Boids, flocking behavior simulation

by Riccardo Nisi, Emanuele Rosini, Federico Seren

Abstract

The project simulates coordinated animal motion such as the behavior of bird flocks, based on the computer model developed by Craig Reynolds in 1986 [1]. The program accepts input parameters that regulate the application of flight rules. The basic task of the program is to compute some statistical quantities of the flock in real-time. Additionally, it can render the flock and display the evolution of the flock's statistics over time using graphs.

1 Introduction

The mathematical model consists of a set of three equations that rule the birds motion:

- **Separation** prevents collisions

$$\vec{v}_1 = -s \sum_{j \neq i} (\vec{x}_{b_j} - \vec{x}_{b_i}) \quad \text{if} \quad \left|\vec{x}_{b_i} - \vec{x}_{b_j}\right| < d_s \tag{1}$$

- **Alignment** causes the boids to proceed in the direction of the flock

$$\vec{v}_2 = a\left(\frac{1}{n-1} \sum_{j \neq i} \vec{v}_{b_j} - \vec{v}_{b_i}\right) \tag{2}$$

- **Cohesion** causes the boids to converge towards the center of mass of the nearby ones

$$\vec{v}_3 = c(\vec{x}_c - \vec{x}_{b_i}) \quad \text{where} \quad \vec{x}_c = \frac{1}{n-1} \sum_{j \neq i} \vec{x}_{b_j} \tag{3}$$

The rules are implemented in the program as functions respectively named *separation()* (1), *alignment()* (2) and *cohesion()* (3). The functions return three vectors to be applied as corrections to the boids' velocities. All flight rules are applied considering only the n nearby boids, i.e. whose distance is less than a fixed value d set by the user. The parameters s, a, c and the values of the distance d, d_s determine the influence of the rules and can be provided by the user.

2 Implementation

2.1 Data structures

The data structures used are a struct and a class, respectively named *Point2D* and *Boid*. *Point2D* objects are used in the program to represent position and velocity vectors in a two-dimensional space; they're composed of two floating-point numbers of type float, each one representing a coordinate (x or y) of the vector. A set of free functions that defines the allowed operations between *Point2D* objects is attached to the struct. A *Boid* object contains all the information of a specific boid. The *Boid* class is formed by two *Point2D* objects representing the position and the velocity of the boid. The class is also provided with four methods that allow to get and set the boid vectors. The *get_()* methods parameters are passed by const reference while the *set_()* methods ones by reference. The flock is represented by a std::vector of type *Boid*, which contains all boids information and is the main object manipulated by the program.

2.2 File organization

Files are organized in five translation units and a header file with no cpp. The header file named "point2D.hpp" contains the *Point2D* struct and its related functions defined inline. Three translation units contain all the other functions used in the program. Each one is composed of a header file (which includes the previous ones) with function declarations and a source file with their definitions: "boids.hpp/cpp" contains the *Boid* class and the functions that rule the behavior of the boids, "statistics.hpp/cpp" handles the calculation of flock's statistics and "simulation.hpp/cpp" manages the graphic rendering. The last two translation units contain "boids.tests.cpp", for function testing, and "main.cpp", for the user interface.

2.3 General choices

The majority of the functions have a for-loop that iterates over the std::vector<Boid> object, i.e. the flock. Most of these for-loops are implemented as a range-for but in a few functions indices are required. In all these functions, user-defined types objects are passed by reference while fundamental types objects by value. All functions have been implemented to correctly comply with *const-correctness.* Cppreference [2] has been consulted regarding the use of several standard library functions.

2.4 Basic functions

At the core of the program are several functions that regulate the main movement functionalities of the flock.
To begin with, the *generate_flock()* function initializes a flock, represented by an std::vector of class *Boid* objects. The initial pseudo-random positions and velocities of each individual boid are assigned using a mersenne_twister_engine. The unsigned integer numbers thus extracted are then converted into real numbers within the specified intervals and assigned to n boids, where n is the size of the flock. To represent the parameter n, a floating point number of type double is passed to the function, so that std::floor can be used to ensure that the user selects an integer.
The rules of flight share a similar implementation with each other. The function is called on the flock and on a specific boid object. Each iteration of a for-loop iterates over the flock, updating a sum vector, represented by a *Point2D*. Then, this vector, which can be a sum of position vectors or velocity vector (depending on the rule), is mathematically manipulated as shown in the equations (1, 2, 3) and, finally, multiplied by a factor specific to the flight rule. The rules of flight are made to throw a runtime error if the size of the flock is less than one.
To move the flock, the *movement()* function is implemented, which updates the positions of each boid by adding the velocity scaled by an appropriate factor t, fixed by the developers for optimization purposes.
To determine the behavior of a boid when it reaches the edge of the simulation area, the two-dimensional space has been chosen to be toroidal by imposing the periodicity of the simulation area: if a boid crosses the boundary of the simulation area, it is "teleported" to the opposite side by the *boundary_behavior()* function.
Finally, the *flocking_behavior()* function is implemented to apply flight rules to each boid of a flock. A for-loop fills a std::vector with the velocity corrections, while a second for-loop iterates over the flock to update each boid velocity and assign it. The decision to create a std::vector to handle the velocities corrections is made to update the boids' velocities simultaneously rather than sequentially in each cycle.

2.5 Additional functions

A few additional functions are implemented to make the boid movement more sophisticated and improve the overall quality of the graphical simulation.
The *is_in_field_of_view()* function provides each boid with a field of view, allowing it to see only nearby boids that fall within this field. It has been conventionally decided to prevent a boid from seeing itself or, in the highly unlikely event (with a probability close to zero), from seeing a boid that occupies exactly the same position and has the same velocity. The function is called inside the flight rules, this also allows to discard from the calculations the boid on which the rules are applied. The field of view is an additional parameter to be chosen by the user.
The *random_boost()* function is designed to prevent boids from slowing down excessively without needing to set a minimum speed that could interfere with the flight rules. Additionally, the function introduces a random angular deviation to make the boids movement appear more natural. The parameters of the function have been carefully set by the developers to ensure proper interaction with the fundamental flight rules.
The *speed_control()* function, very similar to boundary_behavior, is implemented to set an upper speed limit to avoid anomalous behaviors.
The *flocking_behavior_two_flocks()* function is an expanded version of *flocking_behavior()* that rules the contemporary motion of two flocks of different species in the same space. It requires two other parameters in addition to s, a, c, d, d_s and the field of view: d_{s_2} and s_2. The first one represents the distance value under which the *separation()* function is activated between the two species while the second one determines the magnitude of such separation.

2.6 Simulation

SFML (Simple and Fast Multimedia Library) [3] graphic library allows to render and display the flock in a window. The *real_to_pixel()* and *calculate_rotation_angle()* functions are used to set the position and the orientation of the shapes.
The *simulation_one_flock()* function plots on a plane the position of the boids in the flock over time. This function uses several objects provided by the SFML library. For example a sf::RenderWindow, i.e. the window where the simulation is displayed, a sf::Texture and a sf::Sprite, where the background image is loaded, some sf::ConvexShape, a class that represents shapes (in this case isosceles triangles), i.e. the boids. The function first generates the window and the flock, then draws the boids with a while loop that lasts until the window is open. Finally, the position and velocity of each boid is updated and the window is displayed. Basically, *simulation_one_flock()* calls all the motion

functions. The *simulation_two_flocks()* function similarly shows the evolution of two flocks on a plane by calling the same functions and types. The only difference is that two flocks are generated and their behavior is managed by *flocking_behavior_two_flocks()*.

2.7 Parameters

Boids' positions are represented by vectors (*Point2D* objects) of two coordinates (x and y), each one always between 0 and 1. For this reason, the maximum legal value of the d parameter is $\sqrt{2}$. The origin of the space is set in the top left corner of the screen. The d_s parameter must be less then d, which is the value of distance over which the birds cannot see each other and therefore cannot interact in any way. The d_{s_2} parameter is allowed to be greater than d to enhance the separation between different species. The parameters s, a, c, s_2 must be non-negative numbers and the field of view must be in the range [0°, 360°].

2.8 Statistics and graphs

The program is capable to show the trend over time of the mean distance of the boids from the origin, of their mean speed and their mean relative distance, with the respective standard deviations. Six functions, declared in the header file named "statistics.hpp" and defined in its corresponding source file, are implemented to compute those quantities. The *graphs()* function draws in a window these values updated in real time, using lots of entities provided by SFML. In order to manage time, this function uses the types sf::Clock and sf::Time. The function first creates the flock and other useful entities, then a while loop begins. In every loop the graphs are created and displayed on the window. The function ends when the window is closed.

3 Interface

3.1 Compilation instructions

The program can be compiled with the help of CMake by following these instructions:

1. create and configure a build area within a directory usually name "build" with the following commands. Use the first one for the debug mode or the second one for the release mode:

```
cmake -S . -B build/debug -DCMAKE_BUILD_TYPE=Debug
cmake -S . -B build/release -DCMAKE_BUILD_TYPE=Release
```

2. compile the area within the build directory with the following commands. Use the first one for the debug mode or the second one for the release mode:

```
cmake --build build/debug
cmake --build build/release
```

3. to check tests results use the following command:

```
cmake --build build/debug --target test
```

4. the following commands execute the program. Use the first one for the debug mode or the second one for the release mode:

```
build/debug/game
build/release/game
```

3.2 User options

After running the program the users can find 7 options printed on the screen:

(a) **Functionalities**: The program prints on the terminal the evolution over time of the boids' mean position (the distance from the origin), speed and relative distance with respective standard deviations. First, the user is asked to set the parameters of flight, later to set the duration of the simulation and the update time on the terminal. **Interpretations**: Suggested parameters for this option are: $n = 2000$, $d = 0.03$, $d_s = 0.005$, $s = 1.0$, $a = 0.2$,

$c = 0.005$, $FoV = 230°$. With such parameters the standard deviation of the speed is observed to progressively fall down to about 0.06, indicating that the flock is well coordinated. Regarding the mean position and the relative distance, even after some time oscillations are still observable. This is due to the toroidal space implementation, as will become clearer with the help of the graphs in options (d) and (e);

(b) **Functionalities**: The program renders the flock motion using the given values. The user can provide the number of boids in the flock, the parameters used in the equations (d, d_s, s , a , c) and the boids' visual field in degrees. **Interpretations**: Numerous attempts have shown that excessively high values of d, besides decreasing the program's performance, also result in unrealistic flock behavior, as the boids exhibit overly forced coordinated movement after a very short time. In any case, it is always appropriate for the parameter d_s to be approximately an order of magnitude lower than the parameter d. Regarding the flight rule factors, good collective behavior has been observed in parameter combinations where s is at least an order of magnitude higher than a and a full three orders of magnitude higher than c. In fact, especially concerning c, a value that is too high results in excessive compaction and overlap of the boids;

(c) The program automatically renders the flock using the suggested values of the parameters chosen by the developers. These correspond to: $n = 2000$, $d = 0.03$, $d_s = 0.005$, $s = 1.0$, $s = 0.2$, $c = 0.01$, $FoV = 270°$;

(d) **Functionalities**: The user can provide all the parameters mentioned in option (b). The program then shows the evolution over time of the boids' mean position, speed and relative distance (with the respective standard deviations) using graphs.
Interpretations: The same considerations made for option (a) apply to this option. Additionally, the graphs make it easier to give an interpretation of the oscillations mentioned in option (a). The introduction of a toroidal space makes the boids "teleport" from one side of the screen to the other. When the flock is formed and the boids approach the bounds of the screen, while they are "teleporting" their mean position and relative distance rise to a peak before decreasing again, as the graphs clearly (fig. 1) show. With certain parameter combinations, the standard deviation of the speed may exceed the maximum y-value set on the graph. It was preferred to keep it this way in order to plot the lower values with higher precision, as they are considered more interesting for statistical considerations;

(e) The program uses the suggested parameters to show the evolution over time of the boids' mean position, speed and relative distance (with the respective standard deviations) with some graphs: $n = 2000$, $d = 0.03$, $d_s = 0.005$, $s = 1.0$, $a = 0.2$, $c = 0.005$, $FoV = 230°$;

(f) **Functionalities**: The program renders the motion of two flocks of different species using the parameters given by the user. In addition to the previous ones mentioned in option (b), the user must provide the parameters s_2 and d_{s_2}, which regulate the separation rule between the boids of different species.
Interpretations: regarding the behavior of boids of the same species, the same considerations made for option (b) apply to this option. A good behavior between the two species can be observed with a value of d_{s_2} two times the value of d and one order of magnitude greater than d_s. With these parameters the avoidance between the two species is well simulated.

(g) The program automatically renders the motion of two flocks of different species using the suggested parameters: $n_1 = 300$, $n_2 = 300$, $d = 0.03$, $d_s = 0.005$, $s = 1$, $a = 0.15$, $c = 0.01$, $d_{s_2} = 0.05$ $s_2 = 1$, $FoV = 270°$.

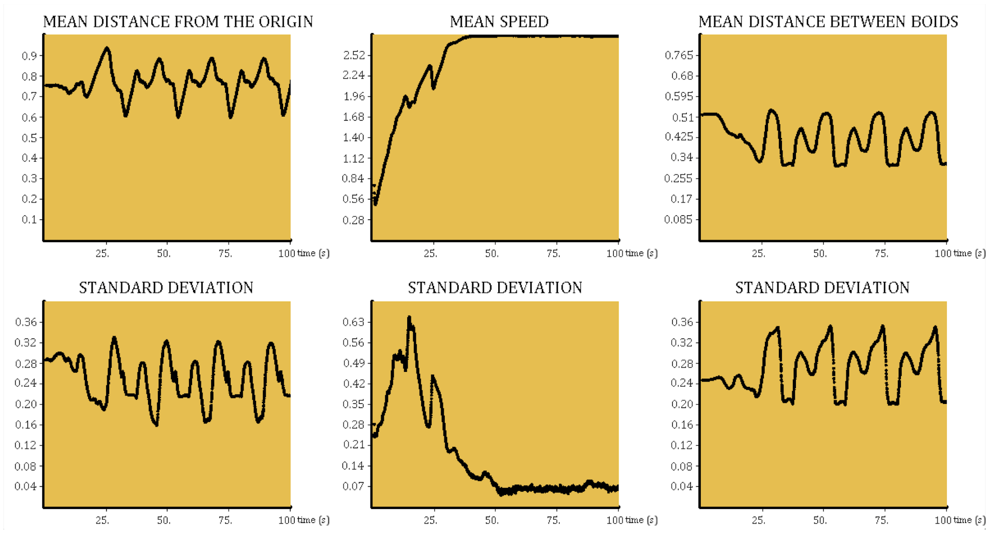

Figure 1: *Graphs displayed by launching option (d) with* $n = 2000$, $d = 0.03$, $d_s = 0.005$, $s = 1.0$, $a = 0.2$, $c = 0.005$, $FoV = 230°$.

3.3 Tests

All functions, with the exception of the graphic simulation ones, are provided with tests that are all included in the "boids.test.cpp" source file. Most of the tests check whether the return of a function in regular situations (i.e. where no exceptions occur) is correct. For example operators and statistics' tests check if the return is equal to the values manually computed. Tests of functions of type void such as *movement()*, *boundary_behavior()* or *flocking_behavior()* check if the position or the velocity of the birds have been correctly modified. Many functions present particular situations that are managed by std::runtime_error and try and catch; for this reason some of the tests are created to check whether the behavior of the function in these situations is as expected. Tests are organized in seven test cases with a total of 261 assertions. All tests are executed using "Doctest".

3.4 Warning and errors

By compiling this program in debug mode, none of the additional warning options written on line 16, 17, 20, 21 of "CMakeLists.txt" generate a message. Some messages are created because of the use of SFML graphic library:

- using Windows Subsystem for Linux, the following message is shown after the window has been created:

  ```
  Setting vertical sync not supported
  ```

  This warning is inevitable using WSL and doesn't affect the functioning of the program [4];

- while using gcc, every time a window is closed, the address sanitizer (-fsanitize=address on line 20, 21 of "CMakeLists.txt") generate an error, like the following one:

  ```
  ==78256==ERROR: LeakSanitizer: detected memory leaks
  ```

  This error has been analyzed using Valgrind [5], a suite of tools for debugging, profiling, and detecting memory errors such as leaks, invalid accesses and uninitialized memory usage, but hasn't been solved. This program does not make use of dynamic allocation and the same error appeared in simpler programs. This suggests that the error may be caused by something external, maybe by SFML or OpenGL.

Appendix

- This project is available on the following git repository:

  ```
  https://github.com/riccardonisi/Boids
  ```

- The project contains an Easter egg that the user must find.

Bibliography

[1] Craig Reynolds website: https://www.red3d.com/cwr/boids/
[2] Cpp reference website: https://en.cppreference.com/w/
[3] SFML website: https://www.sfml-dev.org/
[4] explanation of warning w1: https://github.com/Programmazione-per-la-Fisica/howto/blob/main/other-OSes/XSrvGuide.md
[5] Valgrind website: https://valgrind.org/