EPFL - CHILI LAB

Social experiment in VR using Cellulos and the Oculus Quest

Bachelor Semester Project

Supervised by Barbara Bruno, Arzu Güneysu, and Hala Khodr

Head of Laboratory and supervising professor: Pierre Dillenbourg

Ulysse Ramage

January 7, 2020

CONTENTS

1 Introduction

1.1 The state of Virtual Reality

Since the first release of the Oculus Rift Development Kit 1 back in 2013, often cited as the first headset for Virtual Reality (VR) as we know it, the VR scene has been constantly evolving year to year. New competitors such as HTC, Valve and Samsung have arrived and each new model adds its own features and improvements. But up until now, the market has stayed rather niche: most people interested in the technology don't have VR headsets at home. This is due to many factors, including the high price tag (most headsets range from 500.- to thousands of francs - and a high-end gaming PC is also required to run the games), demanding setup (sensors need to be placed and calibrated in a big room), and overall lack of content.

The Oculus Quest, released in May 2019, has marked a revolution for the VR industry by being the first entirely autonomous and independent headset: no computer nor sensors are required, thanks to a custom on-chip system backed by multiple cameras on the front and on the side to track the exterior environment. It also includes two Oculus Touch controllers used to track the user's hands in space, and in-built handtracking without controllers is even in the works.

1.2 Motivation

The purpose of this project is to develop a virtual reality experience to explore human behaviour within a swarm, using the Oculus Quest VR headset and Cellulo robots developed by the Computer-Human Interaction in Learning and Instruction (CHILI) lab at EPFL. Another VR project was previously led by a student, but the Virtual Reality scene is so recent and has seen so much evolution since last time that it was worth giving it a new try.

The independent and highly portable aspect of the Oculus Quest seemed like a great opportunity for a proof-of-concept of merging Virtual Reality with physical robots in the real world. The Cellulos can also provide haptic feedback and represent an entirely new way of interacting with VR, which the Oculus Touch controllers alone couldn't provide.

Likewise, VR is a great sensory addition to Cellulos; normally, the Cellulo platform was designed to be experienced in the physical world, so the user perception of the environment is global. In VR, the perception of the user can be modified, along with limiting their vision, in order to create a sense of embodiment. The user can feel like it has become the Cellulo robot itself.

2 Controlling the Cellulos with Lua

2.1 The LÖVR framework

Lua is a scripting language often used for games. From World of Warcraft [1] to Garry's Mod [2], the popularity of Lua has risen thanks to its simplicity and native embeddability with its convenient C API. Entire game frameworks have been built on top of Lua, such as the Corona SDK, Defold or LÖVE, which I have personally been using for the past few years to make cross-platform video games.

The actual game development of this project was done with the LÖVR framework. It is an open-source VR game framework inspired by LÖVE and its intuitive API [3], similar to OpenGL. It's written in C and uses Lua as its scripting language, hence being a great option for loading native Cellulo code.

Experimenting with a fairly small, and not commonly used framework such as LÖVR sure provides a lot of freedom compared to engines such as Unity and Unreal, since adding native code is easier and there aren't any pre-defined programming paradigms. But there was a lot of debugging and troubleshooting involved along the project and it would have surely been easier if there were more resources and support from the community.

There are multiple ways to call native code with Lua; two ways in particular, using the Foreign Function Interface (FFI) library, are described below.

The easiest is by loading the native library at runtime.

```lua
local ffi = require("ffi")
ffi.load("cellulo_unity")
```

The second, which is more involved, is to compile the library along the framework / application and present the desired functions through FFI.

```lua
local ffi = require("ffi")
ffi.cdef[[
    int connectRobot(int robot);
]]
ffi.C.connectRobot()
```

This gives an alternative in case the first option doesn't work. It turns out, the first option effectively didn't work on the Oculus Quest, and fortunately the alternative of compiling the library along the framework would function correctly, as will be explained later in the report.

2.2 Porting the cellulo-unity library to Lua

The cellulo-lua library, which represents a big part of the project, consists of a Lua wrapper for a modified version of the cellulo-unity library already developed by the CHILI Lab. The cellulo-unity library [4] was originally made for the Unity3D game engine for use with C# as part of another project; but since it is written in C++ and exposes a C API, it was a better choice to modify it to work with Lua and LÖVR instead of rewriting a new library and try to reinvent the wheel.

- The first modification of the cellulo-unity library was to switch to regular CelluloBluetooth objects instead of CelluloPool, which would let the library connect directly to robots within the game itself without having to communicate with an external application (the Pool Control Panel).
- New functions and utility methods are exposed, such as `getMacAddr` and `getConnectionStatus`, to better handle the connection of the robots on the Lua side.
- The input arguments and return types also had to be adapted for Lua. For instance, returning a list of strings from C to Lua involves fiddling with the Lua stack, so instead the library returns only one string, given an index as an argument.

  Thus,

  ```
  local results = ffi.C.getResults()
  ```

  becomes

  ```
  for i = 1, ffi.C.getResultsLength() do
      local result = ffi.C.getResultAtIndex(i)
  end
  ```

  This is useful in particular for retrieving all the available robots MAC addresses through the Bluetooth scanner.
- A method `newRobotFromMAC` was added to directly connect to a robot given its MAC address, without having to go through the pool.

Once the cellulo-unity library is modified and compiled correctly, the last step is to expose the C methods to Lua using FFI as previously mentioned. Here is an example of exposing the `newRobotFromMAC` method to Lua and using it:

```
/* cellulounity.cpp */
int64_t newRobotFromMAC(const char * address) { /* impl */ }

/* cellulounity.h */
extern "C" {
    int64_t newRobotFromMAC(const char * address);
}

-- cellulo.lua
ffi.cdef [[
    int64_t newRobotFromMAC(const char * address);
    // other methods...
]]
local robot = ffi.C.newRobotFromMAC("00-14-22-01-23-45")
```

Note that the LuaJIT FFI takes care of converting strings and other special types back and forth from Lua to C.

Once the cellulo-core and cellulo-unity libraries are compiled and loaded through FFI, it is possible to establish a Bluetooth connection with Cellulo robots using their MAC address.

2.3 Troubleshooting the environment

Once Qt 5.10.0 was installed, as recommended in the official Cellulo documentation [5], the next step was to build the Cellulo Control Panel application to check if the connection between Cellulo robots and macOS 10.13 would work correctly. At this point, there started to appear many compilation errors, from versions mismatches to dependency issues. Switching versions of Qt multiple times solved some issues and progress was able to be made that way. But one dependency, in particular (epfl-ar), would not compile at all and just throw an error stating "compilation is not supported on macOS". That's when I gave up trying to compile the Control Panel since it would have taken too much time to analyze the codebase and see what was wrong.

When the library was finally working, and although the connection between Cellulos and macOS could be made, Bluetooth packets could only be received by macOS, not sent (due to a change of API in macOS 10.13 [6]). Since the problem was only linked to macOS and not Android, I thought it wasn't worth spending more time on the issue. Instead, I decided to directly go ahead and start developing the Lua interface for the Cellulos for use with the Oculus Quest.

3 Diving into VR with the Oculus Quest

Once the basic functionality of the modified cellulo-lua library was working on macOS, it was time to move to Oculus Quest. The Quest headset runs on Android, which is fortunately supported by Qt. But everything didn't go as planned and a substantial amount of the time allocated for the project was spent on troubleshooting the problems encountered on this part.

3.1 Connection between the Oculus Quest and Cellulo robots

The final version of the cellulo-lua library supports both CelluloBluetooth and CelluloPool APIs. Respectively, a robot can be retrieved in Lua using either `newRobotFromMAC(address)` or `newRobotFromPool()`.

Since Bluetooth functionality cannot be enabled natively on the Quest due to a known issue [7], the Cellulo Pool is used instead.

The final flow for connecting the Cellulo robots to the Quest is to open the Cellulo Pool Server in 2D mode (called "Oculus TV"), start the server, then launch the Cellulo Pool GUI to scan and connect to nearby Cellulo robots in order to add them to the pool. Once this initial setup is done, one can launch the LÖVR app and retrieve the robots from the pool using `local robot = newRobotFromPool()`.

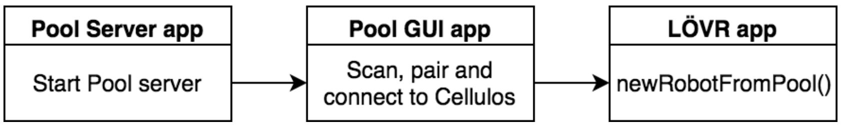

Figure 3.1: Flow of connection between Cellulos and the Quest

3.2 Setting up basic functionality

Getting the connection between the Oculus Quest and the Cellulo robots working was the initial step and foundation required for the game. A basic Graphical User Interface (GUI) and elements which would later be used in the game were first implemented as shown in Figure 3.2.

For this purpose, the lovr-pointer library [8] made by Bjorn Swenson, also author of the LÖVR framework, was used. After experimenting with it, basic 3D buttons were working and interactable with the Oculus Touch controllers.

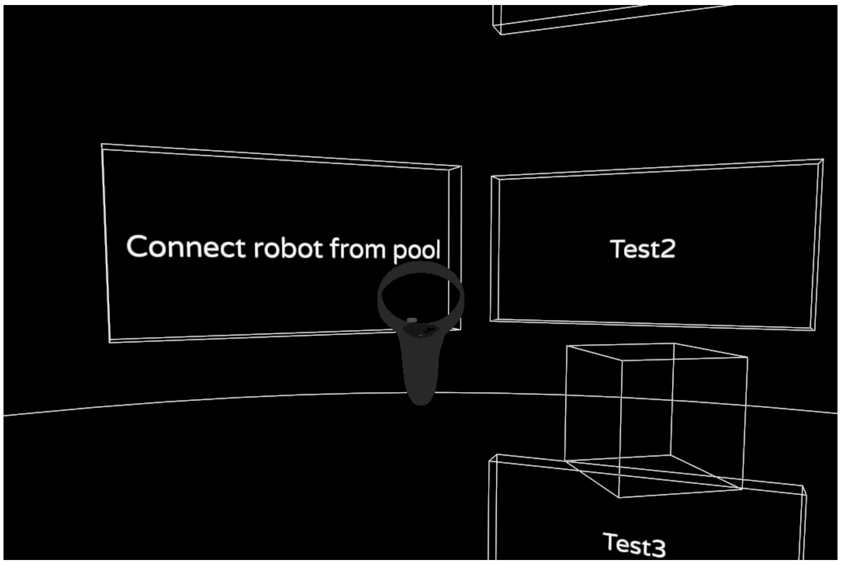

Figure 3.2: Basic GUI - the controller in the middle is used to point at buttons

3.3 Troubleshooting process

The first problem I've encountered, after compiling the libraries for Android, was that the application would crash on any call to `ffi.load` (required to load a shared library). I have reported the issue to the maintainer of the LÖVR Android port and it seems like it is now fixed [9]. But while waiting for the fix, I've decided to work around the issue and compile the libraries directly along the LÖVR Android application.

Upon cloning the `https://github.com/mcclure/lovr-oculus-mobile` repository and adding the Cellulo libraries to the CMake configuration, I was greeted with a "symbol not found" error. After some research, it turned out that the Cellulo libraries I had built with my current version of Qt were 32-bit binaries. The Oculus Quest is based on a 64-bit architecture so I had to find a way to compile the correct binaries. More specifically, the version of Qt I was using only supported android-armv7, and the Oculus Quest required android-arm64-v8a.

As a first attempt, I tried to compile an armeabi-v7a version of the LÖVR Android app (thus

32-bit) and see if it would run on the Quest. To my surprise, it worked but the application would crash every time the library was initialized (by calling `Cellulo.initialize()` in Lua) - this is a required step as it allocates and creates the underlying C objects needed for connecting the Cellulo robots. By analyzing the stack trace of the crash report, I found that it was linked to a SIGSEGV error in cellulothread.cpp - in other terms, the application couldn't correctly create a new thread. A user reported a similar issue on the official Qt tracker [10]; comments from other collaborators made me think this was definitely a problem with Qt, and so I began thinking about workarounds to avoid using Qt with the Quest at all since I was losing time fighting with it.

My first idea was to write a minimalistic local server running on an actual computer (macOS/Linux), which would act as an interface with the Cellulo robots, through UDP sockets. The Quest would connect to the server using the built-in Enet library and indirectly be able to receive and send packets to the robots, through the computer. This setup would get rid of any Qt-related issues on Android altogether.

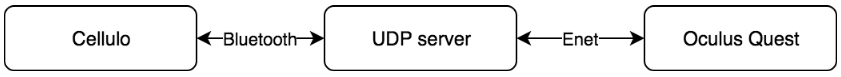

Figure 3.3: Flowchart of the proxy UDP server

The main issue with this setup, though, was the work overhead for the server part, and the fact that this made the portable and independent aspect of the Quest less relevant and beneficial. Fortunately, after thinking about this idea for a few days and beginning to experiment with it, Hala came to me and asked if I had tried to compile the libraries using a more recent version of Qt, even if it wasn't one officially used and supported by the Cellulo team - recent versions of Qt were known to have some problems with Bluetooth and Cellulo.

Following Hala's advice, I have upgraded to Qt 5.12.3, which supports android-arm64-v8a as required. This time, I was able to compile the 64-bit libraries and include them with the 64-bit version of the LÖVR Android application. But the crash upon calling `initialize()` still persisted, and neither Thread nor QtBluetooth objects could be created correctly. This was later fixed by calling `System.loadLibrary()` in the Java application activity, as stated in the documentation of the cellulo-unity project [4]. At this point, the application was running fine but the libqtforandroid.so library would crash with an `java_object == null` error message threw by JNI. The application could be run without the call to `System.loadLibrary()`, but the entire Bluetooth functionality would be disabled (as it was pointed out in the cellulo-unity documentation [7]). Upgrading to Qt 5.13.2 (latest during the project) didn't seem to fix the JNI error. The application and library altogether was very close to working, only the Bluetooth functionality was missing. That's when Hala pointed out that I could use the Cellulo Pool instead of directly connecting to Cellulos through Bluetooth. Unlike the Cellulo Control Panel that I wasn't able to build on macOS, the Cellulo Pool Server and Pool GUI apps ran well on Android. Fortunately, the Oculus Quest is able to run both VR and regular 2D Android apps, and so I've decided to go this path. As a last resort, I adapted the cellulo-unity library back to support both CelluloPool and CelluloBluetooth APIs.

4 Making the social experiment

The goal of the experience is to place the user inside of a swarm of fishes in an ocean and observe how they behave as part of that group. Through this interaction and perception of reality, the goal of the experience is to make the player feel like they embody and become the fish.

4.1 Levels of the game

There are different levels of gameplay, each introducing new behaviours, objectives and obstacles.

- The first level, similar to a tutorial, has the player try to blend within the swarm of fishes and follow it at the right pace, without getting too far away from it.
- Once the swarm has reached its target position, the second level begins. It is similar to the first level, but adds fog around the player to reduce their field of vision, and hence forcing them to rely more on the movement of the surrounding swarm fishes.
- The third level introduces sharks as predators that the player and the swarm should avoid by influencing each other's position and velocity.

4.2 Cellulo input details

The user can move by holding the Cellulo and moving it across the paper board that was designed for the game. The user can also rotate the Cellulo, which will adjust their orientation within the game as well.

The movement of the Cellulo on the board can be retranscribed by absolute or relative positioning in the game, according to the user's preferences and first intuition - different people have different reactions to this new way of interaction.

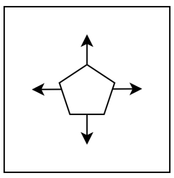
Figure 4.1: Movement of the Cellulo on the game board

- In absolute positioning mode, the position of the Cellulo on the board will be exactly that of the fish in the game.
- On the contrary, when relative positioning is enabled, moving the Cellulo will affect the velocity of the fish, and not its position directly.

The game takes advantage of the Cellulo robots' haptic feedback API to let the user feel the swarm through subtle change in velocity. Some force is applied on the hand of the user while they are holding the Cellulo controller to mimick getting closer or further from the surrounding fishes, similar to feeling water current.

4.3 Swarm behaviour

The swarm behaviour of the fishes was implemented using a similar algorithm as the original Boids program written by Craig W. Reynolds back in 1986 [11]. It is a model of swarm behaviour still commonly used as of today.

Each agent in the swarm (or "fish" in the context of the game) follows three simple rules: separation (avoid other agents), alignment (align direction with surrounding agents) and cohesion (move towards the average position of surrounding agents). In addition to these three rules, predator avoidance has been added for the third level with sharks. The basic concept of the algorithm is described on resources online [12] [13] and depicted in Figure 4.2.

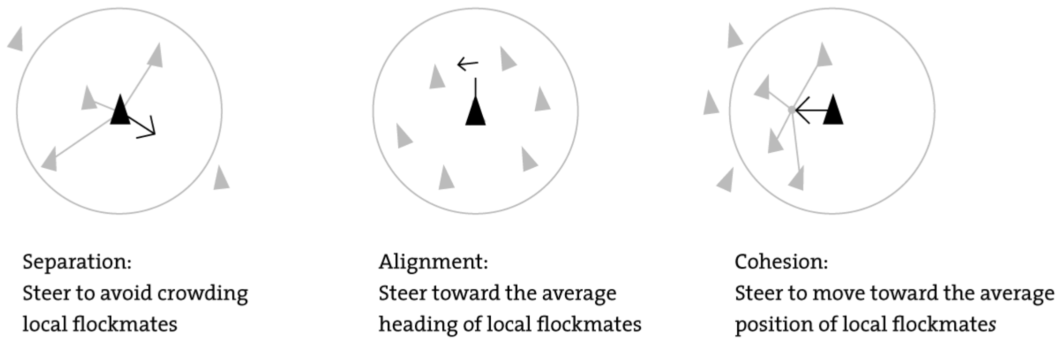
Separation:
Steer to avoid crowding local flockmates
Alignment:
Steer toward the average heading of local flockmates
Cohesion:
Steer to move toward the average position of local flockmates

Figure 4.2: The three basic rules of Boids behaviour

5 Conclusion

5.1 Summary

This project was a great opportunity to introduce a new way of interacting with VR thanks to Cellulos. The Oculus Quest has proven to be a device very much suited for this kind of experiment thanks to its portability and convenience.

Although most of the time allocated for the project was spent on debugging and technical issues, the connection between the headset and the robots is now fully working thanks to the modified library. It will hopefully pave the way for new experiments and experiences with Cellulos and VR in the future.

5.2 Acknowledgments

I would like to sincerely thank the CHILI Lab for the great work atmosphere that I had the chance to be part of, and the opportunity I was given during this semester.

The Cellulo team, notably Barbara Bruno, Arzu Güneysu and Hala Khodr kindly heard and accepted my project proposal, although it wasn't listed on the CHILI website, and gave me the freedom to work on this VR experience. It was a pleasure to discuss about the project and meet for brainstorming sessions together. I would like to thank again Hala for helping me with the issues I had with Qt, since it was my first time working with it. I would probably still be stuck on installing it if it wasn't for her.

Bjorn Swenson, author of LÖVR, and Andi Mcclure, developer of the LÖVR Android port, have both been very helpful and reactive despite my many questions and issue reports.

5.3 Project Repository

`https://c4science.ch/source/cellulo-lovr/`

This repository contains all of the LÖVR game code. It also has precompiled binaries of the cellulo and cellulo-unity binaries.

`https://c4science.ch/source/cellulo-lua/`

This fork of the cellulo-unity [4] library contains the code for the modified version of the library used in this project.

REFERENCES

[1] https://wowwiki.fandom.com/wiki/Lua.

[2] https://wiki.garrysmod.com/page/Beginner_Tutorial_Intro.

[3] https://love2d.org/wiki/Tutorial:Baseline_2D_Platformer.

[4] https://c4science.ch/source/cellulo-unity/.

[5] https://c4science.ch/w/chili-epfl/cellulo/cellulo-software/cellulo-installation-macos/.

[6] https://github.com/eelcocramer/node-bluetooth-serial-port/issues/169.

[7] https://c4science.ch/source/cellulo-unity/browse/master/README.md$75.

[8] https://github.com/bjornbytes/lovr-pointer.

[9] https://github.com/mcclure/lovr-oculus-mobile/issues/10.

[10] https://bugreports.qt.io/browse/QTBUG-69523?focusedCommentId=437155&page=com.atlassian.jira.plugin.system.issuetabpanels%3Acomment-tabpanel#comment-437155.

[11] https://fr.wikipedia.org/wiki/Boids.

[12] https://www.red3d.com/cwr/boids/.

[13] http://www.harmendeweerd.nl/boids/.