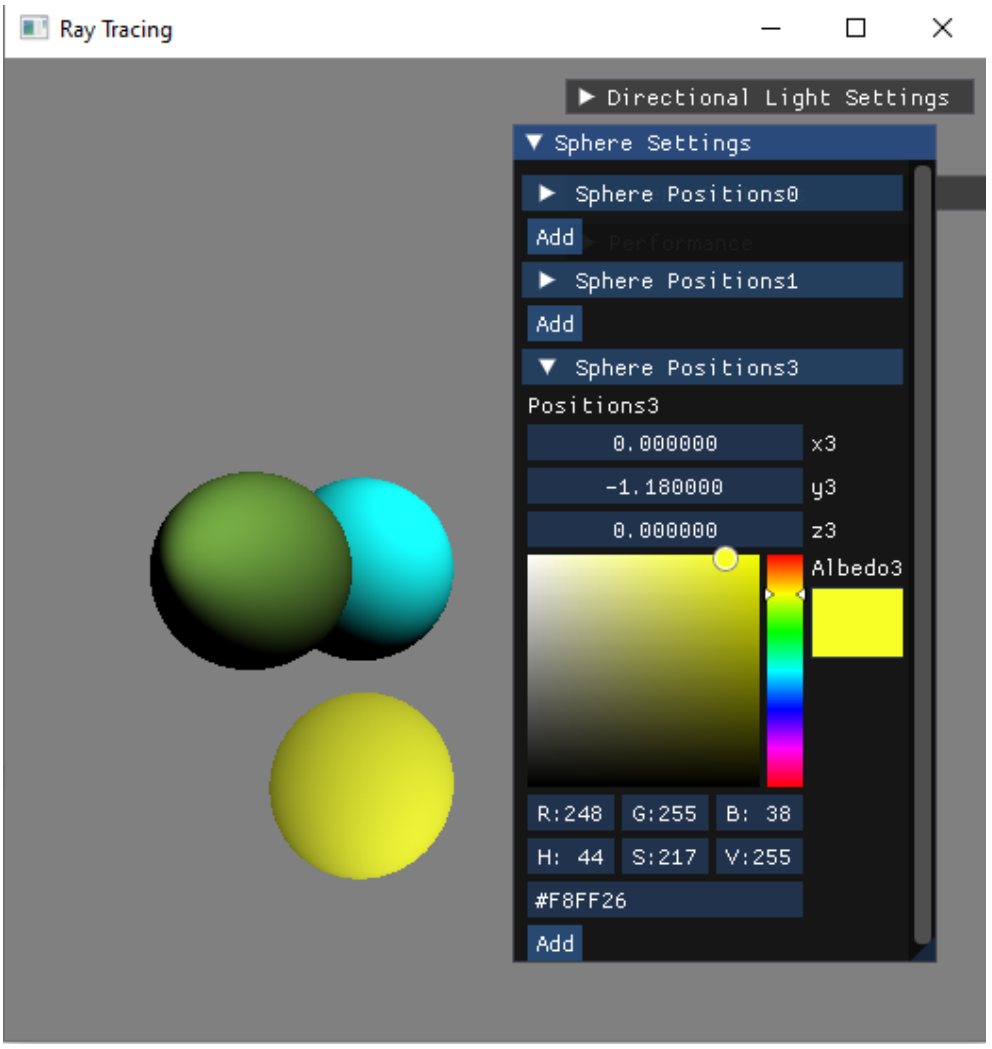
If we open the drop-down menu of a ball, we can see all its values.



We can modify its position by clicking and dragging as with the camera and we can also modify its colour with ImGui's colour picker.

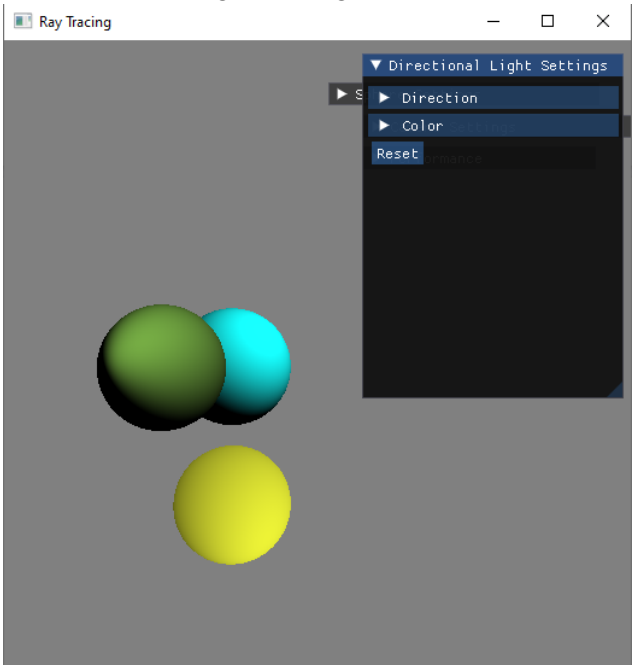


If we click on "Add", we will add a ball with its default values and it will be added to the list of spheres.

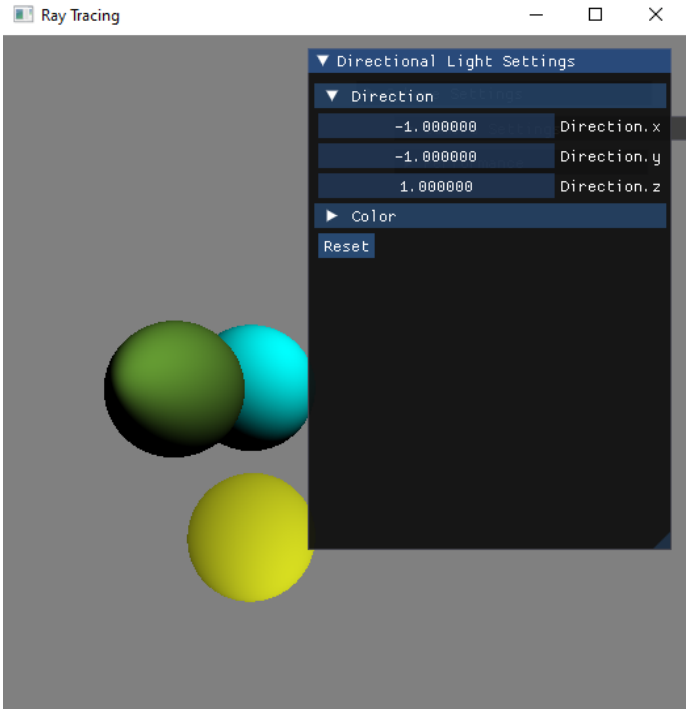


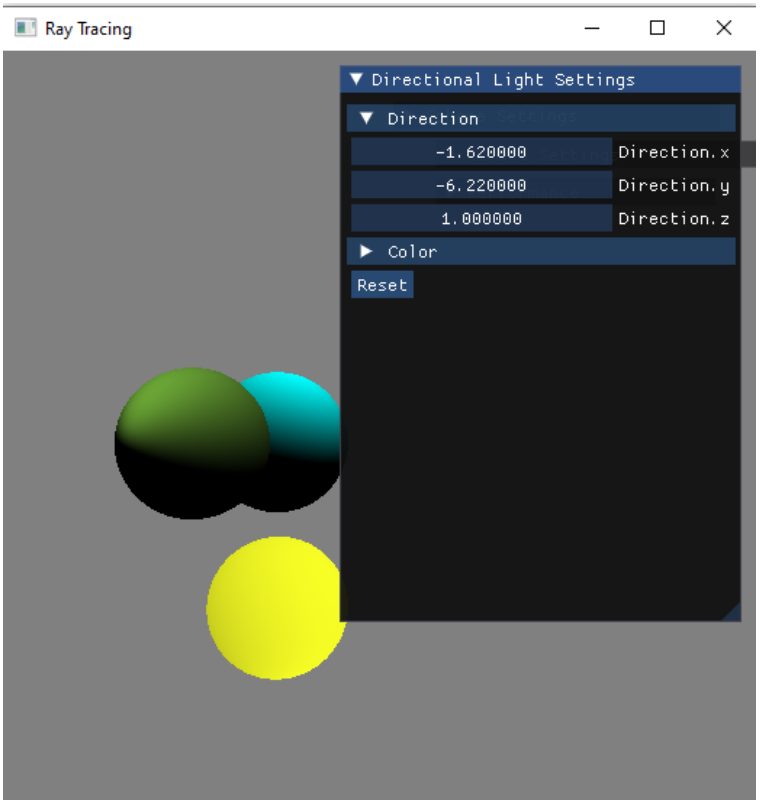
8.3- Directional Light Settings

As with the previous ones, if we click on "Directional Light Settings", a window with the directional light settings will appear.

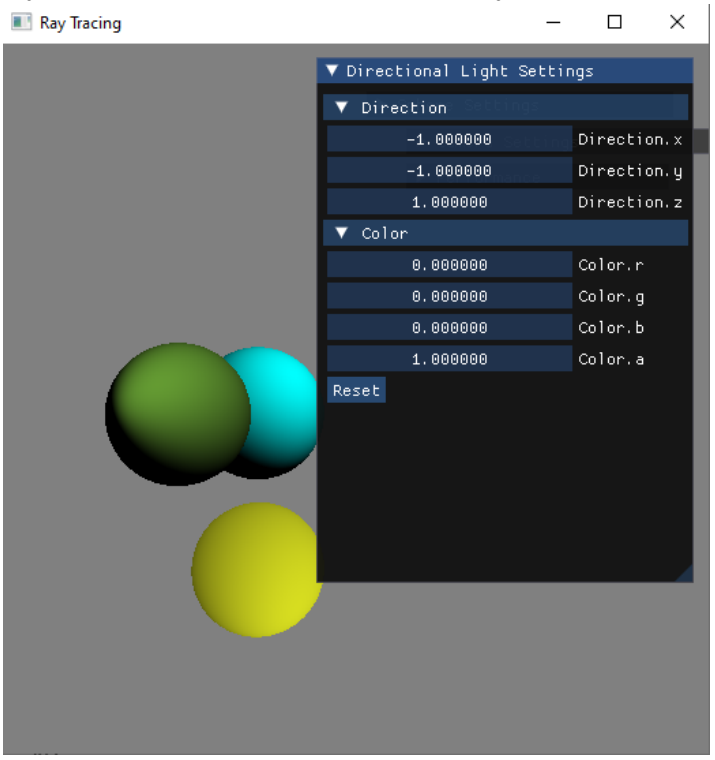


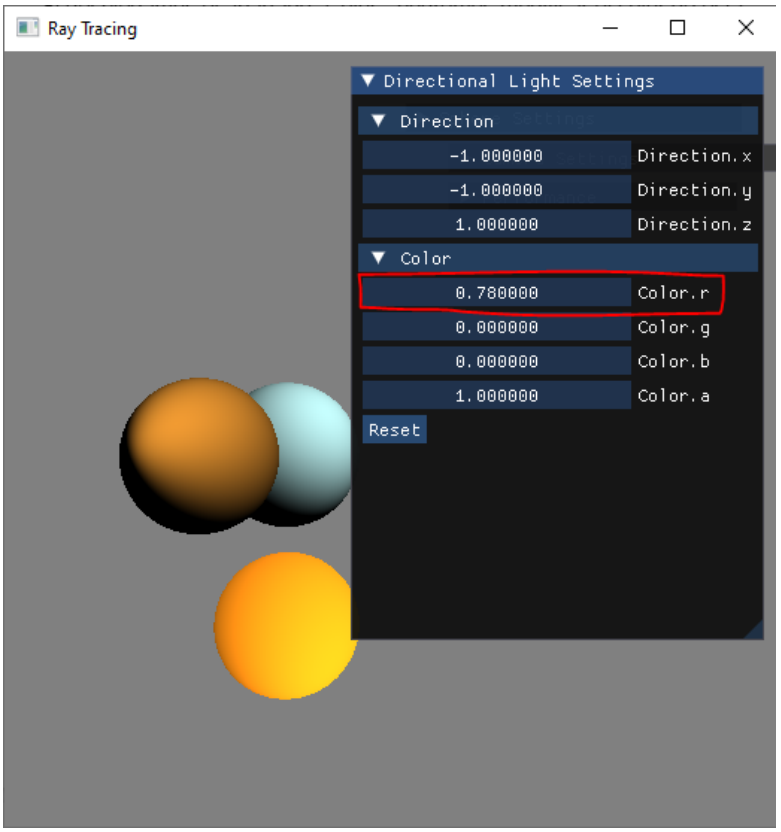
If we click on "Direction" we can modify the source direction of this light.





If we deploy the "Colour" section, we can modify the colour of this light.



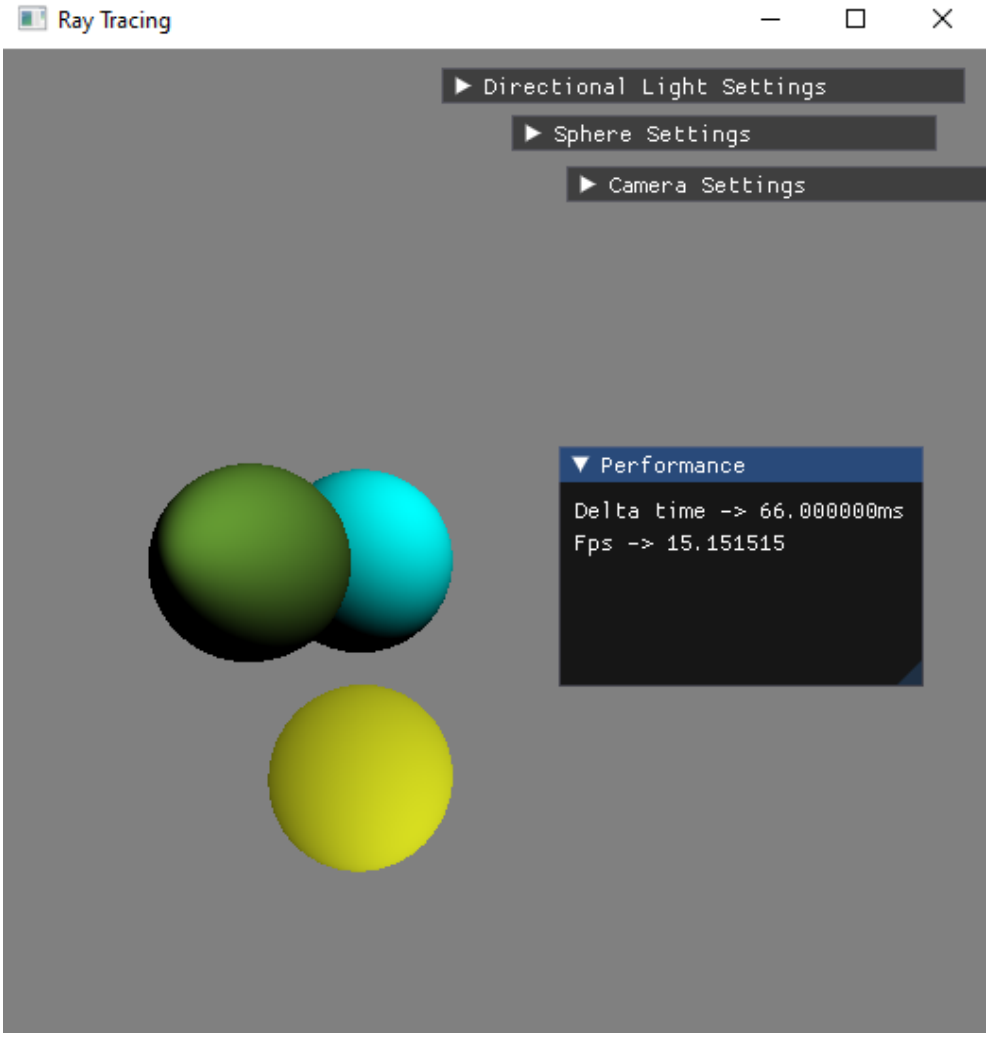


The colour of the light will blend with the colour of each sphere to form a new colour.

There is also a "Reset" button to reset the original colour of the light.

8.4- Performance Settings

In this window you will find information about the execution status of the application, such as Frames Per Second and tick rate.



All ImGui windows can be resizable by holding down the click in the bottom right corner and dragging to the desired size.

9.- Post-Mortem

We have learned a lot in this project about ray tracing on screen and how it works, we have also looked at colours and how light affects them.

We have had some problems with rotating the camera using the mouse, which will be fixed in a future version of the project.

We are also planning to implement a pixel painting buffer to increase performance, as the program is run by the CPU and not the GPU, the performance of the program is considerably reduced.

To make up for this lack of performance, we have opted to temporarily reduce the screen size a little.

We have been helped a lot by a youtube channel called TheCherno, which explains step by step how to do ray tracing on screen.

You can find his channel here:

https://www.youtube.com/@TheCherno/videos

10.- Bibliography

Fig. 1: WASD [digital image] stocklib.com [consulted 19/12/2022]. Available on: https://www.stocklib.com/media-178341168/wasd-keyboard-keys-used-in-pc-video-games-gaming-concept-doodle.html?keyword=wasd

Fig. 2: Google standard [digital information] google.github.io [consulted 19/12/2022]. Available on: https://google.github.io/styleguide/cppguide.html

Fig. 3: Bubble sort dance [digital video] youtube.com [consulted 19/12/2022]. Available on: https://www.youtube.com/watch?v=Iv3vgjM8Pv4&ab_channel=kamalyassin