Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials

Manipulació d'Objectes en Aplicacions de Realitat Virtual - Leap Motion

MEMÒRIA

Autor: Iriondo Soler, Ana Cinta
Director: Susin Sanchez, Toni

Convocatòria: Febrer 2017

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

Aquest Treball de Final de Grau anomenat *Manipulació d'Objectes en Aplicacions de Realitat Virtual*: *Leap Motion* té com a objectius, d'una banda, investigar les noves tecnologies d'interacció persona-computador amb la funció de traslladar a l'usuari a una experiència més profunda i propera a la realitat en l'àmbit del joc. D'altra banda, crear una aplicació que contingui escenes 3D en les quals el propi usuari pugui interactuar amb els objectes mitjançant la representació de les seves mans.

En els últims temps, el mercat del hardware ha estat portant una àmplia gama de tecnologies basades en noves formes d'interacció de l'usuari amb la computadora. En aquest projecte s'utilitzarà el software *Unity3D* per la creació de l'aplicació i el dispositiu *Leap Motion* pel control i seguiment de les mans de l'usuari. Aquest dispositiu està dissenyat per al reconeixement 3D gestual de les nostres mans. Per tant, la versatilitat que ofereix aquesta tecnologia és molt gran, ja que hi ha moltes aplicacions de camps molts diferents en les quals l'ús de gests pot facilitar la interacció persona-màquina.

La implementació d'aquest projecte es divideix en diverses línies de desenvolupament: s'han dissenyat 4 escenes virtuals, de les quals cadascuna té una finalitat diferent, per demostrar que la connexió de *Unity* amb el *Leap Motion* és possible. La primera escena s'ha programat per que el *Leap Motion* detecti el que es coneix amb el nom de "*pinch*", o pessic en català, que farem amb les mans per agafar objectes i moure'ls de lloc. Les altres tres escenes, en canvi, s'han configurat per que en detectar un gest determinat, ja sigui el moviment d'un o varis dits, del palmell de la mà, o qualsevol altre gest, realitzin una certa acció. Per poder visualitzar les escenes, s'ha dissenyat una interfície d'usuari (UI) amb un menú per accedir més còmodament a l'aplicació.

Un cop finalitzat el treball, es pot concloure que es poden crear fàcilment *Apps* de realitat virtual utilitzant el software *Unity* i el sensor *Leap Motion*, i que existeixen infinites possibles aplicacions en múltiples sectors. S'ha provat l'aplicació i s'ha demostrat que es pot interactuar amb la pantalla de l'ordinador sense haver de fer ús del ratolí o del teclat, simplement amb el moviment de les nostres mans.

Paraules clau: Leap Motion, Realitat Virtual, interacció persona-computador, software, hardware, Unity3D, escena 3D, interfície d'usuari (UI), Apps.

Abstract

This End of Degree Project, called *Manipulació d'Objectes en Aplicacions de Realitat Virtual*: *Leap Motion* has as a goal, firstly, to investigate new human-machine interaction technologies with the purpose of bringing the user to a deeper and closer experience to reality within a playable environment. Secondly, to create an *App* that contains 3D scenes in which the user can interact with objects by representing his hands.

Over the last years, hardware market has been providing a wide range of technologies based on novel ways of human-computing interaction. In this project is used the *Unity3D* software to create the *App* and the *Leap Motion* device for control and monitoring of user hands. This device is designed for gesture 3D recognition of our hands. Therefore, the versatility offered by this technology is great, as there are many applications in many different fields in which the use of gestures can facilitate human-computer interaction.

There have been defined different lines of development for the implementation of this project: there have been designed four virtual scenes, each of which has a different purpose, to show that the connection between *Unity* and *Leap Motion* is possible. The first scene is programmed for *Leap Motion* to detect what is known as a "pinch", to grab and move objects with hands. The other three hands, however, have been configured to detect a particular gesture, or a movement of one or several fingers or any other gesture to perform a specific action. To view the scenes, it has been designed a user interface (UI) with a menu to access the application more easily.

Once the project is finished, we can conclude that Apps can be easily created using the virtual reality software *Unity* and the *Leap Motion* sensor; there are endless possibilities and applications in multiple sectors. The App has been tested and it has been proved that is possible to interact with the screen of the computer without having to use the keyboard or mouse, simply moving our hands.

Key words: Leap Motion, Virtual Reality, human-computer interaction, software, hardware, Unity3D, 3D scene, user interface (UI), Apps.

Índex

RESUM 1

ABSTRACT 3

ÍNDEX 5

1. GLOSSARI 7

Índex de figures 8

Índex de codis 11

2. INTRODUCCIÓ 13

2.1. Objectius del projecte 13

2.2. Abast del projecte 13

2.3. Motivació 14

2.4. Problemes trobats 14

2.5. Fases de desenvolupament 14

3. ESTAT DE L'ART 17

3.1. Motion Sensor 17

3.1.1. Taxonomia Motion Sensor 17

3.1.2. Productes Motion Sensor 18

3.1.3. Interfície Home-Màquina (HMI) 20

3.2. Eines de desenvolupament 22

3.2.1. Unity 3D 22

3.2.2. Leap Motion 23

3.3. Situació actual del desenvolupament d'aplicacions 24

3.3.1. Interfície de programació d'aplicacions (API) 24

3.3.2. Seguiment de les mans en VR 25

4. ANÀLISI DEL SISTEMA 29

4.1. Leap Motion 29

4.1.1. Instal·lació del Leap Motion 29

4.1.2. Característiques tècniques 32

4.1.3. Principi de funcionament 38

4.1.4. Anàlisi de la API 43

4.2. Incorporació del Leap Motion a Unity3D 50

4.2.1. Llibreria SDK de Leap Motion 50

4.2.2. Creació d'un projecte amb Unity 3D 51

4.2.3. Adició de mans a l'escena 51

5. EL MOTOR DE VIDEOJOCS *UNITY 3D* 55

5.1. Unity 3D 55

5.1.1. Interfície de Unity 55

5.1.2. Editor gràfic 57

5.1.3. Estructura i programació 58

5.2. Estructura de l'editor 59

5.3. Conceptes bàsics 59

5.3.1. GameObjects i Components 59

6. DISSENY I CONSTRUCCIÓ DE L'APLICACIÓ 63

6.1. Inici d'aprenentatge 63

6.1.1. Primera escena d'aprenentatge 63

6.2. Descripció del directori de treball 64

6.3. Interfície d'usuari 65

6.3.1. Canvas 65

6.4. Estructura de l'aplicació 66

6.4.1. Escena 1: Manipulació d'objectes 66

6.4.2. Escena 2: Nit i dia 71

6.4.3. Escena 3: Spinning Wheel 74

6.4.4. Escena 4: Point and shoot 77

7. PLANIFICACIÓ TEMPORAL I PRESSUPOST 81

7.1. Fases de desenvolupament 81

7.2. Pressupost 82

8. IMPACTE MEDI AMBIENTAL 83

CONCLUSIONS 85

AGRAÏMENTS 87

BIBLIOGRAFIA 88

Referències bibliogràfiques 88

Bibliografia complementària 89

ANNEXOS 91

Annex 1 : Codi *MouseLook* 91

Annex 2 : Codi *CharacterMovement* 92

1. Glossari

API: Interfície de Programació d'Aplicacions.

Assets: Paquet de dades. Conjunt de fitxers que s'importen a Unity per poder crear les escenes.

Buffer: Memòria d'emmagatzemament temporal d'informació que permet transferir les dades entre unitats funcionals amb característiques de transferència diferents.

Canvas: Objecte de Unity 3D que permet inserir text a les escenes.

C# o CSharp: Llenguatge de programació.

Dev Kit: "Development kit", kit de desenvolupament. Conjunt d'eines que permeten al programador desenvolupar aplicacions per cada tipus de plataforma.

Driver: Controlador, rutina o programa que enllaça un dispositiu perifèric amb el sistema operatiu.

Eye Tracking: Tecnologia que pretén extreure informació de l'usuari analitzant els seus moviments oculars.

Frame: Són una sèrie d'instantànies que contenen dades sobre la posició i altra informació sobre la entitat detectada.

Game Object: qualsevol objecte que pertany a una escena de Unity 3D.

Hardware: Conjunt d'elements físics o materials que formen part d'una computadora o d'un sistema informàtic.

Hierarchy: És la finestra de Unity que conté tots els gameObjects que es troben a l'escena.

HMD: "Head Mounted Display", ulleres de realitat virtual.

IDE: Entorn de desenvolupament integrat.

LeapHandController: És el prefab que dibuixa la posició i l'orientació de les mans a l'escena**.**

Leap Motion: Sensor de moviment que ens permet controlar l'ordinador mitjançant el moviment de les mans i els dits amb alta precisió

LM: Leap Motion.

OpenGL: *Open Graphics Library*, és una especificació estàndard que defineix una API multi-llenguatge i multi-plataforma per escriure aplicacions que produeixen gràfics 2D i 3D.

PC: Ordinador.

Prefab: És un prefabricat que actua com una plantilla a partir de la qual es poden crear

noves instàncies d'un objecte amb els seus components i propietats.

Script: Document que conté instruccions, escrites en codi de programació.

Skybox: Conjunt de 6 parets que formen un cub, espai on es representa l'escena.

Software: Conjunt de programes i rutines que permeten a la computadora realitzar determinades tasques.

SDK: Kit de desenvolupament de software.

Tracking: Rastreig de la posició i la ubicació d'un objecte.

UI: Interfície d'usuari.

Unity: Motor de videojocs multi-plataforma creat per Unity Technologies.

VR: Realitat Virtual.

WebGL: És una especificació estàndard que està sent desenvolupada actualment per mostrar gràfics 3D en els navegadors web.

WebSocket: És una tecnologia que proporciona un canal de comunicació bidireccional i full-duplex sobre un únic socket TCP. Està dissenyada per ser implementada en navegadors i servidors web, però pot utilitzar-se per qualsevol aplicació client/servidor.

Wi-Fi: Wireless Fidelity.

Índex de figures

Fig. 3.1: Dispositiu Microsoft Kinect..... 19

Fig. 3.2: Consola Nintendo Wii i control Wiimote..... 19

Fig. 3.3: Font: ITAINNOVA, Institut Tecnològic d'Aragó. 21

Fig. 3.4: Dispositiu Leap Motion 24

Fig. 3.5: Font: http://www.dacast.com/blog/choose-best-video-streaming-api/..... 25

Fig. 3.6: La primera reacció que es té en una primera experiència de VR 26

Fig. 3.7: Les coordenades Leap es transformen en un espai Unity basat en la posició del cap en el món virtual..... 26

Fig. 4.1: Com instal·lar Leap Motion – primers passos..... 29

Fig. 4.2: Com instal·lar Leap Motion - Descàrrega del controlador 30

Fig. 4.3: Com instal·lar Leap Motion – Assistent d'instal·lació 31

Fig. 4.4: Com instal·lar Leap Motion – Icona de Leap Motion 31

Fig. 4.5: Com instal·lar Leap Motion – Tenda d'aplicacions de Leap Motion 32

Fig. 4.6: Leap Motion App Home "Airspace" – Demos 32

Fig. 4.7: Hardware del dispositiu desplegat 33

Fig. 4.8: Vista de planta del dispositiu Leap Motion 33

Fig. 4.9: Parts del dispositiu Leap Motion 33

Fig. 4.10: LEDs incorporats al dispositiu Leap Motion 35

Fig. 4.11: Microcontrolador del Leap Motion 35

Fig. 4.12: Descripció del microcontrolador (pin) 35

Fig. 4.13: Controlador USB de reconeixement del dispositiu 36

Fig. 4.14: Ports de sèrie del controlador 36

Fig. 4.15: Vista de les mans sobre el controlador Leap Motion 37

Fig. 4.16: Zona de cobertura del dispositiu Leap Motion 37

Fig. 4.17: Zona d'interacció de Leap Motion 38

Fig. 4.18: Les dos imatges que arriben al driver del Leap Motion 39

Fig. 4.19: Tipus de distorsió de les lents 40

Fig. 4.20:Tipus de distorsió a les lents del Leap Motion 40

Fig. 4.21: Calibratge de la distorsió mitjançant un mapa de mallat de punts 40

Fig. 4.22: Imatge amb la distorsió corregida 41

Fig. 4.23: Sistema de visió estereoscòpica de les dos càmeres del Leap Motion 42

Fig. 4.24: Determinació de la posició de les imatges al sistema de coordenades cartesianes de Leap Motion a través de tècniques de visió estereoscòpica 43

Fig. 4.25: Llenguatges de programació i plataformes de desenvolupament de Leap Motion 44

Fig. 4.26: Sistema de coordenades de Leap Motion 44

Fig. 4.27:Mètodes que conté l'objecte Controller() 45

Fig. 4.28: Diagrama de relació entre l'objecte Frame i els altres objectes 46

Fig. 4.29: Els vectors Hand PalmNormal i Direction defineixen la orientació de la mà. 47

Fig. 4.30: Esquema de les parts que componen l'objecte Hand 48

Fig. 4.31: Ossos dels dits de la mà 49

Fig. 4.32: Nou model de mà millorat respecte la versió 2.0 50

Fig. 5.1: Interfície d'usuari de l'editor de Unity 3D 55

Fig. 5.2: HandControllerSandBox del sensor Leap Motion 57

Fig. 5.3: Eines bàsiques de manipulació de la scene view i els objectes 58

Fig. 5.4: Panell Inspector amb els components d'un GameObject 59

Fig. 5.5:BoxCollider, SphereCollider i CapsuleCollider. 60

Fig. 6.1: Terrain Settings a l'Inspector de Unity 64

Fig. 6.2: Directori de treball de l'aplicació 65

Fig. 6.3: UI - Menú principal i menú d'escenes 65

Fig. 6.4: Visualització de l'escena 1 en mode Play 66

Fig. 6.5: Inspector del prefab de LM HandControllerSandBox 67

Fig. 6.6: Model de mans de l'escena 1 68

Fig. 6.7: Imatge de dia de l'escena 2 71

Fig. 6.8: Imatge de nit de l'escena 2 71

Fig. 6.9: Representació del gest KeyTap [8] 72

Fig. 6.10: Visualització de l'escena 3 en mode play 75

Fig. 6.11: Representació del gest Circle [8] 75

Fig. 6.12: Visualització de l'escena 4 en mode play 77

Fig. 6.13: Representació del gest ScreenTap 78

Fig. 7.1: Fases de desenvolupament del projecte 81

Índex de codis

Codi 1: Accés a la API del Leap Motion 53

Codi 2:Global Scriot que controla l'escena 1 69

Codi 3:TriggerControllerBlau, detecció dels cubs blaus al seu panell 70

Codi 4: GestureRecognition_TypeKeyTap, reconèixer el gest KeyTap 73

Codi 5: GestureRecognition_Circle, reconeix el gest Circle 77

Codi 6: GestureRecognition_TypeScreenTap, reconeix el gest ScreenTap 79

Codi 7: Done_Mover, script que serveix per disparar les bales 80

2. Introducció

2.1. Objectius del projecte

L'objectiu principal d'aquest treball ha estat la creació d'una interfície senzilla mitjançant l'ús d'un motor de videojocs, en aquest cas Unity3D, on l'usuari pugui interactuar amb l'escena utilitzant les seves mans a través del dispositiu Leap Motion. Per això, s'han desenvolupat els següents objectius:

1. Avaluar les capacitats d'aquest dispositiu i crear una aplicació que sigui capaç de reconèixer gests i moviments amb les mans.
2. Desenvolupar una sèrie d'escenes amb el programa Unity3D per tal de poder introduir les facilitats que ofereix el Leap Motion.
3. Descriure i documentar bé totes les fases i el procés realitzat per la creació de les diferents escenes i la incorporació de les mans a elles.

Una de les finalitats de la realització d'aquest projecte és la seva futura integració amb altres dispositius com són la Kinect o l'Hàptic, per tal de poder donar una major sensació d'immersió en l'aplicació o en el videojoc creat. Aquest projecte servirà com a base i directriu per futurs projectes també relacionats amb el desenvolupament d'aplicacions de realitat virtual, i s'utilitzarà per poder crear aplicacions encara més complexes i desenvolupades.

2.2. Abast del projecte

Per assolir els objectius esmentats a l'apartat anterior, es proposa la creació d'una interfície senzilla on siguem capaços de controlar l'escena amb les nostres mans. No es tracta ni de crear un videojoc complet ni de desenvolupar una aplicació molt complicada, ja que no es disposa del temps necessari per profunditzar tant amb l'aprenentatge.

Donar solucions en quant a gràfics, usabilitat i programació (entre d'altres), posar-ho tot en conjunt i que funcionin a la vegada no és una tasca fàcil. Per tant, amb un període aproximat d'entre 3-4 mesos, s'aprendrà de quina manera opera Unity3D com a programa de disseny i després enfocant-nos en el terreny de la programació, en aquest cas amb scripts en C#. S'utilitzarà principalment la informació de la web oficial de Unity [1] començant amb la realització de tutorials i poc a poc creant les nostres pròpies escenes.

Un cop tinguem dominada la API de Unity3D, ens centrarem amb la incorporació del sensor Leap Motion al Unity, i també en l'aprenentatge de la seva API [2].

2.3. Motivació

La motivació principal d'aquest projecte va ser poder veure quines possibilitats ofereix el dispositiu Leap Motion per poder manipular entorns relacionats amb la Realitat Virtual. Durant el grau en Enginyeria en Tecnologies Industrials, no em tingut la possibilitat d'estudiar sobre que tracta el món de la Realitat Virtual, i és per això que em va motivar la elecció d'aquest treball.

També considero un repte haver d'enfrontar-se a un món al fet d'haver de programar amb un nou llenguatge, C#, el qual no hem estudiat mai a la carrera, i per tant, partim d'un coneixement pràcticament nul.

Per últim, cal afegir que gràcies a la versatilitat que té aquest dispositiu, són moltes les aplicacions en les quals es podria utilitzar aquest sensor. Per això, en aquest treball l'objectiu és poder veure quines primeres aplicacions pot tenir, i de cara a un futur, poder-lo integrar amb altres dispositius com ara la Kinect o l'Hàptic, per poder crear un sentit d'immersió gairebé absolut en el món virtual.

2.4. Problemes trobats

Durant la realització del projecte, els problemes trobats varen ser majoritàriament en el moment de connectar el dispositiu Leap Motion amb l'ordenador. Es va instal·lar el software del Leap Motion per a la versió amb VR, i no funcionava amb l'ordinador propi, perquè requeria que la targeta gràfica de l'ordenador havia de ser Nvidia (d'una qualitat de gràfics superior). Es podia ometre aquest pas, i en fer-ho, no detectava el dispositiu Leap Motion al connectar-lo al port USB. Es va haver d'anar al CRV per demanar ajuda, i finalment, es va poder solucionar el problema que havia durat durant uns quants dies.

Una altra dificultat que es va trobar, però si més no un gran repte, va ser la programació dels scripts amb el llenguatge de programació C#. No s'havia programat mai amb *scripting*, i s'ha de dir que al principi va costar familiaritzar-se amb el codi, però poc a poc es van anar aprenent les eines més bàsiques per poder programar tant en la API de Unity com en la del Leap Motion.

2.5. Fases de desenvolupament

Bàsicament, les fases que s'han seguit per al desenvolupament d'aquest projecte les podem dividir en els punts següents:

- Aprenentatge i experimentació amb la API del software Unity3D.

- Creació de diverses escenes 3D amb el programa Unity3D on es puguin incorporar les mans per al control de l'escena.
- Aprenentatge i definició del llenguatge gestual per la interacció 3D mitjançant la API de Leap Motion.
- Implementació i prova de diverses interaccions 3D, com transformacions degudes als gests o transformacions provocades pel "pinch" (pessic) d'objectes.
- Disseny d'una aplicació amb la interacció del dispositiu.
- Implementació i prova de l'aplicació.

3. Estat de l'art

3.1. Motion Sensor

3.1.1. Taxonomia Motion Sensor

Les tecnologies relacionades als sensors de moviment cada vegada són més diverses. Aquestes han anat evolucionant amb els anys i cada cop es compten amb eines de sensors i moviments per poder interactuar amb l'ambient extern o amb el seu propi usuari a través de gests, moviments oculars o corporals. Així mateix, aquesta diversitat succeeix gràcies a que hi ha diversos dispositius que poden capturar els moviments de diferents parts del cos o objectes a través de diferents mitjans i mètodes. Entre les formes de captura més comuns hi ha les que funcionen a través de detecció d'imatges per mitjà de càmeres, per senyals de WiFi o per sensors.

- En primer lloc, les tecnologies de captura de moviment que funcionen a través de la visió de l'ordinador, en altres paraules, una càmera incorporada a l'ordinador que visualitza els gests que un realitza. Si bé aquest tipus de tecnologia, en els seus inicis, no contava amb gran precisió, en l'actualitat existeixen dispositius que sí que la tenen. A més, aquest tipus de tecnologia té dos branques importants: Gesture Recognition i Eye Tracking. El primer està relacionat als gests que l'usuari pot realitzar amb el cos i que té com a principals dispositius al mercat com ara Microsoft Kinect, Samsung Smart TV, Sony PlayStation Move, entre d'altres. El segon, està orientat al moviment dels ulls i la captura d'aquests i té com a eines que la utilitzen el Google Glass, Samsung Galaxy S4, etc.

- En segon lloc, es troben les tecnologies relacionades també a la captura de moviment però funcionen a través de senyals WiFi. Aquest tipus de dispositius emeten aquestes senyals per l'ambient on es trobin, i reboten al col·lisionar amb el cos de l'usuari, i d'aquesta manera aconsegueixen identificar alguns gests i moviments que realitzi la persona. Així mateix, entre la varietat de categories que posseeixen aquest tipus d'eines amb sensors de moviment, es ressalten 3: Gesture Recognition, Seguretat i Domòtica.

  La primera està relacionada, com en el cas anterior, al reconeixement de gests que realitza una persona i entre el dispositiu més ressaltant es troba el WiSee. La segona està enfocada a la seguretat, majoritàriament, de la llar i té a Karuoshi com un dels seus artefactes que consisteix en un detector de moviments mitjançant senyals WiFi. Finalment, els dispositius orientats a la domòtica cada vegada són més diversos i un

d'ells és el WeMO, que és un conjunt d'eines que ajuden a automatitzar els sistemes d'un habitatge.

- En tercer lloc, es tenen totes aquelles eines que funcionen a través de sensors i que compten amb la captura i identificació de gestos corporals de les persones, seguiments dels moviments dels dits, però a més poden detectar el moviment de músculs, variacions de camps magnètics, sons ultrasònics i la vibració d'objectes. Entre els aparells més ressaltants que utilitzen aquest tipus de tecnologies hi ha el control de videojocs Wii Motion de Nintendo, el MYO de Thalmic Labs, el qual és un braçalet per controlar dispositius amb gests; i Niessen, el qual és un dispositiu que serveix per detectar els moviments que es realitzen en un espai físic i que funciona a través de sensors infrarojos.

3.1.2. Productes Motion Sensor

3.1.2.1. Kinect

És un dispositiu de detecció de moviment desenvolupat per Microsoft per al seu ús amb ordinadors amb Windows i consoles de joc Xbox 360. Aquest dispositiu permet controlar i interactuar amb el computador o consola mitjançant moviments del cos humà (també anomenat *natural user interface*) i comandaments de veu. Es defineix el mecanisme de desenvolupament cognitiu com "qualsevol procés mental que millora la capacitat del nen per processar informació". D'acord a això, mitjançant els videojocs basats en la tecnologia Kinect els nens i estudiants poden practicar diferents ciències com les matemàtiques, la geografia, la història i la llengua, entre d'altres.[1]

Així mateix, els estudiants poden construir comunitats d'aprenentatge a través del xat en línia, jocs i funcions de vídeo. Kinect porta l'aprenentatge actiu com una eina completament nova, permetent als alumnes utilitzar-lo sense la necessitat de manipular un controlador.[2]

A més del seu ús en l'educació i entreteniment, Kinect és de gran ajuda per a qualsevol classe d'activitat en la qual es necessiti un major dinamisme, ja sigui una conferència en temps real, o una classe. D'aquesta manera, es pot obtenir un major grau d'interacció i comunicació entre els usuaris.[3]

[1] 6th European Conference on Games Based Learning 2012:224

[2] DePriest y Barilovits 2011:52

[3] DePriest y Barilovits 2011:52

Fig. 3.1: Dispositiu Microsoft Kinect

3.1.2.2. Wii

La Nintendo Wii és un aparell de joc que permet als usuaris jugar una varietat de jocs i algunes altres aplicacions. Wii compta amb un comandament a distància de detecció de moviment que permet als usuaris interactuar sense necessitat de tocar alguna pantalla. La forma en la qual el sistema de Wii és únic, és el seu ús de la detecció de moviment. El moviment és important en diverses maneres. Sembla que hi ha dues àrees principals d'aplicació de la Wii a les escoles: les classes d'educació física i els talls de tecnologia. Tenint en compte de baix cost i la facilitat d'ús de la Wii, així com la "naturalitat" del moviment físic, la Wii podria tenir un major ús en els contextos educatius que el que té ara.

Així i tot, en el camp de la investigació, s'han realitzat diversos estudis explorant les capacitats del Wii més enllà de l'educació i l'entreteniment, tenint acceptació en camps relacionats a la interacció social i investigació sobre noves IHC

Fig. 3.2: Consola Nintendo Wii i control Wiimote

3.1.2.3. Leap Motion

Dispositius tàctils com l'iPhone i les tablets en general, han passat a ser una de les plataformes escollides per desenvolupadors fent ús de gests tàctils. Però Microsoft va llençar al mercat un dispositiu per interaccionar amb la seva consola mitjançant el moviment. Ja no es tractava de tocar un dispositiu sinó d'un que capturés la posició corporal i el moviment de la persona. Aquest dispositiu s'anomena Kinect. Aquesta tecnologia no restringeix a l'ús domèstic de la Xbox 360, sinó que és una eina d'investigació, especialment en el camp de visió per computador.

I aquesta evolució ens porta al que es coneix com Leap Motion [1]. Es tracta d'un dispositiu de només 90€ que es connecta mitjançant USB a un ordinador Mac o Windows. Aquest dispositiu es diferencia de la Kinect no només per la seva gran precisió (0.01 mm) sinó en que Leap Motion està dissenyat per al reconeixement gestual de les nostres mans [4]. Per tant, la versatilitat que ofereix aquesta tecnologia és molt gran, ja que hi ha moltes aplicacions de camps diferents en les quals podria facilitar molt la interacció entre l'home i màquina. Per exemple, en el camp de la medicina, on evitant el contacte amb el dispositiu es podria guanyar molt més control sobre la higiene. Un altre exemple podria ser la arquitectura, on l'ús de la gestualitat de les mans facilitaria la manipulació d'objectes 3D.

Leap Motion té una àmplia comunitat de desenvolupadors, sent aquesta molt activa. Es celebren molts esdeveniments i hackathons com les 3D Jams i els VR Challenges. Els llenguatges més populars en els quals es programen les aplicacions per a Leap Motion són JavaScript i Unity3D.

Durant el treball, s'explica amb més detall les característiques i funcionalitats d'aquest dispositiu.

3.1.3. Interfície Home-Màquina (HMI)

- **Definició**

Una interfície d'usuari assistida per ordinador, també coneguda com **interfície home-màquina (HMI),** forma part del programa informàtic que es comunica amb l'usuari. En ISO 9241-110, el terme interfície d'usuari es defineix com "totes les parts d'un sistema interactiu (software o hardware) que proporcionen la informació i el control necessari per a que l'usuari dugui a terme una tasca amb el sistema interactiu".

Una interfície d'usuari / interfície home-màquina (HMI) és el punt d'acció en el que un home entra en contacte amb una màquina. El cas més simple que es coneix és el d'un interruptor. No es tracta d'un humà ni d'una "màquina" (la llum), sinó una interfície entre les dos. Per que una interfície home-màquina (HMI) sigui útil i significativa per a les persones, ha d'estar

adaptada als seus requisits i capacitats. Per exemple, programar un robot per que encengui el llum seria massa complicat i un interruptor al sostre no seria pràctic per a un llum en un soterrani.

- **Evolució de les interfícies home-màquina**

La manera en què ens comuniquem amb els dispositius està evolucionant molt en els últims anys. S'està treballant per millorar tant el disseny com les aplicacions HMI per permetre que la interacció de les persones amb els sistemes sigui simple i intuïtiva, o que els sistemes siguin accessibles, usables i adaptables. Aquesta interacció home-màquina ha de poder realitzar-se a través d'interfícies multimodals per poder comprendre de la manera més completa possible les accions que l'usuari vol realitzar, així com el seu context.

Fig. 3.3: Font: ITAINNOVA, Institut Tecnològic d'Aragó[4].

L'automatització de màquines i processos consta de diferents parts independents però àmpliament relacionades entre si. Una d'elles indubtablement serà la secció de control, on es defineixen les pautes de comportament del sistema, és a dir, què ha de fer i com ha d'actuar tot un conjunt de dispositius per dur a terme una tasca més o menys complexa, i amb un major o menor grau d'autonomia. Però hi ha altres elements tan importants com l'anterior, que són els que conformen la interfície que permetrà la interacció per part d'un operari o assistent amb la prèviament comentada part de control. En l'automatització industrial aquests components comprenen el que d'una forma genèrica es denomina HMI.

S'ha passat de dependre d'un par de botons d'un ratolí o un conjunt de tecles d'un *gamepad*, a interaccionar a través de **gests**. Aquests canvis s'han introduït en les aplicacions d'escriptori, però on realment s'ha fet un canvi de paradigma és en la industria del videojoc.

[4] http://www.itainnova.es/competencias/multimedia/linea-de-investigacion-tecnologias-multimedia-hombre-maquina

3.2. Eines de desenvolupament

3.2.1. Unity 3D

Unity [1] [5] és un motor gràfic per a videojocs, en 2 i 3 dimensions, creat per Unity Technologies. És una aplicació multi-plataforma que pot executar-se en Apple OS X, Linux i Microsoft Windows i permet desenvolupar jocs, aplicacions interactives, visualitzacions i animacions en temps real[5] per als sistemes operatius Windows, Mac, Linux, Android, BlackBerry, per a les videoconsoles Xbox 360, Xbox One, PlayStation 3, PlayStation 4, PlayStation Vita, i per WebGL[6] i Samsung Tv actualment.

El IDE[7] de Unity és el centre de la línia de desenvolupament que consisteix en un editor visual amb una sèrie d'eines potents per a la creació d'aplicacions amb entorn gràfic. Aquest contingut gràfic pot ser realitzat des del propi editor o ampliat gràcies a la compatibilitat amb aplicacions de creació de gràfics i animació 3D com són 3D Studio Max i Maya entre d'altres. A més, compta amb la integració amb Visual Studio 2015 mitjançant una extensió.

La jugabilitat o *gameplay* es programa mitjançant llenguatges de scripts C#[8], Javascript i Boo[9]. El nucli del motor proveeix una sèrie d'esdeveniments, que són fàcilment programables per cada script, associat als objectes o elements, i d'aquesta manera defineix els comportaments d'aquests. Per a aquest projecte s'ha utilitzat C#.

La estructura de les aplicacions a Unity són separades per escenes que consten d'un conjunt d'objectes gràfics, disparadors i scripts. Aquests objectes poden ser jerarquitzats de forma visual i còmoda que facilita la edició i relació entre objectes i elements de l'aplicació.

Unity té un gran recolzament de la comunitat de desenvolupadors professionals i novells. En la seva tenda (Asset Store) que ve integrada al propi IDE, es pot trobar gran quantitat d'elements així com alguns projectes d'exemple complets. Com a mostra d'aquesta activitat de la comunitat, els fòrums i canals de *youtube* són molt actius.

---

[5] Un sistema en **temps real** (STR) és aquell sistema digital que interactua activament amb un entorn de forma dinàmica i els seus processos d'execució es realitzen en el moment en que es rep un paràmetre d'entrada o una interacció amb l'usuari.

[6] WebGL és una especificació estàndard que està sent desenvolupada actualment per mostrar gràfics en 3D en navegadors web. El WebGL permet mostrar gràfics en 3D accelerats per maquinari (GPU) en pàgines web, sense la necessitat de plug-ins en qualsevol plataforma que suporti OpenGL 2.0 o OpenGL ÉS 2.0. Tècnicament és un API per javascript que permet usar la implementació nativa de OpenGL ÉS 2.0 que serà incorporada en els navegadors.

[7] **Integrated Development Environment**: entorn de desenvolupament integrat.

[8] **C#** o **CSharp** és un llenguatge de programació orientat a objectes desenvolupat per Microsoft i basat en C/C++ per la seva plataforma .NET.

[9] Boo és un llenguatge de programació orientat a objectes, de tipus estàtics amb una sintaxi inspirada en Python.

Aquest motor gràfic està en continu desenvolupament, i prova d'això és que cada any preparen una nova versió amb multitud de millores.

La llicència de Unity és gratuïta, però compta amb una versió *Pro* que inclou diverses característiques i eines que no es troben a la versió gratuïta. Amb la versió *Pro* el desenvolupador pot comercialitzar el seu producte.

Com alternatives a Unity més conegudes i potents, s'ha de destacar a Unreal Engine, desenvolupat en la seva primera versió per al joc multi-jugador Unreal Tournament i que actualment es un referent en quant a il·luminació realista en temps real, i CryEngine desenvolupat per l'empresa alemanya Crytek i que ha donat títols de gran qualitat gràfica realitzats per la pròpia companyia i tercers.

La elecció de Unity com a motor gràfic per la creació de l'aplicació rau en lo següent:

- Unity és dels únics motors que proporciona eines completes per al desenvolupament d'aplicacions amb els dispositius Leap Motion, Kinect, hàptic i Oculus Rift, els quals formen part dels altres tres projectes que s'estan realitzant paral·lels a aquest.
- La corba d'aprenentatge del propi Unity és més accessible que la de la seva competència.
- Unity inclou un entorn de desenvolupament integrat (IDE) anomenat **Microsoft Visual Studio**, Visual Studio, per a sistemes operatius Windows. Suporta múltiples llenguatges de programació, tals com C++, C#, , F#, Python , Ruby, PHP, Java, Visual Basic .NET.
- Compta amb una gran quantitat de contingut ja creat per la pròpia Unity Technologies i la comunitat que li dóna suport.

3.2.2. Leap Motion

El Leap Motion Controller és un sensor de moviment que ens permet controlar l'ordinador mitjançant el moviment de les mans i els dits amb alta precisió, i en genera un informe sobre la seva posició, velocitat i orientació amb baixa latència i bona precisió. El controlador pot ser muntat en un dispositiu de realitat virtual (RV) o ser utilitzat sobre una taula. El sistema de control Leap Motion consta d'un dispositiu de maquinari (hardware) i d'un component de programari (software) que s'executa com un servei a l'ordinador amfitrió.

Aquest aparell de interacció compta amb dos càmeres monocromàtiques i tres Leds infrarojos, permetent-nos interactuar amb l'ordinador de forma intuïtiva i simple mitjançant l'ús de les nostres mans.

Fig. 3.4: Dispositiu Leap Motion

Com s'ha dit, Leap Motion és una interfície de control de moviment dissenyada per al seu ús amb ordinadors existents en l'actualitat. David Holz, un dels seus creadors, va parlar sobre el començament de la idea, la qual va ser crear una millor manera per que els usuaris interactuïn amb els seus dispositius computacionals. La idea de fer-ho va ser durant uns dies en els que va adquirir un doctorat en matemàtiques de la UNC, i mentre treballava en mecànica de fluids. Al seu treball, va tenir problemes en la creació i la manipulació de models 3D utilitzant un ratolí i un teclat – una operació innecessàriament complicada que implica clics i menús desplegables. Holz volia una manera de "modelar argila virtualment, tan fàcil com modelar argila en el món real". Després de quatre anys d'investigació i moltes iteracions de hardware, el Leap Motion va ser acabat[10].

A diferència d'altres opcions de control de moviment com el Kinect de Microsoft, l'enfocament de Leap Motion és la interactivitat personal i delineament subtil de moviments específics. Això vol dir que les persones poden interactuar amb el seu ordinador d'una manera molt més natural, en lloc d'haver d'estar lluny i ser excessivament gestual perquè els sensors de seguiment captin els seus moviments. En termes de ser una eina pràctica la gent pot utilitzar el Leap Motion per realitzar les seves tasques diàries d'una forma més productiva, per exemple, emulant saltar amunt i avall en una bassa de riu, aquest proporciona un avantatge molt marcada en contrast amb realitzar aquest moviment mitjançant controls convencionals.[11]

En el punt 5 s'explica més detalladament totes les característiques del Leap Motion i de la seva API, ja que en aquest punt es fa referència a l'estat de l'art d'aquestes eines.

3.3. Situació actual del desenvolupament d'aplicacions

3.3.1. Interfície de programació d'aplicacions (API)

Una interfície de programació d'aplicacions, abreviada com a API [6] de l'anglès "Application Programming Interface", és un conjunt de subrutines, regles (codi), funcions i procediments

[10] LEAP MOTION, Inc. 2013

[11] Seminar Reporton, Leap Motion 2012:4

(o mètodes, en la programació orientada a objectes) que ofereix una certa biblioteca per ser utilitzada per un altre software com una capa d'abstracció.

Les API poden servir per comunicar-se amb el sistema operatiu (WinAPI), amb bases de dades (DBMS) o amb protocols de comunicacions (Jabber/XMPP). En els últims anys, per suposat, s'han sumat múltiples xarxes socials (Twitter, Facebook, Youtube, Flickr, LinkedIn, etc) i altres plataformes online (Google Maps, WordPress…), lo que ha convertit el social media marketing és una mica més senzill, més rastrejable i, per tant, més rendible.

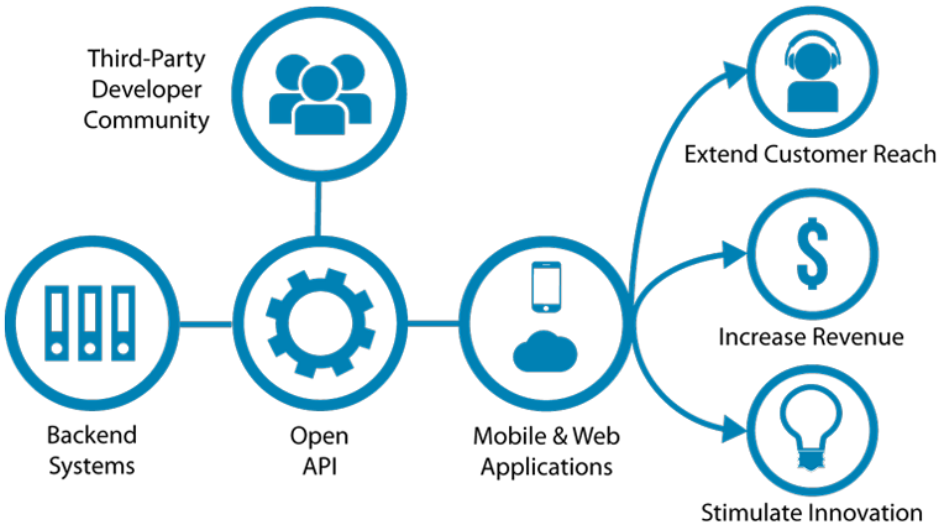


Fig. 3.5: Font: http://www.dacast.com/blog/choose-best-video-streaming-api/

Les API són valuoses perquè permeten fer ús de funcions ja existents en un altre programari (o de la infraestructura ja existent en altres plataformes) per no estar reinventant la roda constantment, reutilitzant així codi que es sap que ja està provat i que funciona correctament. En el cas d'eines propietàries (és a dir, que no siguin de codi obert), són una manera de fer saber als programadors d'altres aplicacions com incorporar una funcionalitat concreta sense per això haver de proporcionar informació sobre com es realitza internament el procés.

3.3.2. Seguiment de les mans en VR

La Realitat Virtual és un sistema d'entrada sensorial que transporta a l'usuari a un món virtual. La tecnologia actual utilitza pantalles muntades al cap com a mitjà principal d'entrada sensorial. Aquestes pantalles utilitzen la posició i orientació de seguiment de diversos nivells de sofisticació per submergir a l'usuari en un món virtual i proporcionar un sentit de presencia física. El seguiment de les mans pot amplificar, per tant, aquest sentit de presència.

Fig. 3.6: La primera reacció que es té en una primera experiència de VR

Degut a que cadascú té un sentit natural, físic on les seues pròpies mans estan a l'espai, és important alinear els mons físics i virtuals. Es pot aconseguir aquesta alineació col·locant el LeapHandController a la ubicació virtual corresponent a la seva ubicació en el món real. En VR, el marc de referència comú entre el món físic i el virtual és el HMD (Head-Mouted Display).

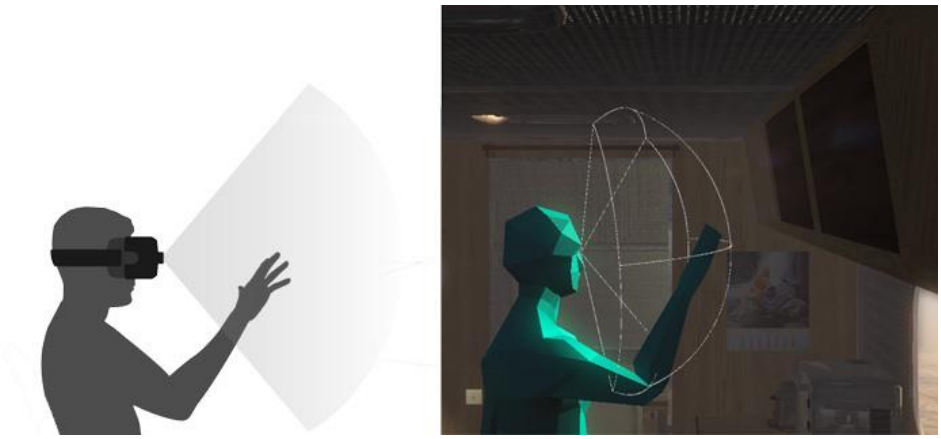


Fig. 3.7: Les coordenades Leap es transformen en un espai Unity basat en la posició del cap en el món virtual

Unity controla les càmeres de l'escena per que coincideixin amb el moviment del HMD dintre de la seva àrea de seguiment ("traking area"). El prefab LMHeadMountedRig utilitza la ubicació de la càmera proporcionada per Unity per col·locar el LeapHandController en la posició correcta en el món virtual. Les coordenades en les dades de seguiment es transformen des de l'espai de Leap a l'espai de Unity en relació amb la posició i la orientació del gameObject LeapHandController.

Si s'utilitza directament un SDK de HMD, en lloc del suport integrat de Unity VR, s'hauran d'afegir els components de Leap Motion a un equip de càmera ja existent.

3.3.2.1. Hand Tracking: Desktop vs. VR

Existeixen unes quantes diferències segons si utilitzem el Leap Motion com a versió d'escriptori (Desktop) o si l'utilitzem en versió de VR. El software Orion ha estat dissenyat per visualitzar les mans des de dalt per a l'ús en VR. En canvi, el la versió v2 Desktop ha estat dissenyada per col·locar les mans sobre el controlador.

Les diferencies de la versió Desktop respecte la VR són les següents:

- En VR, els usuaris tenen una millor idea d'on estan les seves mans en relació als objectes de la escena 3D. Això pot produir una major sensació de presència a l'usuari.
- La escala del que ens envolta, en RV és un tema important i encara més quan se li afegeixen les mans de l'usuari. El sentit de l'escala es veu afectat per l'escala relativa entre objectes familiars, com les mans, així com per la separació de les càmeres estèreo.
- Les mans s'auto-oclouen de manera diferent ja que el dispositiu Leap Motion està mirant la mà des d'un punt de vista diferent. Algunes interaccions, com l'apuntat horitzontal, que funciona bé en un context d'escriptori podria no funcionar tan bé en un context de VR. Per contra, altres interaccions, com el polze cap amunt, funcionen molt millor en VR que en un context d'escriptori.
- S'ha de dir al software de servei Leap Motion que reconeixi les mans des de la perspectiva HMD. Això s'aconsegueix establint l'opció *Is HMD Mounted* al LeapHandController.

S'ha de dir que és molt millor la versió per a VR, però tal i com s'explica al capítol 4 (al punt 4.1.1), aquest treball ha estat desenvolupat amb una versió desktop, ja que per la VR es necessita el Oculus Rift i uns PC amb unes targetes gràfiques específiques amb molt bona qualitat d'imatge.

Com que aquest treball s'ha desenvolupat aquesta versió, seria interessant que en futurs es treballés amb la versió de VR per poder crear una aplicació encara més semblant al món real.

4. Anàlisi del sistema

En aquest apartat, s'explica quins passos s'han de seguir des de zero per utilitzar el dispositiu Leap Motion, des de la seva instal·lació al PC, els packs i divers necessaris, quin és el seu principi de funcionament i com treballar amb la seva API per tal de poder programar el control de l'escena amb les nostres mans.

4.1. Leap Motion

4.1.1. Instal·lació del Leap Motion

1. En primer lloc, s'ha d'entrar a la pàgina web de Leap Motion a l'apartat de configuració[12]. Apareixerà una pantalla com la següent:

Fig. 4.1: Com instal·lar Leap Motion – primers passos.

Just a sota d'aquests passos, es veuen un botó per la descàrrega del controlador (Windows, Mac o Línux):

[12] https://www.leapmotion.com/setup

Fig. 4.2: Com instal·lar Leap Motion - Descàrrega del controlador

Existeixen dos versions del software; una d'elles és la versió d'escriptori (Desktop) i l'altra és la versió de Realitat Virtual (VR). Per tant, sigui quina sigui el tipus d'aplicació que volem crear, haurem de descarregar una versió o l'altra. S'ha de tenir en compte que per cadascuna de les versions existeixen diferents requeriments del sistema, és a dir, el tipus de sistema operatiu, la memòria RAM, un tipus específic de targeta gràfica de l'ordinador (GPU), etc. que ara es detalla a continuació:

- Versió Dekstop:
  - Windows® 7+ or Mac® OS X 10.7+
  - AMD Phenom™ II or Intel® Core™ i3/i5/i7 processador
  - 2 GB RAM
  - USB 2.0 port
  - Connexió a Internet
- Versió VR:
  - NVIDIA GTX 970 / AMD R9 290 equivalent o superior
  - Intel i5-4590 equivalent o superior
  - 8GB+ RAM
  - Compatible HDMI 1.3 video output
  - 2x USB 3.0 ports
  - Windows 7 SP1 o més nou

En el nostre cas, s'utilitzarà la versió d'escriptori, ja que el PC de treball no disposa de la GPU requerida per la versió de VR.

2. Quan s'hagi descarregat l'arxiu de la instal·lació, s'obrirà (doble click) i apareixerà un assistent que ens guiarà durant tot el procés.

Fig. 4.3: Com instal·lar Leap Motion – Assistent d'instal·lació

3. Quan acabi la instal·lació comprovarem que tot hagi sortit bé. Per fer això, hem de dirigir-nos a la part inferior dreta del nostre escriptori, on tindrem que localitzar una icona com el de la següent imatge:

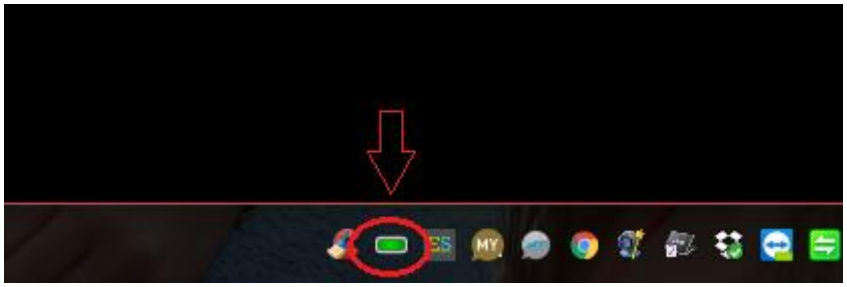


Fig. 4.4: Com instal·lar Leap Motion – Icona de Leap Motion

Important: si la icona no es troba en verd significa que el sensor no està connectat a l'ordinador.

Personalment, vaig tenir el problema que el dispositiu sí que estava connectat a l'ordinador però en canvi la icona estava de color vermell, és a dir, com si es trobés apagat. En aquest cas vaig haver de "forçar" l'engegada del dispositiu a través de la pantalla negra de Windows. Els passos que vaig seguir varen ser:

- Al buscador de Windows escriure: cmd (Símbol del sistema) i executar-ho com administrador amb el botó dret.
- A la pantalla negra escriure: *net start LeapService* + Intro
- Apareixerà un missatge dient que el dispositiu s'ha activat correctament.

Un cop realitzats aquests passos, si cliquem el botó dret sobre aquesta icona, ja de color verd, ens apareixeran diverses opcions. Una d'elles és el visualitzador. Si l'obrim, veurem tot el que està capturant el sensor del nostre Leap Motion.

Al instal·lar el controlador, es crea un accés directe al nostre escriptori: Leap Motion App

Store, amb el que podrem trobar aplicacions per al dispositiu.

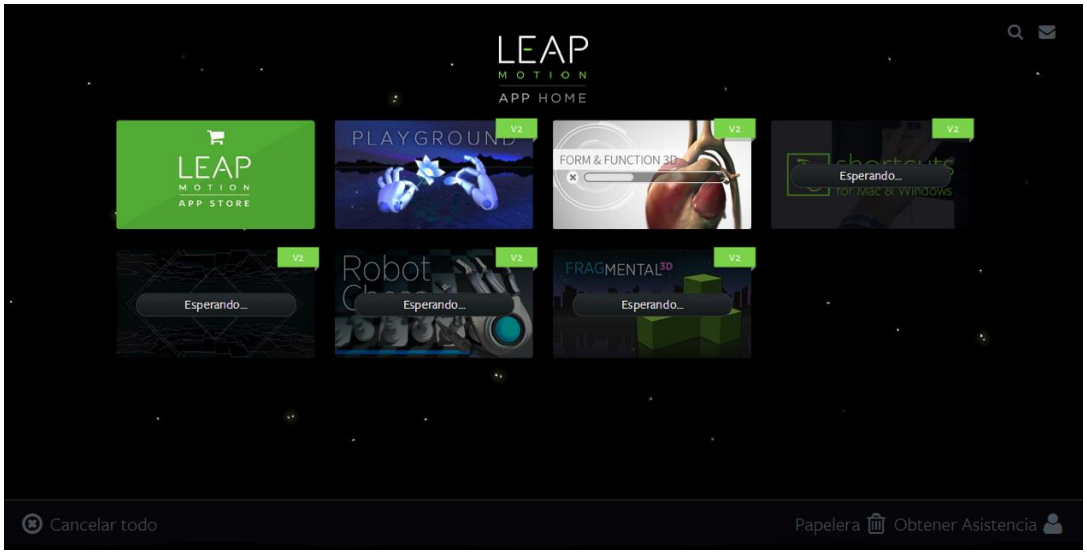


Fig. 4.5: Com instal·lar Leap Motion – Tenda d'aplicacions de Leap Motion

Aquestes són una sèrie d'aplicacions prèviament instal·lades amb les quals es pot comensar a fer proves amb el Leap Motion.

Fig. 4.6: Leap Motion App Home "Airspace" – Demos

4.1.2. Característiques tècniques

En aquest apartat, es descriuen les característiques tècniques del dispositiu Leap Motion s'analitzaran els components que formen el hardware.

Fig. 4.7: Hardware del dispositiu desplegat[13]

1. Dimensions

   Aquest dispositiu té unes dimensions molt reduïdes en comparació amb altres interfícies gestuals que hi ha actualment en el mercat: tan sols mesura 75mm de llarg, 25mm d'ample i 11mm d'alt.

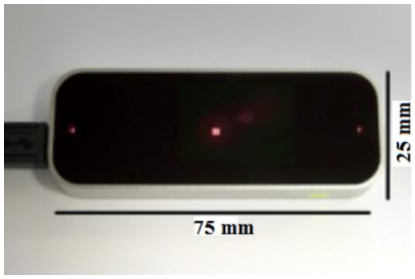


Fig. 4.8: Vista de planta del dispositiu Leap Motion

2. Parts del dispositiu

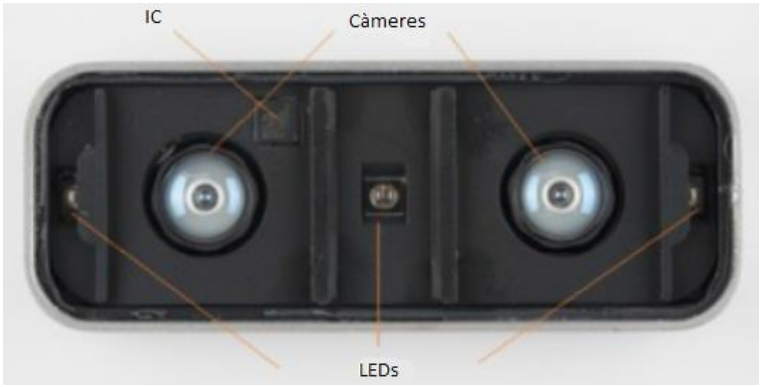


Fig. 4.9: Parts del dispositiu Leap Motion

[13] http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

Com es pot observar en la imatge anterior, Leap Motion disposa de dos càmeres, tres LEDs i un microcontrolador. A continuació anem a veure cada part amb una mica més de detall.

- Càmeres

  Les càmeres són una de les parts més importants del dispositiu, atès que són les encarregades de capturar les imatges i el seu bon funcionament condicionarà el correcte funcionament de la resta de sistema.

  Cadascuna d'aquestes càmeres compta amb un sensor monocromàtic, sensible a la llum infraroja, amb una longitud d'ona de 850 nm. Aquests sensors poden treballar a una velocitat de fins a 200 fps, depenent del rendiment de l'ordinador / tablet al qual connectem el dispositiu. A més, cada sensor és de tipus CMOS. Per què aquest tipus de sensor?

  - ✓ La digitalització dels píxels en un sensor CMOS es produeix dins de cada cel·la, pel que no és necessari un xip extern com passaria en el cas d'utilitzar sensors CCD. Això es tradueix en major velocitat per capturar imatges i en menor espai per albergar els sensors.
  - ✓ Aquests sensors són més econòmics que els sensors CCD.
  - ✓ En aquest tipus de sensor no es produeix el fenomen blooming, al contrari que en els sensors CCD. Aquest fenomen es produeix quan una cel·la se satura de llum i fa que les cel·les de l'entorn també es saturin.
  - ✓ La lectura simultània de cel·les en els CMOS és més gran que en els CCD.
  - ✓ El consum elèctric dels CMOS és menor que el dels CCD.

- Il·luminació Infraroja

  Els LEDs s'encarreguen d'il·luminar la zona de cobertura per inundació. Treballen en l'espectre de llum infraroja a una longitud d'ona de 850nm que, com és lògic, és la mateixa a la qual són sensibles els sensors òptics. Varien el seu consum elèctric -i per tant la il·luminació- depenent de la llum que hi hagi en la zona de cobertura per a assegurar una mateixa resolució d'imatge.

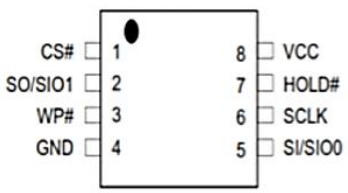
Fig. 4.10: LEDs incorporats al dispositiu Leap Motion

Com es pot observar en la imatge anterior, els LEDs estan separats per petites barreres de plàstic. D'aquesta manera s'assegura que la il·luminació sigui uniforme en tota la zona de cobertura. A més, es protegeix als sensors òptics d'una possible saturació de llum, atès que d'aquesta manera la llum infraroja no els il·lumina directament.

- El microcontrolador

Fig. 4.11: Microcontrolador del Leap Motion

Es tracta d'un circuit integrat que se sol utilitzar per fer la funció de BIOS (MXIC MX25L3206E-32M bits CMOS SERIAL FLASH). En aquest cas conté el programa que controla tot el dispositiu -per, entre altres coses, regular la il·luminació- i s'encarrega de recollir la informació dels sensors per després enviar-la al driver o controlador instal·lat a l'ordinador / tablet.



PIN DESCRIPTION

SYMBOL	DESCRIPTION
CS#	Chip Select
SI/SIO0	Serial Data Input (for 1 x I/O)/ Serial Data Input & Output (for Dual Output mode)
SO/SIO1	Serial Data Output (for 1 x I/O)/ Serial Data Output (for Dual Output mode)
SCLK	Clock Input
WP#	Write protection
HOLD#	Hold, to pause the device without deselecting the device
VCC	+ 3.3V Power Supply
GND	Ground

Fig. 4.12: Descripció del microcontrolador (pin)

- Controlador USB

Fig. 4.13: Controlador USB de reconeixement del dispositiu

Leap Motion compta amb un controlador USB perquè l'ordinador pugui reconèixer el dispositiu. Aquest controlador és d'alta velocitat i pot suportar USB 3.0.

3. Enviament i recepció de dades

Les dades s'envien i es reben al controlador de l'ordinador a través de dos ports sèrie: UART_RX i UART_TX.

Fig. 4.14: Ports de sèrie del controlador

4. Zona de cobertura

Gràcies a les seves lents de gran abast angular, el dispositiu compta amb un gran espai d'interacció de vuit peus cúbics, que pren la forma d'una piràmide invertida - la intersecció dels camps de visió de la càmera binocular. En la versió d'escriptori, el rang de visió del controlador de Leap Motion es limita a aproximadament 2 peus (61

cm) per sobre del dispositiu.

Fig. 4.15: Vista de les mans sobre el controlador Leap Motion

Amb el programari Orion beta, això s'ha ampliat a 2,6 peus (80 cm). Aquesta gamma està limitada per la propagació de la llum LED a través de l'espai, ja que es torna molt més difícil de deduir la posició de la seva mà en 3D més enllà d'una certa distància. La intensitat de la llum LED està limitada en última instància per la corrent màxima que pot extreure a través de la connexió USB.

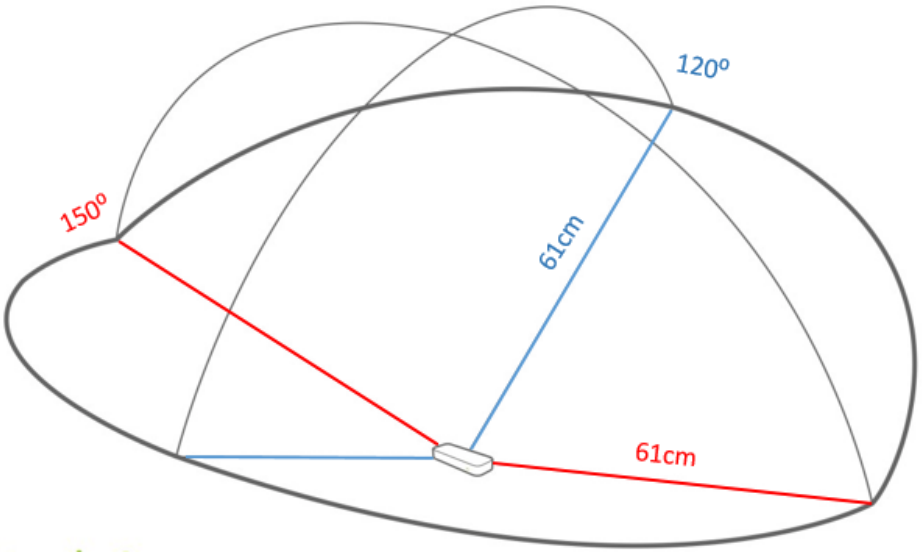


Fig. 4.16: Zona de cobertura del dispositiu Leap Motion

Aquesta zona depèn de l'angle de visió de les lents de les càmeres i de la intensitat màxima que pot lliurar la connexió USB als LEDs. Al seu torn, l'angle de visió (α) depèn de la distància focal i de la mida del sensor de la següent manera:

$$\alpha = 2 \cdot \arctan\left(\frac{d}{2 \cdot f}\right)$$

(on *d* és la diagonal del sensor i *f* la distància focal)

Tant l'angle de visió horitzontal de Leap Motion com el vertical, són de 150,92º. Aquests angles delimiten la zona d'interacció.

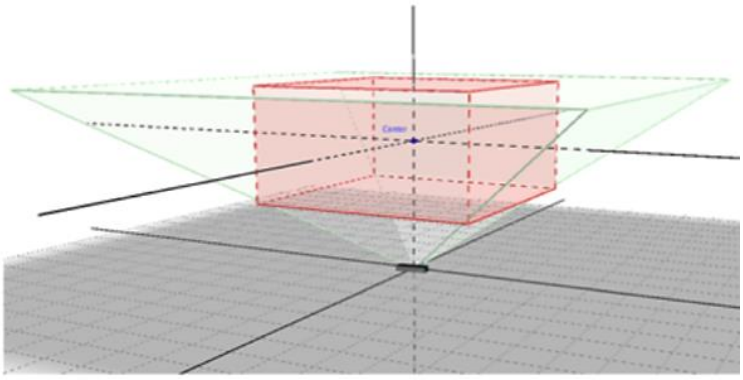


Fig. 4.17: Zona d'interacció de Leap Motion

En la API del dispositiu (que es veurà en l'apartat següent) es defineix una zona de treball anomenada "Interaction Box" per un volum de *110.55mm d'alçada* x *110.55 mm d'amplada* x *69.43 mm de profunditat*, que varia les seves dimensions depenent d'on es trobi l'objecte a rastrejar. Aquesta és la zona en la que es marca el centre del sistema de coordenades cartesià de Leap Motion. Des del *driver* del dispositiu es pot configurar l'altura a la qual es trobarà el centre d'aquesta zona d'interacció. Aquesta altura pot estar entre 7 i 25 cm des del dispositiu.

4.1.3. Principi de funcionament

Com ja s'ha comentat anteriorment, Leap Motion és un sensor capaç d'obtenir la màxima informació possible de la configuració i del moviment realitzat per les nostres mans.

Existeixen diferents tècniques per obtenir paràmetres de profunditat mitjançant un sistema de visió (per exemple, Kinect obté imatges a color i imatges de profunditat projectant un mallat de punts dels quals n'analitza la distància a partir de la distorsió del mallat). Amb aquesta informació, es poden obtenir variables suficients per poder determinar un gest realitzat.

4.1.3.1. Funcionament del controlador Leap Motion

El dispositiu il·lumina la zona de cobertura mitjançant la llum infraroja emesa a traves de les seves tres LEDs, amb una longitud d'ona de 850nm. Aquesta zona de cobertura està limitada per l'angle de visió dels sensors (s'ha explicat en l'apartat anterior) i per la corrent màxima que pot entregar la connexió USB.

- *Ajust de la resolució*

Quan un objecte – en el nostre cas, les mans – és il·luminat, es produeix una reflexió de llum que arriba al dispositiu i incideix sobre les lents de les dues càmeres. Aquestes lents, de tipus biconvexes, concentren els raigs en el sensor de cada cambra; i les dades recollides pels sensors s'emmagatzemen en una matriu (imatge digitalitzada) a la memòria del controlador USB, on es realitzen els ajustaments de resolució adequats mitjançant el microcontrolador del dispositiu.

- *Emmagatzematge de les dades en un buffer*

Un cop ajustada la resolució, les dades dels sensors s'envien directament al *driver* instal·lat a l'ordinador. Aquestes dades representen un valor d'intensitat lluminosa per cada píxel de la imatge capturada i es guarden en un *buffer*. El valor d'intensitat lluminosa es quantifica a 8 bits per generar una imatge RAW en escala de grisos -per tant, hi ha un total de 256 possibles valors de lluminositat-. I ens falta una altra dada: cada imatge té una mida de 640 x 120px, de manera que en total hi ha 76.800 píxels per imatge. I tot i així, el dispositiu és ràpid, perquè les imatges no són tractades en el propi dispositiu, ja que aquest només recull i envia dades.

- *Enviament de les imatges al driver*

Quan les imatges de les dos càmeres arriben al *driver*, són analitzades per identificar les mans i els dits a partir d'un model matemàtic de caracterització anatòmic. A més, s'obté la profunditat mitjançant un algoritme que s'explicarà a continuació.

Fig. 4.18: Les dos imatges que arriben al driver del Leap Motion

- *Tipus de distorsió de les lents i calibratge*

Abans d'explicar l'algoritme d'identificació i el de profunditat cal tenir en compte que les lents del dispositiu produeixen una distorsió en la imatge òptica, deformant l'objecte observat.

Fig. 4.19: Tipus de distorsió de les lents

Al Leap Motion es produeix el que es coneix com a **distorsió complexa**: una mescla entre la *distorsió de barril* i la *distorsió de coixí*.

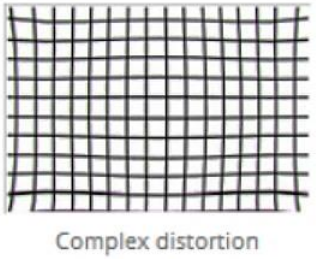


Fig. 4.20:Tipus de distorsió a les lents del Leap Motion

Per millorar aquesta distorsió, Leap Motion té una opció de calibratge mitjançant la qual s'obté un mapa de mallat de punts de calibratge que es superposa a la imatge captada per cada sensor.

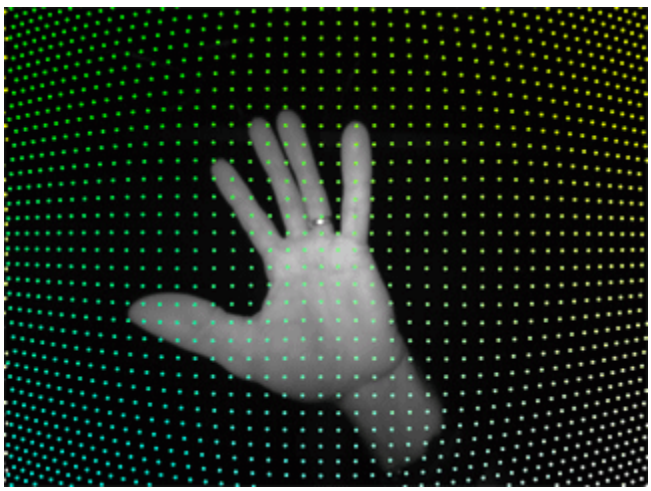


Fig. 4.21: Calibratge de la distorsió mitjançant un mapa de mallat de punts

Per tant, cada *buffer* de dades d'imatge que s'envia al *driver* va acompanyat d'un altre *buffer* que conté les dades de distorsió. Aquestes dades són una reixa de 64 x 64 punts amb dos valors de 32 bits cadascun. Cada un d'aquests punts representa un raig projectat a la càmera. El valor d'un punt del mallat defineix la lluminositat d'un píxel en la imatge i es poden obtenir les dades de lluminositat de tots els píxels mitjançant interpolació.

Així doncs, es pot obtenir el valor de brillantor per a qualsevol raig projectat. Els valors de la quadrícula que cauen fora del rang [0 ... 1] no corresponen a un valor de dades de la imatge i, per tant, hem d'ignorar aquests punts. A veure si s'entén millor amb un exemple:

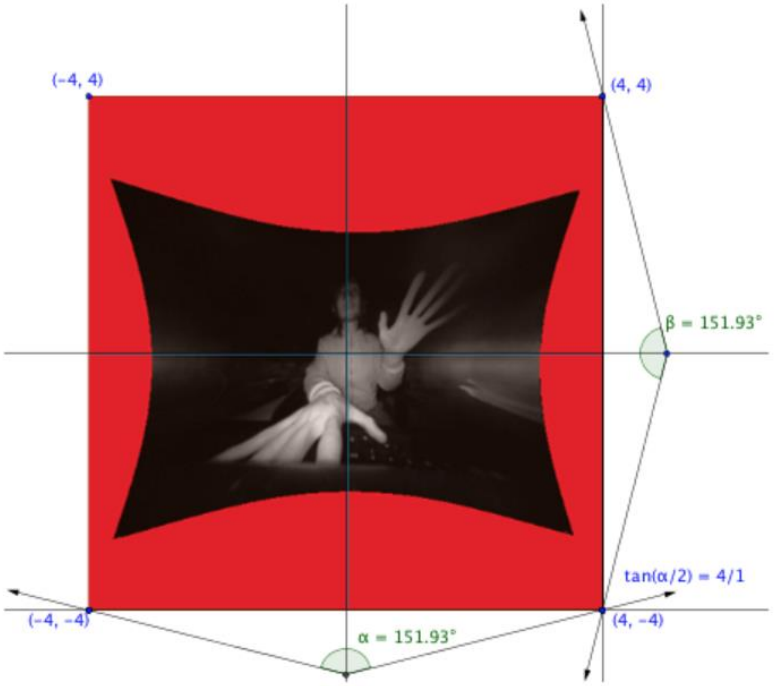


Fig. 4.22: Imatge amb la distorsió corregida

La imatge ens mostra una reconstrucció de les dades d'una imatge amb la distorsió corregida. El valor de brillantor de cada píxel de la imatge es va originar a partir d'un raig de llum que va entrar a la cambra des d'una direcció específica. La imatge es reconstrueix mitjançant el càlcul de les pistes horitzontals i verticals representats per cada píxel i es pot trobar el valor de brillantor veritable de les dades de la imatge utilitzant el mapa de calibratge. Les parts vermelles de la imatge representen les àrees dins de la prestació per a la qual cap valor de brillantor està disponible (el camp de visió real és de menys de 150 graus).

- *Identificació de les mans i els dits amb tècniques de visió estereoscòpica*

Un cop han arribat les imatges, s'han corregit degudament i el driver ha identificat les mans i els dits, es pot determinar la posició d'aquestes en el sistema de coordenades cartesianes

de Leap Motion a través de tècniques de visió estereoscòpica. Perquè, com ja hem vist, Leap Motion és un sistema de captació d'imatges basat en la visió binocular i per això es poden obtenir distàncies.

Bàsicament, un sistema estereoscòpic funciona de la següent manera: gràcies a la separació de les dues càmares en l'eix X s'obtenen dues imatges amb petites diferències (disparitat).

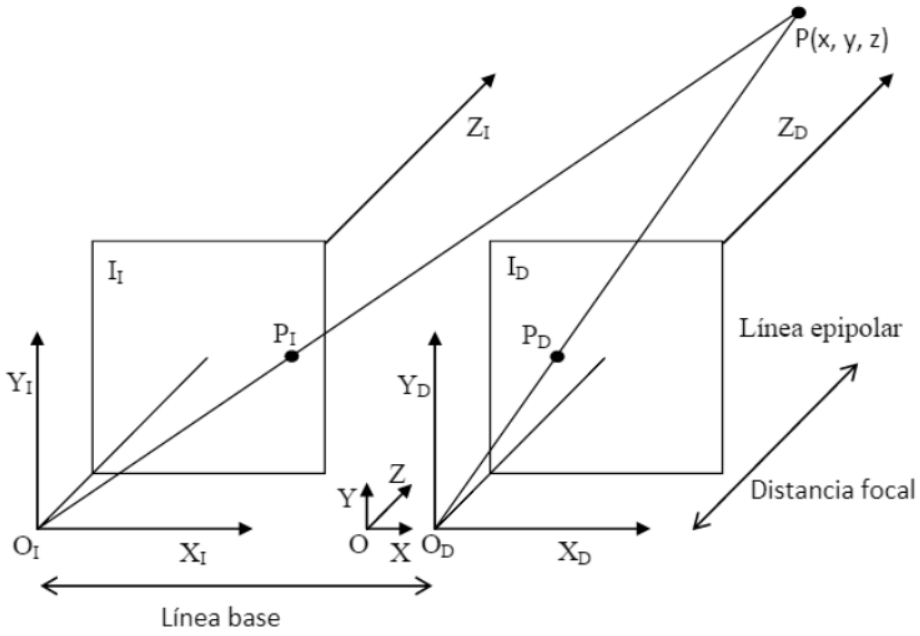


Fig. 4.23: Sistema de visió estereoscòpica de les dos càmeres del Leap Motion

Com es pot observar en la imatge anterior, les dues càmeres - representades per O_I i O_D - estan en el mateix pla Z, sobre la línia base. Si tracem una línia epipolar entre les dues imatges I_I i I_D, atès que O_I i O_D estan en el mateix pla Z i les dues càmeres tenen la mateixa distància focal, podem veure la projecció del punt P en les dues imatges. Per tant, es pot obtenir un valor de disparitat *d* per cada parell de punts aparellats *Pi(xi, yi)* i *Pd(xd, yd)* donat per d = XI-XD.

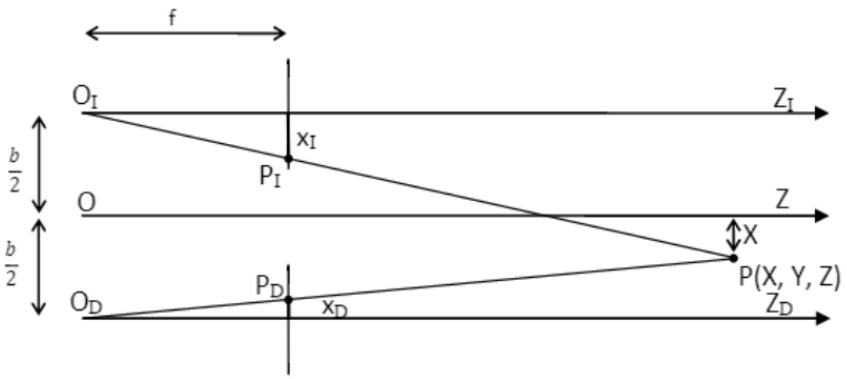


Fig. 4.24: Determinació de la posició de les imatges al sistema de coordenades cartesianes de Leap Motion a través de tècniques de visió estereoscòpica

Considerant igual la distància focal ***f*** en les dues càmeres i coneixent la distància entre càmeres ***b***:

$$\left.\begin{cases} O_I : \dfrac{\frac{b}{2}+X}{Z} = \dfrac{x_I}{f} \\ O_D : \dfrac{\frac{b}{2}-X}{Z} = \dfrac{x_D}{f} \end{cases}\right\} \rightarrow \quad \left.\begin{cases} x_I = \dfrac{f}{Z}\left(X+\dfrac{b}{2}\right) \\ x_D = \dfrac{f}{Z}\left(-\dfrac{b}{2}\right) \end{cases}\right\} \rightarrow \; d$$

Com es pot veure, a partir del sistema anterior es poden obtenir les coordenades del punt P.

4.1.3.2. Resum

En resum, els passos bàsics que realitza Leap Motion per entregar variables descriptives d'una configuració de mà i d'un gest són:

1. Obté les imatges des dels sensors i les càmeres del dispositiu.
2. Aplica una correcció de la distorsió que produeixen els sensors.
3. Aplica un model per determinar la configuració de cada mà i executa un algorisme de visió estereoscòpica entre cada par d'imatges per obtenir la posició en el pla tridimensional.

4.1.4. Anàlisi de la API

En aquest projecte, anem a analitzar la API[14] de Leap Motion des del punt de vista del llenguatge de programació C# /Unity, però és necessari saber que Leap Motion es pot programar amb els següents llenguatges de programació i plataformes de desenvolupament:

[14] https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html

Fig. 4.25: Llenguatges de programació i plataformes de desenvolupament de Leap Motion

Des de la API es pot obtenir tot tipus d'informació tridimensional en referència als avantbraços, mans, eines, dits i inclús als "ossos" dels dits que es tracten com a objectes. A més a més, des de la versió 2.1. es pot accedir a la informació de les imatges en format RAW que recull el dispositiu.

4.1.4.1. Visió general

El dispositiu fa servir un sistema de coordenades cartesianes destre. L'origen es centra a la part superior del controlador de Leap Motion. Des de la posició de l'usuari, l'eix Y és vertical i té valors positius creixent cap amunt, l'eix X cap a la dreta i l'eix Z cap a l'usuari.

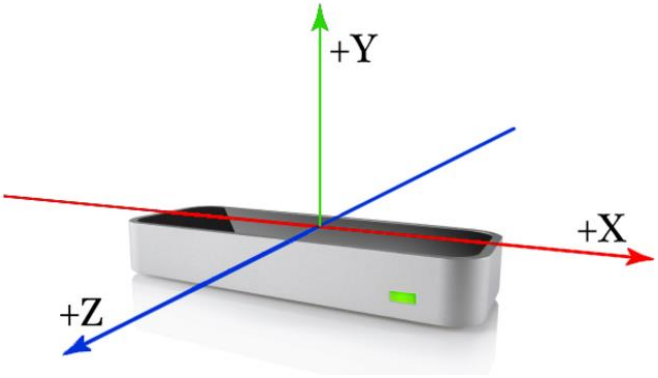


Fig. 4.26: Sistema de coordenades de Leap Motion

L'API de Leap Motion mesura les magnituds físiques amb les següents unitats:

Distància	Mil·límetres (mm)
Temps	Microsegons (µs)
Velocitat	mm/s
Angle	Radiants (rad)

A continuació, s'analitzaran els components principals d'aquesta versió. Es veuran quins són els principals objectes que el dispositiu pot rastrejar, juntament amb el model anatòmic que

s'utilitza per a identificar cada un, i també s'analitzaran les classes principals que descriuen la API.

4.1.4.2. Components principals : Controller()

El principal objecte és el Controller(), que s'encarrega de fer d'interfície entre l'aplicació que desenvoluparem i el dispositiu. A continuació podem veure els mètodes que conté:

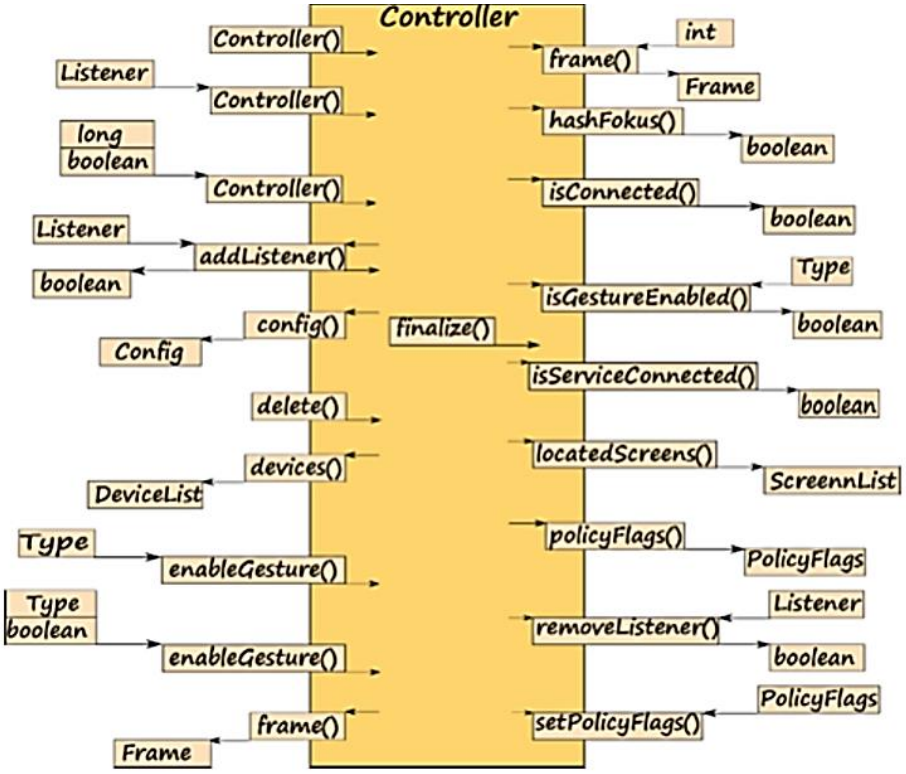


Fig. 4.27:Mètodes que conté l'objecte Controller()

El que més ens interessa d'aquesta classe és el mètode *frame()*, a través del qual podem accedir a l'objecte *Frame* que desitgem; per defecte, a l'últim que ha arribat des del dispositiu.

- Mètode *frame()*

Cada objecte Frame conté una instantània de l'escena gravada pel controlador Leap Motion. Les mans, els dits i les eines són les entitats físiques bàsiques a ser rastrejades pel sistema Leap Motion.

La API de Leap Motion presenta les dades de seguiment del moviment com una sèrie d'instantànies anomenades *frames*. Cada objecte Frame representant un *frame* conté

qualsevol rastrejat de mans detectat en aquesta instantània, que detalla les seves propietats en un sol moment en el temps.

- Mètode *onFrame()*

Un altre mètode interessant – al igual que en el cas del *Controller* – és el *onFrame()*, esdeveniment que es produeix cada cop que el dispositiu captura una imatge. Aquest és l'esdeveniment principal, ja que dintre d'aquest es poden implementar totes les accions que es volen fer sobre els objectes que veurem més endavant (mans, dits, gests, etc).

Així doncs, hi ha dos maneres d'accedir a un objecte Frame:

1. Des de l'objecte *Controller,* a través del mètode *frame()*
2. Des de l'objecte *Listener*, a través dels esdeveniments que es van disparant

Escollir un mètode o l'altre dependrà de si es vol analitzar o no tots els *frames* que arribin des del *Controller*. Si els volem analitzar tots, accedire, a la informació de l'objecte *Frame* a través de *Listener*; però si volem analitzar un objecte *Frame* cada cert temps, s'accediria des de l'objecte *Controller.*

4.1.4.3. Objecte *Frame* i les seves classes

A continuació, podem veure un diagrama de la relació entre l'objecte *Frame* i els quals es poden accedir a través d'ell:

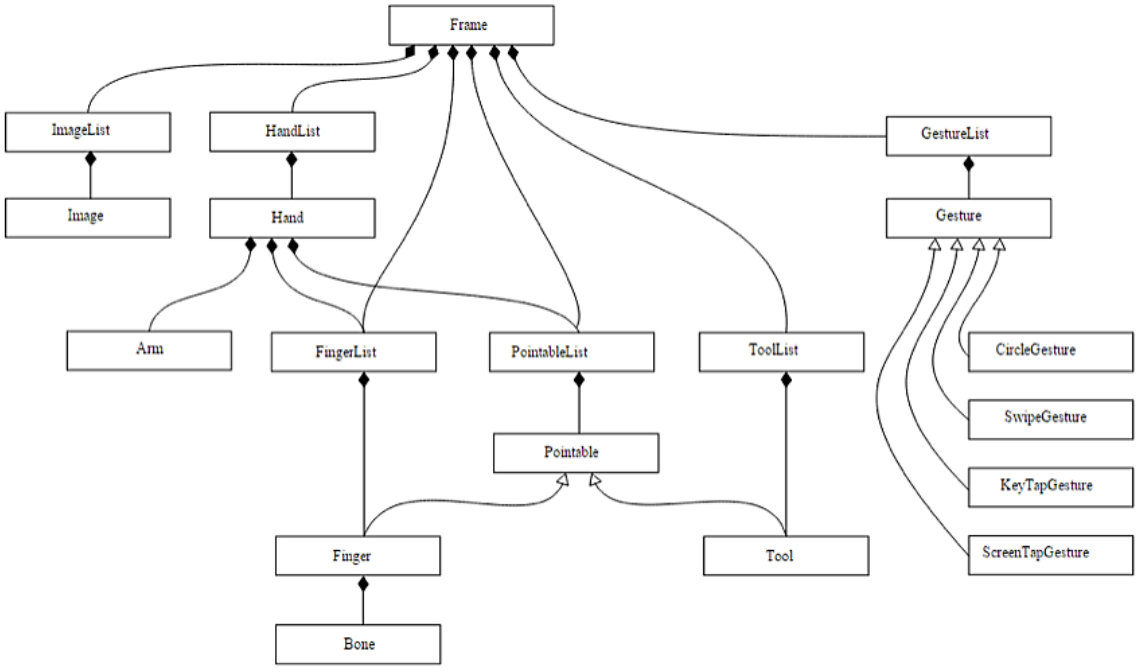


Fig. 4.28: Diagrama de relació entre l'objecte Frame i els altres objectes

Com veiem, l'objecte *Frame* és l'arrel de totes les dades dels objectes que analitza Leap Motion. *Frame* ens dóna la possibilitat d'accedir a totes les classes *List*, a partir de les quals es pot accedir a una llista d'objectes del mateix tipus i que apareixen en cada *Frame*.

La part que ens interessa és la que depèn de ***HandList*.** Aquesta classe és una llista d'objectes de tipus *Hand* que, al seu torn, conté objectes de tipus *Arm, Finger, Pointable i Tool* per a cada objecte *Hand*.

4.1.4.4. Classe *Hand* i els seus objectes

El model de la mà proporciona informació sobre la identitat, posició i altres característiques d'una mà detectada, el braç al qual s'uneix la mà, i tots els dits associats amb la mà.

Les mans estan representats per la classe Hand.

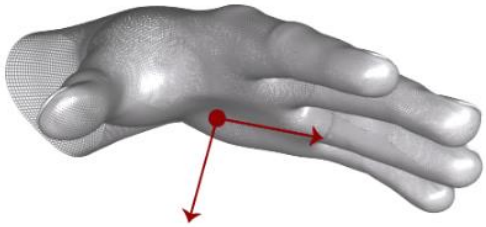


Fig. 4.29: Els vectors Hand PalmNormal i Direction defineixen la orientació de la mà.

El software Leap Motion utilitza un model intern d'una mà humana per proporcionar un seguiment predictiu inclús quan les parts d'una mà no són visibles. El model de la mà sempre proporciona posicions per a cinc dits, encara que el seguiment és òptim quan la silueta d'una mà i tots els seus dits són clarament visibles.

Dintre de l'objecte *Hand*, podem accedir a una llista d'objectes *Finger* per a cada *Hand*, en la qual podem accedir a informació de cada objecte *Finger* que apareix al *frame* i que forma part de l'objecte *Hand*. Al seu torn, cada objecte de tipus *Finger* conté objectes de tipus *Bone*. A més, com es pot observar, els objectes *Finger* i *Bone* són de tipus *Pointable*.

A continuació podem veure un esquema de les parts que componen l'objecte Hand amb la seva representació anatòmica:

Fig. 4.30: Esquema de les parts que componen l'objecte Hand

En la següent taula, es mostra la informació principal que es pot extreure de cada objecte que depèn de *Hand*:

Objecte	Informació
Hand	• Posició i velocitat del palmell • Direcció i vectors normals • Base ortonormal
Finger	• Posició i velocitat de la punta • Direcció i vectors normals • Base ortonormal • Amplada i longitud
Bone	• Posició de l'articulació • Base ortonormal • Amplada i longitud
Arm	• Posició del canell i colze • Vector de direcció • Base ortonormal • Amplada i longitud
Pointable Tool	• Posició i velocitat de la punta • Vector de direcció • Base ortonormal • Amplada i longitud

Tabla 4.1: Informació principal que es pot extreure de cada objecte de la classe Hand

- Fingers

El controlador Leap Motion proporciona informació sobre cada dit de la mà. Si tot el dit o una part d'ell no és visible, les característiques del dit són estimades basades en observacions recents i el model anatòmic de la mà. Els dits són identificats per nom: *thumb, index, middle, ring,* i *pinky*.

Els dits estan representats per una classe Finger.

Els vectors TipPosition i Direction del dit proporcionen la posició de la punta d'un dit i la direcció general on està apuntant el dit

Un objecte Finger conté un objecte Bone que descriu la posició i la orientació de cada os anatòmic del dit. Tots els dits contenen quatre ossos ordenats de la base a la punta:

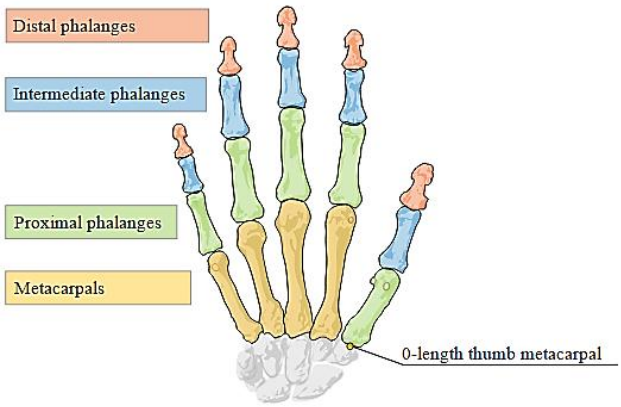


Fig. 4.31: Ossos dels dits de la mà

4.1.4.5. Classe HandList

A través d'aquesta classe es pot extreure un objecte del tipus *Hand* que apareixen en el *Frame* que s'està analitzant mitjançant el mètode *get()*. També es poden accedir als objectes *Finger* i *Arm* que depenen de cada objecte *Hand* en concret. Es a dir, podem extreure objectes del tipus *dit* i *avantbraç* de cada mà que apareix en la imatge.

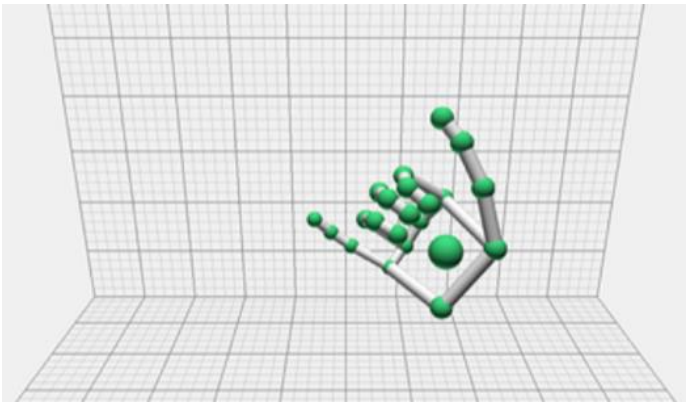


Fig. 4.32: Nou model de mà millorat respecte la versió 2.0

Com es pot observar, el model de mà es basa en una estructura de punts i línies que descriuen millor a un objecte de tipus *mà*. Amb aquest nou model apareix l'objecte *Bones*, que ha ajudat a que el dispositiu tingui millor precisió a l'hora de determinar la posició i el gest de la mà, ja que gràcies a la definició d'aquests punts si el dispositiu no veu algun d'ells pot determinar per exemple on està el dit que no es veu o en quina posició està la mà.

4.2. Incorporació del Leap Motion a Unity3D

4.2.1. Llibreria SDK de Leap Motion

El Leap Motion SDK conté dues biblioteques bàsiques que defineixen l'API per a les dades de seguiment del Leap Motion. Una biblioteca està escrita en C++, la segona està escrita en C. Les classes per a aquestes biblioteques defineixen enllaços de llenguatge per a C # i Objective-C.

Tota la biblioteca, el codi i els arxius de capçalera necessaris per desenvolupar aplicacions i complements compatibles amb Leap s'inclouen al SDK de Leap Motion, excepte la biblioteca JavaScript de client leap.js. Es pot descarregar el SDK de Leap Motion des del portal de desenvolupament de Leap Motion (s'indica a l'apartat següent com fer-ho). Un paquet SDK està disponible per cada sistema operatiu compatible. La biblioteca de client JavaScript es distribueix per separat i es pot descarregar des del repositori LeapJS GitHub.

Els complements per a Unity 5 i Unreal Engine 4 es proporcionen per separat del SDK principal. El complement Unreal està inclòs en Unreal Engine 4. El plugin està disponible a la pàgina de leapmotion developer.

4.2.2. Creació d'un projecte amb Unity 3D

Els passos que s'han de seguir a l'hora de crear un projecte amb Unity 3D [7] són els que s'indiquen a continuació:

1. Obrir l'editor de Unity
2. Seleccionar *File* > *New Project*
3. Escollir un nom per al projecte i guardar la seva ubicació
4. Clicar *Create Project*

Tot seguit, s'ha d'importar el paquet d'assets de Leap Motion al projecte:

5. Descarregar el v2 Desktop SDK de Leap Motion
6. Seleccionar al menú de comandaments *Assets* > *Import Package* > *Custom Package*.
7. Ubicar el paquet d'assets i seleccionar *Open*

Important: Si s'estan important els assets a un projecte que ja els conté, és recomanable que s'esborrin primer els assets antics (fer una còpia de seguretat). El procés d'importació de Unity només afegeix arxius nous i sobreescriu els arxius modificats.

4.2.3. Adició de mans a l'escena

- **HandController prefab**

El *LeapHandController* és un prefab que s'encarrega de consultar el Leap Motion Service per fer-ne un seguiment de les dades i les utilitza per col·locar les mans a l'escena. Les dades de seguiment del servei es transformen en relació a la posició i l'orientació del prefab a l'escena. Els scripts del controlador gestionen els objectes que representen les mans físiques detectades pel dispositiu Leap Motion.

Per afegir el prefab LeapHandController a una escena no-virtual, s'ha de col·locar el prefab a l'escena exactament sota d'on les mans haurien d'aparèixer.

- **Adició de les mans**

Un cop importat el SDK al projecte, col·locar el prefab HandController per sota del punt en què es desitja que apareguin les mans. Les mans en l'escena estan col·locades en la mateixa posició que les mans reals en relació amb el dispositiu Leap Motion. Es pot canviar l'escala de l'objecte HandController per canviar la mida de les mans i modificar la configuració de Hand Movement Scale per permetre que les mans es moguin dintre d'un

volum més gran.

Per canviar el model de les mans, s'ha d'afegir un prefab *hand* diferent al panell *Left/Right Graphics Model.*

Per afegir les mans a l'escena:

1. Al panell del Projecte, localitzar el prefab HandController a la carpeta Assets/LeapMotionPrefabs
2. Arrossegar el prefab HandController a la Scene.
3. Configurar la posició del controlador on es desitja.
   Per veure les mans, la HandController ha d'estar dins del camp de visió de la càmera.
4. Canviar l'escala de les mans per adaptar-les apropiadament a l'escena. Una escala d'1 vol dir que les mans es mostren a la seva mida real (que sovint són massa petites).
5. Establir les propietats de Left i Right Hand Model Graphics del HandController amb el prefab desitjat de les carpetes HandModelsHuman o HandModelsNonHuman. (Generalment, no hi ha necessitat de canviar el model de la física de RigidHand).
6. Provar l'escena > Play.

S'hauria de veure que les mans apareixen al panell Hierarchy, així com en la pestanya Game quan es posen les mans a sobre del dispositiu Leap Motion quan estem amb mode *play.* Si no es veuen les mans, obrir el Visualizer des del panel de control de Leap Motion per assegurar-se que el dispositiu està connectat i enviant les dades correctament. Si les mans estan fora de visió, aturar el joc i utilitzar la Scene per localitzar-les.

- **Accedir directament a la API de Leap Motion**

A més dels prefabs i els scripts inclosos en el paquet d'assets, es poden escriure els nostres propis scripts per accedir a les dades de seguiment des de l'API de Leap Motion. Les classes de Leap Motion estan definides pel Leap.

En el nostre cas, com que farem servir Unity i el llenguatge de programació C#, utilitzarem la funció "Update()" que es crida automàticament per Unity a cada frame per obtenir el "Frame" de Leap Motion.

A continuació, es mostra un petit exemple sobre com accedir a la API de Leap Motion:

```
using UnityEngine;
using System.Collections;
using Leap;

public class LeapBehavior : MonoBehaviour
{

    Controller controller;
    void Start()
    {
        controller = new Controller();
    }

    void Update()
    {
        Frame frame = controller.Frame();
        // resta del codi…
    }
}
```

Codi 1: Accés a la API del Leap Motion

- ***Key Components* del HandController prefab**

Els *Key Components* del prefab LeapHandController inclouen:

LeapHandController (Script)

La classe LeapHandController defineix la lògica per administrar dades de seguiment i les mans, No conté propietats que es puguin establir a l'Inspector.

Leap Service Provider

The Leap Service Provider és el punt de connexió entre el servei Leap Motion i la resta dels assets de Unity. El service provider obté marcs i imatges del servei i els hi proporciona altres parts de la seva aplicació.

- Is Head Mounted – ha d'estar habilitat per al seguiment adequat de la mà quan el hardware de Leap Motion està muntat en un HMD. Ha d'estar desmarcada quan s'utilitza el hardware de Leap Motion en una configuració d'escriptori (cap amunt)
- Override Device Type With:
    - Peripheral – el dispositiu estàndard

    - Dragonfly – prototip de dispositiu intern. Només s'hauria d'utilitzar si es té un dels pocs d'aquests dispositius que existeixen al món.

- Use Interpolation – permet un marc d'interpolació / extrapolació. Quan està activat, el servei proporciona una sèrie de dades interpolades o extrapolades per a que coincideixi amb el temps real quan s'executen els esdeveniments Unity Update o FixedUpdate.
- Interpolation Delay – ajusta el temps d'interpolació a una quantitat ajustada. Un retràs massa petit pot portar a una excessiva extrapolació, la qual pot ser reconeguda per sacsejades en els moviments de la mà i els dits. Un retard massa gran afegirà una latència excessiva.

Hand Pool

El Hand Pool gestiona la representació de les mans en una escena. Una sola mà pot tenir qualsevol nombre de GameObjects de Unity associats a ella. Per exemple, una representació podria ser per a la visualització de gràfics, una per a les interaccions de la física i una altra per la fixació d'objectes d'utilitat a la mà.

Cadascuna d'aquestes tasques específiques dels GameObjects s'han d'afegir a la hand pool en un element pool separat. Cada element de la pool ha de tenir un game object per a cada mà, la dreta i l'esquerra.

S'ha d'incrementar el Model Pool Size per poder afegir més elements. Per a cada element de la pool es poden fer els següents ajusts:

- Group Name – un sring arbitrari que li assignes per identificar el seu grup quan es criden les funcions EnableGroup() and DisableGroup() de l'objecte HandPool.
- Left Model – el gameobject o prefab per usar la mà esquerra.
- Right Model – el gameobject o prefab per usar la mà dreta.
- Is Enabled – quan aquest grup està actualment habilitat. Es pot canviar la configuració al Unity Editor o en temps d'execució amb les funcions EnableGroup() i DisableGroup().
- Can Duplicate – determina si més d'una instancia d'aquests game objects pot existir en una escena. Per exemple, si hi ha més d'una mà dreta o esquerra en una escena perquè el software de Leap Motion identifica erròniament la quiralitat d'una mà o dues persones diferents posen les mans al sensor, es genera un model addicional de mà quan aquest Can Duplicate és True. Quan és False, només la primera mà a la vista seria representada per un model d'aquest grup.

5. El motor de videojocs *Unity 3D*

5.1. Unity 3D

Unity3D és l'eina principal amb la qual s'ha desenvolupat aquest projecte. Com ja s'ha explicat en el punt 3 d'aquest treball, Unity3D és una eina de creació de videojocs amb una alta potència. Desenvolupar amb Unity a un nivell expert és bastant complicat d'aconseguir, no obstant és un programa bastant intuïtiu. A més a la seva web [1] hi ha una secció dedicada exclusivament a l'aprenentatge, on es poden trobar tutorials i tot lo relacionat amb la documentació a utilitzar.

Unity treballa principalment amb projectes i escenes. Un projecte és un videojoc o aplicació, i cada projecte pot tenir una o varies escenes on es poden afegir objectes de forma gràfica, i suporta codi escrit en tres tipus de llenguatges de programació orientats a scripts: C#, Boo i JavaScript. A continuació, s'explicarà com funciona la interfície de Unity3D i es definiran els conceptes més bàsics que s'han de conèixer a l'hora de començar a utilitzar l'editor.

5.1.1. Interfície de Unity

L'editor de Unity s'ha dissenyat de manera que totes les accions es poden fer arrastrant objectes, scripts i enllaçant-los els uns amb els altres. La interfície de l'editor de Unity té un aspecte que es pot personalitzar, depenent de la preferència personal o del tipus de treball que s'estigui realitzant, simplement canviant els panells de posició. Les finestres més comuns i útils es mostren a continuació en les seves posicions per defecte:

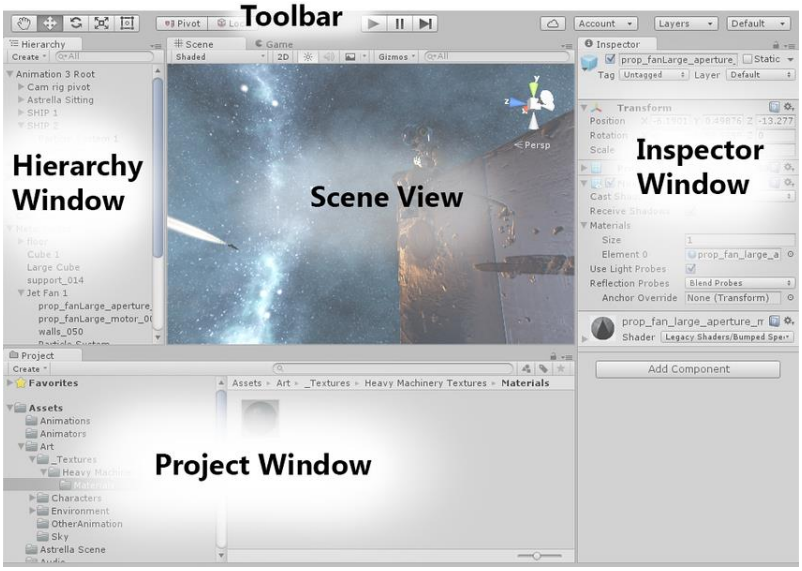


Fig. 5.1: Interfície d'usuari de l'editor de Unity 3D

- ***Project Window*** **(finestra del projecte)**

El panell Project Window mostra la informació d'una carpeta anomenada Assets, la qual és única per a cada projecte. En aquesta carpeta es guarden totes les altres carpetes i dades necessàries per la creació del projecte. Dintre d'aquesta carpeta existeixen vàries subcarpetes que seran tot el material que s'utilitzarà per la creació del projecte, per exemple: escenes, scripts, prefabs, materials, textures, etc. Quan s'importen assets al projecte, aquests apareixen aquí.

- ***Scene View*** **(vista de l'escena)**

  Permet una navegació visual de l'escena i editar-la. És on s'afegeixen i s'editen els objectes, les càmeres, la il·luminació i tot lo relacionat amb l'escena. La scene view pot mostrar una perspectiva 2D o 3D depenent del tipus de projecte en el qual s'estigui treballant.

- ***Hierarchy Window*** **(finestra de jerarquia)**

Aquesta finestra és una representació jeràrquica de cada objecte de l'escena. Cada element de l'escena té una entrada a la jerarquia, pel qual les dos finestres estan inherentment vinculades. La jerarquia revela la estructura de com els objectes estan agrupats els uns als altres.

- ***Inspector Window*** **(finestra de l'inspector)**

El inspector window permet visualitzar i editar totes les propietats de l'objecte actualment seleccionat. Com que diferents objectes tenen diferents propietats, el layout (disseny) i contingut de la finestra de l'inspector variaran.

- ***Toolbar*** **(barra d'eines)**

Proporciona un accés a les característiques més essencials per treballar. A l'esquerra hi ha les eines bàsiques per manipular la scene view i els objectes dins d'aquesta. Al centre, estan els controls de reproducció, pausa i passos. Els botons de la dreta donen accés als serveis de Unity Cloud i al compte de Unity, seguit d'un menú de visibilitat de capes i, finalment, el menú del layout de l'editor (que proporciona alguns dissenys alternatius per a la finestra de l'editor, i permet guardar layouts personalitzats).

La barra d'eines no és una finestra, i solament és part de la interfície de Unity que no es pot re-ajustar.

5.1.2. Editor gràfic

En cada escena, es poden crear objectes o importar-los des d'altres programes de modelat 3D. Cada objecte d'una escena és un "Objecte de Joc" (*GameObject*), on també són tractades les càmeres i les llums com GameObjects, per la qual cosa se'ls pot aplicar codi per fer que es comportin a gust del desenvolupador.

- L'**escena**: és l'escenari del joc, on hi hauran tots els objectes que siguin necessaris per al projecte.
- Les **càmeres**: és el dispositiu a través del qual l'usuari pot veure l'escena. S'ha d'utilitzar una càmera principal que apunti al lloc e l'escena que es vulgui fer visible per a l'usuari. També es poden utilitzar vàries càmeres per posar diferents punts de vista de l'escena.
- Les **llums**: es poden obviar, però amb elles es dóna un efecte 3D més realista.
- Els **GameObject**: són els "objectes" en Unity3D. Són entitats que els hi pots assignar forma, malla, textura, material, propietats i scripts, podent-se crear nombroses combinacions. Tots els elements que estan dins l'escena són GameObjects.
- La HandControllerSandBox: és una zona que delimita l'espai abastable pel moviment de les mans. És la zona de cobertura del controlador Leap Motion, i està definit com es mostra a la següent figura:

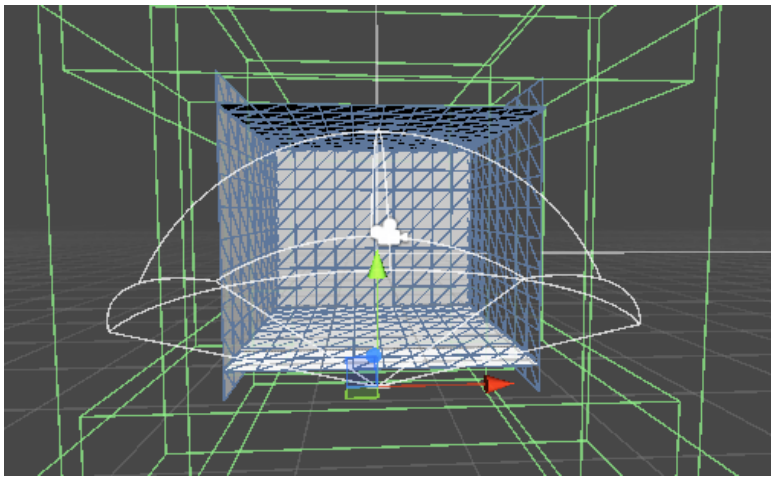


Fig. 5.2: HandControllerSandBox del sensor Leap Motion

Eines importants: els botons que hi ha a continuació, són les eines bàsiques per manipular la vista de l'escena i els objectes dintre d'aquesta. També s'ha de saber una combinació important de tecles per moure't bé dintre de l'escena amb el ratolí, que és prement la tecla Alt + el botó esquerre del ratolí.

Fig. 5.3: Eines bàsiques de manipulació de la scene view i els objectes

5.1.3. Estructura i programació

- Les **classes**: Unity3D inclou vàries classes pre-definides, de la qual s'ha de destacar la classe principal de la qual es solen heretar la majoria dels scripts: **MonoBehaviour**. Aquesta classe principal posseeix els mètodes *Start()* i *Update()*, entre d'altres. *Start()* és un mètode el qual el seu codi és executat un cop carregat el GameObject que conté l'script que l'implementa. *Update()*, d'altra banda, és un mètode que el seu codi s'executa constantment, pel qual és molt útil per a elements que necessiten ser constantment revisats i actualitzats.

    - *Start():* Aquest mètode es crida només a l'inici del joc. És molt útil per iniciar variables, posicions, estats, etc.
    - *Awake():* Aquest mètode escrida just un fotograma abans que *Start(),* pel qual també és pràctic per iniciar variables, sobretot variables que puguin tenir conflictes de temps amb *Start().*
    - *Update():* Aquest mètode és cridat una vegada per cada fotograma del joc, en temps d'execució. Dintre seu s'ha d'incloure tot el codi que es necessita actualitzar un cop per fotograma, com el moviment o la detecció de canvis. Per això és el més important a l'hora de desenvolupar el videojoc.
    - *OnCollisionEnter():* Aquest és un dels mètodes predefinits més útils. A aquest mètode se'l crida cada cop que l'objecte que porta aquest Script topa amb un altre objecte de l'escena.
    - *OnTriggerEnter():* Un component del tipus *Collider* pot ser marcat com a *Trigger.* Això significa que els objectes que topen amb ell no reaccionaran com si es tractés d'una col·lisió real, sinó que passaran de llarg. No obstant això, quedarà registrat que un objecte ha entrat a l'àrea definida pel col·lisionador. Aquest mètode detecta aquesta intrusió en aquesta àrea. És molt útil per crear sistemes de detecció.

- La **estructura de directoris**: s'ha de seguir una estructura de directoris per a una major facilitat a l'hora de fer el programa. La carpeta arrel és la de "Assets". Dintre d'aquesta carpeta aniran tots els recursos que s'utilitzaran: materials, textures, scripts, escenes, etc., cadascun a la seva respectiva carpeta, per a un major ordre.
- Els **materials**: per donar més realisme als objectes de Unity, normalment se'ls assigna un material. El material pot tenir física (elasticitat, rebot, etc.) però el que més el caracteritza és la seva textura.

- Les **textures**: a cada material se li assigna una textura, les quals consten d'un color o d'una imatge renderitzada sobre aquest material i una lluminositat. Gràcies a les textures es pot donar un efecte realista als objectes.
- Els **prefabs**: es tracta d'objectes que s'han creat i per als quals s'ha definit una sèrie de propietats que s'han guardat per utilitzar vàries còpies del mateix tipus. D'aquesta manera si es va a utilitzar molts cops un objecte amb les mateixes característiques no s'han de definir totes les propietats des de zero.

5.2. Estructura de l'editor

5.3. Conceptes bàsics

5.3.1. GameObjects i Components

Els GameObjects són els objectes que s'afegeixen a l'escena i tot el que hi ha en ella són GameObjects. Cada GameObject consta d'una sèrie de components, que es poden afegir o treure per caracteritzar l'objecte. Aquests components es poden modificar des de la interfície de Unity, però també es pot accedir a ells des de codi, per canviar les seves característiques en temps d'execució, si es requereix. Alguns dels components més importants, que molts GameObjects d'un joc solen tenir són els següents:

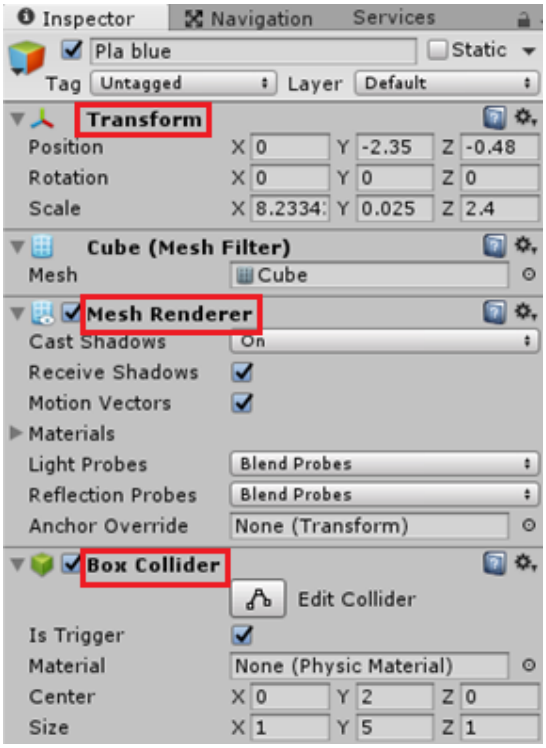


Fig. 5.4: Panell Inspector amb els components d'un GameObject

- ***Transform***: és el component més bàsic que tot GameObject ha de tenir. Aquest component posiciona i dimensiona el GameObject dins de l'escena virtual.

- ***MeshRenderer***: aquest component s'encarrega de renderitzar un objecte en temps d'execució. Per això, tot aquell objecte que volem que es vegi en una escena ha de comptar amb aquest component.

- ***Colliders***: són àrees que envolten al GameObject, que són utilitzades per comprovar les col·lisions entre els diferents objectes del joc. Hi ha diferents tipus de col·lisionadors:
  - *BoxCollider*: col·lisionador de caixa. Es crea una àrea amb forma de cub al voltant de l'objecte.
  - *SphereCollider*: col·lisionador d'esfera. Es crea una àrea amb forma d'esfera al voltant de l'objecte.
  - *MeshCollider*: col·lisionador de malla. El col·lisionador s'acobla a la malla de l'objecte al que se li assigna. És una manera d'obtenir col·lisions d'una forma molt precisa, però consumeix molts de recursos.

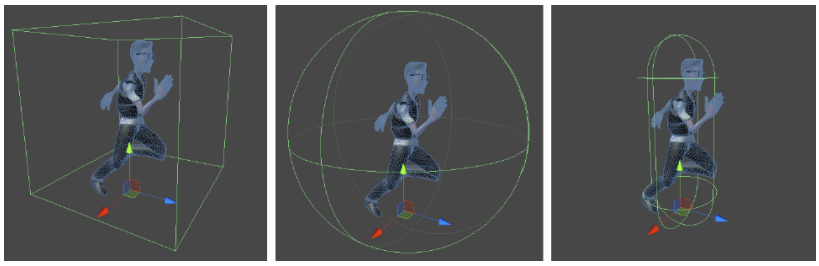


Fig. 5.5:BoxCollider, SphereCollider i CapsuleCollider.

- **RigidBody**: Adjudica al GameObject algunes característiques que tindria un cos a la realitat: massa, gravetat, fricció, etc. És molt útil per fer que objectes de la escena interaccionen entre sí.

- **CharacterController**: Normalment s'afegeix aquest component al personatge principal i/o als que hagin de ser controlats pel jugador. *CharacterController* conté propietats d'interacció amb l'entorn.

- **Animation**: Tot GameObject que contingui animacions, ha de tenir aquest component. En ell s'especifiquen el nombre d'animacions i el rang de fotogrames que ocupa cada animació. A més, serveix de referència a la hora de controlar aquestes animacions des de codi.

- **AudioSource**: Qualsevol GameObject que emeti sons ha de tenir associat aquest component. Funciona com un magatzem de sons i proporciona opcions per la reproducció d'aquests.

- **AudioListener**: Aquest component actua com una oïda, capta els sons emesos pels objectes que tenen associat un component de tipus *AudioSource* i els reprodueix per l'altaveu del dispositiu. En una escena, només pot haver un únic objecte que tingui aquest component, normalment és la càmera principal qui el conté.

- **Script**: Un Script és una peça de codi afegida com a component a un GameObject.

6. Disseny i construcció de l'aplicació

En aquest apartat s'expliquen les diferents escenes que s'han desenvolupat incorporant el Leap Motion amb el Unity, quin n'ha estat el procediment i amb quina finalitat s'ha creat cadascuna d'elles.

La creació d'aquestes escenes serveix per verificar que es pot utilitzar el Leap Motion amb el Unity per crear aplicacions de realitat virtual, en aquest cas una aplicació d'escriptori "*desktop*". S'han programat les escenes per tal de que el Leap Motion funcioni de dos maneres; una mitjançant la interacció amb els objectes de l'escena i l'altra mitjançant el reconeixement gestual de les mans.

La primera forma, consisteix en que les mans interactuïn amb els objectes de l'escena, és a dir, que sigui l'usuari qui determini la posició dels elements i pugui moure'ls amb les seves mans. En la segona forma, les mans no interaccionen amb els objectes directament, sinó que es tracta bàsicament de reconeixement de gests de les mans i els dits. Mitjançant codi, es programa quins gests han de ser reconeguts pel sensor i què ha de passar a l'escena.

6.1. Inici d'aprenentatge

A l'inici del treball, s'havia de planejar com es començaria a treballar i quins serien els passos a seguir durant el desenvolupament del projecte. Com que es partia amb experiència i coneixements nuls sobre el programa Unity, s'havia de dedicar força temps a l'inici per familiaritzar-se amb l'editor i el nou codi de programació C# utilitzat per als scripts.

6.1.1. Primera escena d'aprenentatge

Per començar a fer servir el Unity, es va començar amb tutorials com *Roll a ball* i *Space Shooter*, disponibles a la pàgina oficial de Unity tutorials [1]. Un cop ja s'havien après les eines bàsiques, es va decidir crear una escena pròpia començant des de zero. L'objectiu de la creació d'aquesta escena era implementar noves tècniques de creació i importació de GameObjects, implementació de nous scripts associats a aquests GameObjects i disseny d'un escenari utilitzant diferents tipus de materials, terrenys i textures.

L'escena es pot visualitzar a continuació:

Principalment, aquesta escena es va dissenyar per poder aprofitar posteriorment, en la creació de l'aplicació principal, algunes de les tècniques utilitzades i així poder-les vincular també amb el dispositiu Leap Motion:

- Control del personatge: per poder tenir control sobre moviment del personatge amb les fletxes del teclat ← ↑ → ↓ i amb Espai per saltar, per a això s'havia de generar un script anomenat *Movement*, que controlés aquestes opcions.
- Treballar amb terrenys: hi ha una eina al toolbar de Unity que et permet inserir terrenys i modificar-los a gust de l'usuari. Aquesta es troba a *GameObject > 3D Object > Terrain*, el que fa és inserir un pla on hi pots anar afegint deformacions (muntanyes, turons...) i també la vegetació desitjada. És una eina molt bona que usada amb habilitat pot fer que quedin escenes molt atractives.

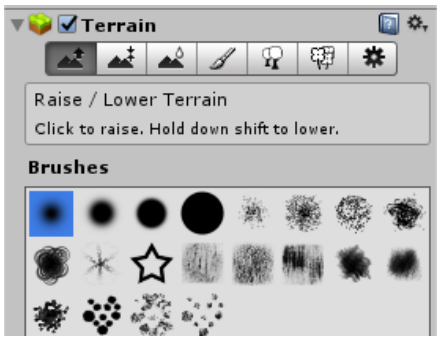


Fig. 6.1: Terrain Settings a l'Inspector de Unity

- Disparar bales: aquesta funció s'ha volgut aprendre principalment per que ja es tenia en ment dissenyar una escena utilitzant el Leap Motion en la qual es puguin disparar "bales" amb el dit. Més tard s'explicarà com fer-ho en l'escena amb el Leap Motion incorporat.

6.2. Descripció del directori de treball

A la carpeta arrel del treball, "Assets", s'han anat afegint totes les altres subcarpetes necessàries per al desenvolupament de l'aplicació. El treball en sí està guardat a la carpeta "TFG", on podem distingir les subcarpetes: *escenes prova, materials, prefabs, scenes, scripts* i *textures*.

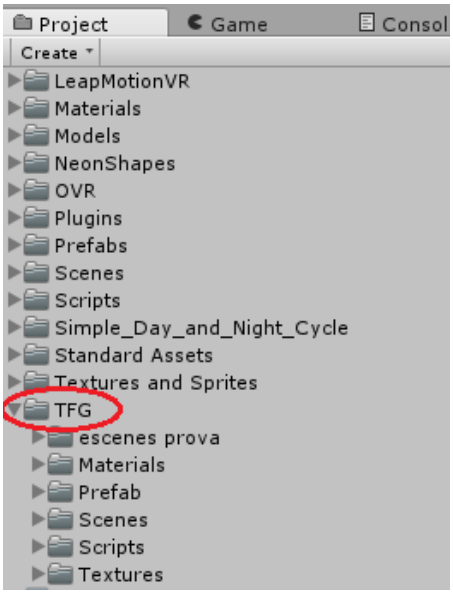


Fig. 6.2: Directori de treball de l'aplicació

6.3. Interfície d'usuari

L'estructura de l'aplicació s'ha dissenyat amb una interfície d'usuari que és la primera escena que apareix quan s'executa l'aplicació. La primera pantalla que apareix és la d'un menú, amb les opcions *Start* o *Exit*. La finalitat d'aquesta escena és poder accedir a cadascuna de les escenes que formen l'aplicació i provar qualsevol d'elles quan l'usuari ho desitgi. Quan es clica el botó Start, apareix una altra pantalla amb un menú d'escenes, el qual permet triar quina de les escenes es vol jugar. Quan es selecciona una escena, s'obre al costat esquerre un altre panell explicant el funcionament de l'escena.

6.3.1. Canvas

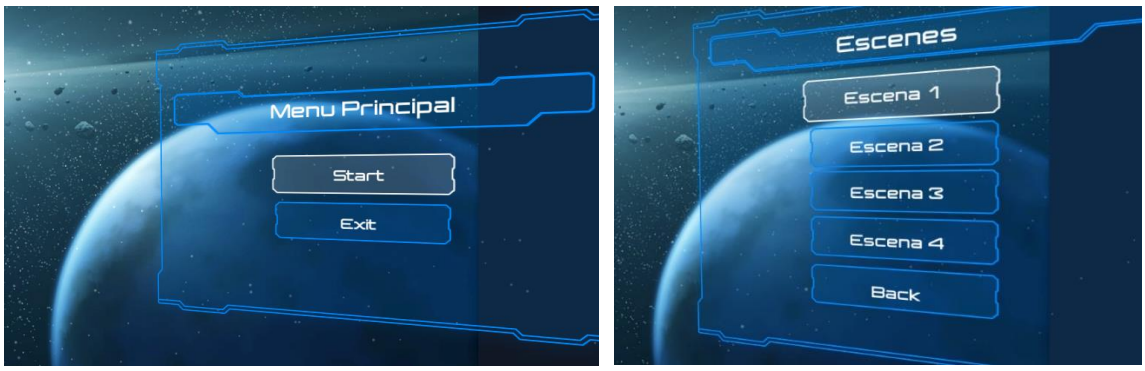


Fig. 6.3: UI - Menú principal i menú d'escenes

6.4. Estructura de l'aplicació

Seguidament, s'explicarà com s'ha estructurat l'aplicació i com s'han programat i dissenyat les diferents escenes, i amb quina finalitat. Com s'ha comentat anteriorment, l'escena 1 s'ha desenvolupat per que puguem interactuar amb els objectes de l'escena entrant "en contacte" amb ells. Les altres tres escenes, s'han programat per que reconeguin diferents gests de la mà i que serveixi per canviar l'escenari.

6.4.1. Escena 1: Manipulació d'objectes

- **Objectiu**

En aquesta escena, hi apareixen cubs de 3 colors diferents i al terra 3 panells també d'aquests colors. La finalitat d'aquesta escena és col·locar els cubs d'un color al rectangle corresponent utilitzant les nostres mans. El que s'ha d'intentar a l'hora de moure els cubs és tancar el polze i l'índex, com si féssim un pessic, i traslladar els cubs al panell que els toca.

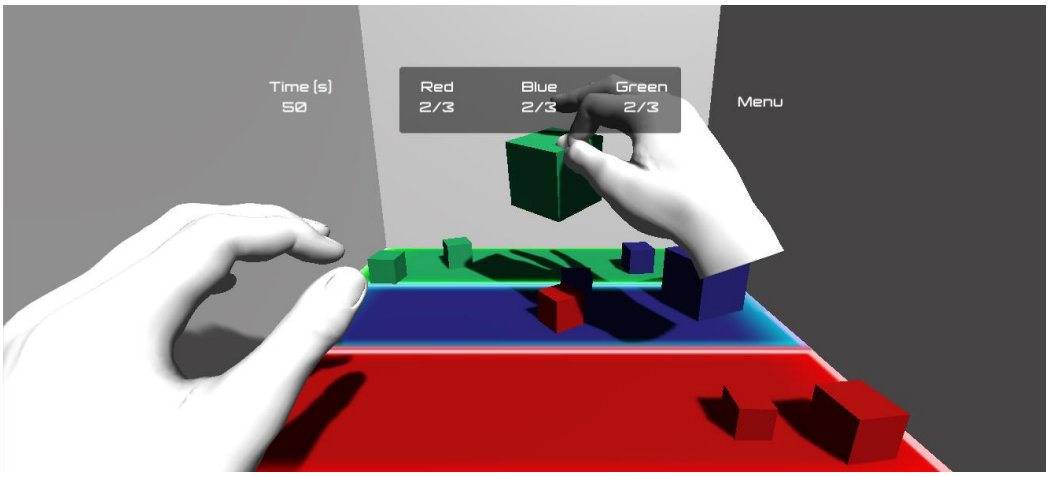


Fig. 6.4: Visualització de l'escena 1 en mode Play

- **Procediment**

L'escena està formada per GameObjects bàsics. Hi apareixen tres plans de colors vermell, blau i verd, i nou cubs de diferents mides (3 de cada color), un comptador del temps que ha transcorregut des del començament de l'escena (*canvas*), un comptador sobre el total del número de cubs que hi ha al panell que li correspon (*canvas*) i els components de *rendering* necessaris per poder visualitzar l'escena (com són la

il·luminació, càmera, etc).

En aquesta escena, com en les següents, no s'explicarà detall a detall tot el desenvolupament de l'escena, ja que hi ha coses que es sobreentenen o que són bastant lògiques de fer (com per exemple, per assignar un color diferent a cada cub, cal arrossegar dintre de l'escena el material del color corresponent a sobre seu), i per tant només s'expliquen les característiques importants referents al Leap Motion, els scripts que permeten el control de les mans, i la utilització dels *canvas* per crear els comptadors.

Afegir mans a l'escena

Com ja s'ha explicat al punt 4, és indispensable tenir el pack d'assets del Leap Motion *(Core Assets depelop; SDK v2)* importat al directori de carpetes del projecte. Recordem resumidament que per afegir les mans al projecte s'ha d'arrastrar el prefab LeapHandController (carpeta *Leap Motion* > *Prefabs*) a l'escena, que és qui crea la instància de les mans i les eines per representar-les a l'escena. En el cas d'aquesta escena, s'ha decidit no afegir el prefab LeapHandController, sinó que s'ha utilitzat un altre prefab anomenat HandControllerSandBox (*Leap Motion* > *Prefabs* > *HandModelsNonhuman*) el qual es mostra com una "caixa" que representa la zona de treball de les mans dins l'escena (explicat anteriorment al punt 5, editor gràfic de Unity). En aquesta escena s'ha decidit fer-ho així per provar diferents tipus de controladors del Leap Motion, i també pel tipus de joc que és era millor utilitzar un espai tancat i delimitat per parets.

Existeixen molts models de mans diferents, entre les quals hi ha models de mà humanes i models de mà robòtiques o no reals. S'havia de configurar quin tipus de mans es volien per l'escena, i per a això s'ha d'accedir a l'inspector tenint seleccionat el HandControllerSandBox. Com es pot veure, es pot configurar la mida de les mans canviant la **Scale** al panell de Transform**.** El ***HandController*** **script** va unit al prefab HandController. Quan el HandController està actiu a una escena, afegeix els models de mans 3D especificats cada cop que les mans físiques són rastrejades pel dispositiu Leap Motion. Per defecte, aquests objectes es destrueixen quan les mans físiques es perden.

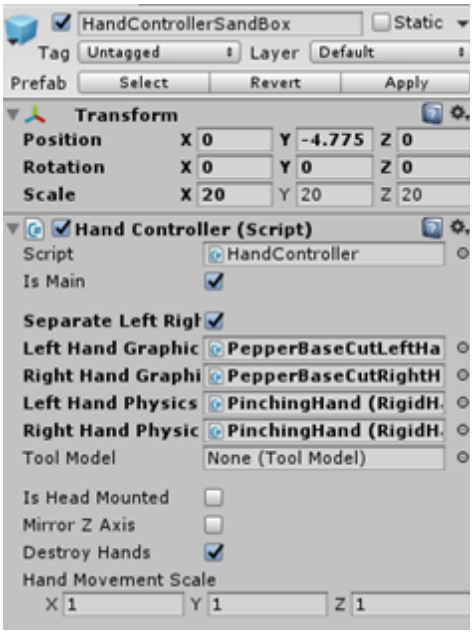


Fig. 6.5: Inspector del prefab de LM HandControllerSandBox

Llavors, en aquest cas s'han afegit un model de mans esquerre i dret (en alguns casos no ho separa), les quals s'han d'adjuntar a **Left Hand Graphics Model** i **Right Hand Graphics Model.** El model que s'ha adjuntat és el prefab *PepperBaseCut* que es tracta d'unes mans humanes de color blanc, com les que es veuen a la imatge. Aquest prefab es troba a la carpeta *Leap Motion* > *Prefabs* > *HandModelsHuman* .

A continuació, s'explicarà quins scripts s'han necessitat per poder programar la interacció de les mans amb els cubs de l'escena.

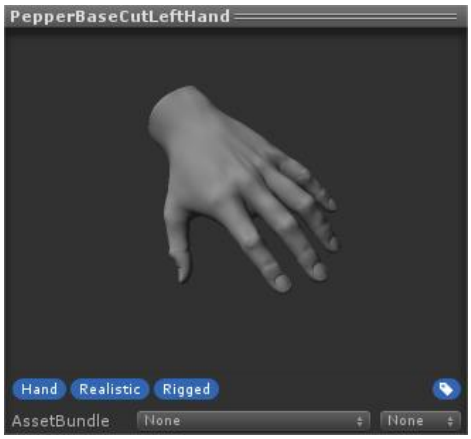


Fig. 6.6: Model de mans de l'escena 1

- **Scripts**
  - *GlobalScript:* Aquest script és el que controla el joc. S'encarrega bàsicament de comptar els cubs que es troben al panell del seu color, i de mostrar-ho al menú a la part de dalt de la pantalla.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class GlobalScript : MonoBehaviour {
    public static int redcounter;
    public static int bluecounter;
    public static int greencounter;
    public static int totalred=3;
    public static int totalblue=3;
    public static int totalgreen=3;

       public static Text redText;
       public static Text time;
       public static Text blueText;
       public static Text greenText;

       // Use this for initialization
       void Start () {
              redText =
GameObject.Find("Canvas/Panel/RedCounter").GetComponent<Text>();
              blueText =
GameObject.Find("Canvas/Panel/BlueCounter").GetComponent<Text>();
              greenText =
GameObject.Find("Canvas/Panel/GreenCounter").GetComponent<Text>();
              time =
GameObject.Find("Canvas/Panel/TimeCounter").GetComponent<Text>();
```

```
            greenText.text = greencounter + "/" + totalgreen;
            blueText.text = bluecounter + "/" + totalgreen;
            redText.text = greencounter + "/" + totalgreen;
            time.text = " "+ (int) Time.timeSinceLevelLoad;

      }

    void Update()
    {
            greenText.text = greencounter + "/" + totalgreen;
            blueText.text = bluecounter + "/" + totalgreen;
            redText.text = greencounter + "/" + totalgreen;
            time.text = " "+ (int) Time.timeSinceLevelLoad;
        if(bluecounter==totalblue)
        {
            print("Blue completed");
            if (redcounter == totalred)
            {
                print("Red completed");
                if (greencounter == totalgreen)
                {
                    print("Green completed");
                    print("You win");
                }
            }
        }
    }

}
```

Codi 2:Global Scriot que controla l'escena 1

- *TriggerControllerBlau:* Aquest script en concret, està programat per als cubs de color blau, però s'ha de saber que també existeixen dos scripts iguals que aquest però configurats per als cubs vermell i verd, anomenats *TriggerControllerVermell* i *TriggerControllerVerd*, respectivament.
  En aquest script, es defineixen unes etiquetes "tags" per a cada cub: "cubBlau", "cubVermell" i "cubVerd". Cada script està associat al panell del color que li correspon, i la seva funció és mirar i comptar cada cop que cau un cub del mateix color que el del panell a sobre seu, i s'ho guarda en una variable *bluecounter*. Així pot comparar la variable *bluecounter* amb la variable *totalblue* i mirar si hi són tots els cubs o no. Aquest recompte es mostra a un panell a la part de dalt de la pantalla en forma de menú.

```
using UnityEngine;
using System.Collections;

public class TriggerControllerBlau : MonoBehaviour {
```

```
        // Update is called once per frame
        void Update () {

        }

    void OnTriggerEnter(Collider other)
    {
        if(other.tag=="cubBlau")
        {
            GlobalScript.bluecounter++;

            print("Total Blau: "+GlobalScript.bluecounter);
        }

    }
    void OnTriggerExit(Collider other)
    {
        if (other.tag == "cubBlau")
        {
            GlobalScript.bluecounter--;
            print("Total Blau: " + GlobalScript.bluecounter);
        }
    }

}
```

Codi 3:TriggerControllerBlau, detecció dels cubs blaus al seu panell

- *GrabbingHand:* Aquest script és el que controla el *grabbing* dels objectes quan fem un *pinch* amb la mà, és a dir, permet agafar i moure els rigidbodies més propers quan tanquem els dits índex i polze amb qualsevol de les dos mans.

  No es copiarà en la memòria tot l'script, ja que ve incorporat en el pack d'assets de leap motion i no fa falta copiar aquí més de 300 línies de codi.

  Unes de les variables més importants que són necessàries per poder crear l'script són les següents:

  - Relació de la longitud de l'ós proximal del polze que activarà un pessic:

    ```
    public float grabTriggerDistance = 0.7f;
    ```

  - Relació de la longitud de l'ós proximal del polze que activarà un alliberament:

    ```
    public float releaseTriggerDistance = 1.2f;
    ```

  - Màxima distància d'un objecte que podem agafar quan el pessiguem:

    ```
    public float grabObjectDistance = 2.0f;
    ```

- Si l'objecte queda lluny del pessic es trencarà el vincle:

```
public float releaseBreakDistance = 0.3f;
```

6.4.2. Escena 2: Nit i dia

- **Objectiu**

L'objectiu d'aquesta escena és tenir control sobre el dia i la nit amb el dit índex de la mà. Bàsicament, s'ha programat per que el sensor Leap Motion detecti que el dit índex fa de polsador, és a dir, baixant-lo i pujant-lo per cambiar de la nit al dia. Com es pot apreciar, a les següents figures:

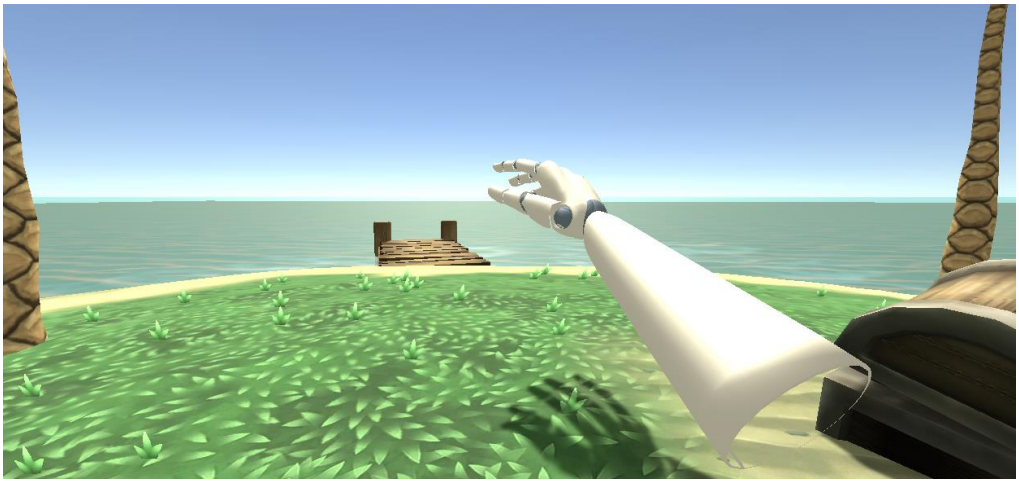


Fig. 6.7: Imatge de dia de l'escena 2

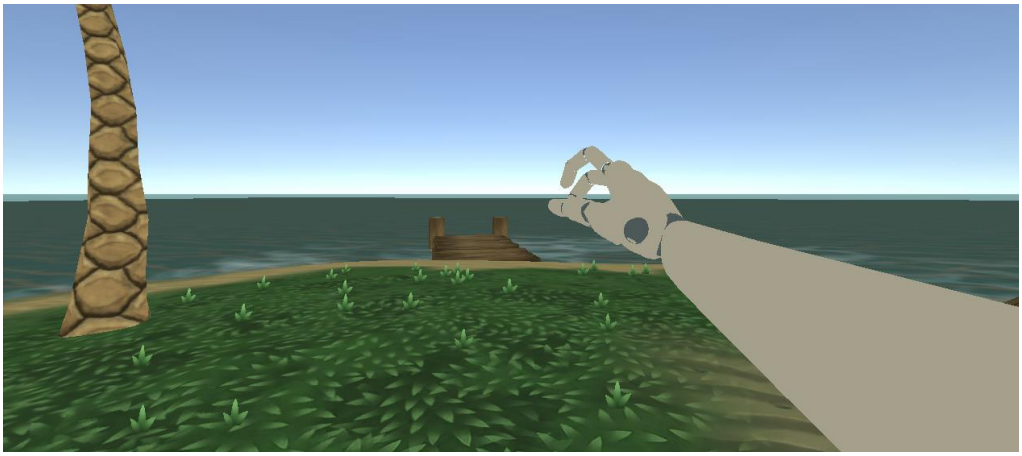


Fig. 6.8: Imatge de nit de l'escena 2

- **Procediment**

L'escena està formada per gameObjects més complexos que en l'anterior. Aquesta consta d'un pla que simula l'aigua del mar, un terreny on s'ha construït la petita illa i els arbres, també consta d'objectes de decoració com ara un baül o un pont de fusta, un component de vent anomenat *WindZone*, i un GameObject anomenat Capsule que actua com a personatge, amb el qual podem moure la vista de l'escena amb el cursor del ratolí i el personatge amb les lletres "w" (avant), "s" (avall), "a" (esquerra) i "d" (dreta).

Com s'ha explicat en l'escena anterior, també s'ha afegit el controlador de Leap Motion. En aquest cas el model de mans és el prefab *CleanRobotFullHand* que es troba a *Leap Motion > Prefabs > HandModelsNonhuman.*

- **Scripts**

A continuació s'explicaran els scripts més importants que s'han hagut d'utilitzar per la creació de l'escena:

- *GameController*: *GestureRecognition_TYPEKEYTAP*

Aquest és l'script més important de l'escena. Es crida una classe anomenada *Gesture,* i s'utilitza una de les seves subclasses que representa un gest que es reconegut pel software de Leap Motion. En aquest script es recineixerà el gest *KeyTap*, que és reconegut quan la punta del dit gira cap avall, cap al palmell de la mà i després salta cap enrere fins aproximadament la posició original. El gest Key Tap és equivalent a teclejar un teclat, és a dir, un gest ràpid del dit en sentit descendent. S'ha de tenir en compte que es tracta d'un gest discret.

Fig. 6.9: Representació del gest KeyTap [8]

A continuació està detallat l'script GestureRecognition_TYPEKEYTAP.

```
using UnityEngine;
using System.Collections;
using Leap;

public class GestureRecognition_TYPEKEYTAP : MonoBehaviour {

    Leap.Controller controller;
    private bool activat = true;

    void Start () {
        controller = new Controller();
        controller.EnableGesture(Gesture.GestureType.TYPEKEYTAP);
    // Per configurar les característiques del gest
        controller.Config.SetFloat("Gesture.KeyTap.MinDownVelocity", 20.0f);
        controller.Config.SetFloat("Gesture.KeyTap.HistorySeconds", 2f);
        controller.Config.SetFloat("Gesture.KeyTap.MinDistance", 1.0f);
        controller.Config.Save();
    }

    // Update is called once per frame
    void Update() {
        Frame frame = controller.Frame();

        foreach (Gesture gesture in frame.Gestures())
        {

            switch (gesture.Type)
            {
                case (Gesture.GestureType.TYPEKEYTAP):
                    {
                        GameObject DirectionalLight =
GameObject.Find("DirectionalLight");
                        if (activat)
                        {
                            activat = false;
                            DirectionalLight.GetComponent<Light>().intensity =
0;
                        }
                        else
                        {
                            activat = true;
                            DirectionalLight.GetComponent<Light>().intensity =
1;
                        }
                        print("type keytap");
                        break;
                    }
            }
        }
    }

}
```

Codi 4: GestureRecognition_TypeKeyTap, reconèixer el gest KeyTap

- *MouseLook:* **[Annex 1]**

S'explicarà l'script a continuació però s'adjuntarà al final de la memòria a l'apartat dels annexos.

L'script *MouseLook* gira la vista de la càmera quan es mou el cursor del ratolí. Es poden ajustar els valors de les variables Minimum i Maximum per limitar la possible rotació de la càmera.

Els passos que s'han de seguir per crear un personatge com aquest (estil FPS) són :

- Crear una *capsule* (gameObject)
- Afegir l'script MouseLook a la *capsule*.
- Establir el bloqueig del ratolí per que només es mogui en l'eix x, utilitzant *Lookx*. (Només es vol donar caràcter, no es vol que s'inclini)
- Afegir l'script *FPSInputController* a la *capsule*.
- S'afegiran automàticament els components *CharacterMotor* i *CharacterController*.

Per crear una càmera:

- S'ha de crear una càmera i afegir-la com a fill de la *capsule*. Restablir la posició (Reset) al panell de *Transform*.
- Afegir l'script MouseLook a la càmera.
- Establir el bloqueig del ratolí per poder usar el LookY, ja que es vol que la càmera s'inclini cap amunt i cap avall, com si fos un cap.

- *CharacterMovement:* **[Annex 2]**

Aquest script (programat amb Js) s'ha dissenyat per dotar la capsule de moviment amb les tecles "w" (avant), "s" (avall), "a" (esquerra) i "d" (dreta). Les variables més importants que es creen són la *velocitat* a la que es mourà la capsula, gravetat, controller (el component CharacterController) i un vector3 de moviment (moveDirection: Vector3).

6.4.3. Escena 3: Spinning Wheel

- **Objectiu**

L'objectiu principal d'aquesta escena és aconseguir que la roda giri en el sentit que nosaltres li indiquem amb el dit. Si el dit gira en sentit de les agulles del rellotge, la roda girarà en aquest sentit, i viceversa.

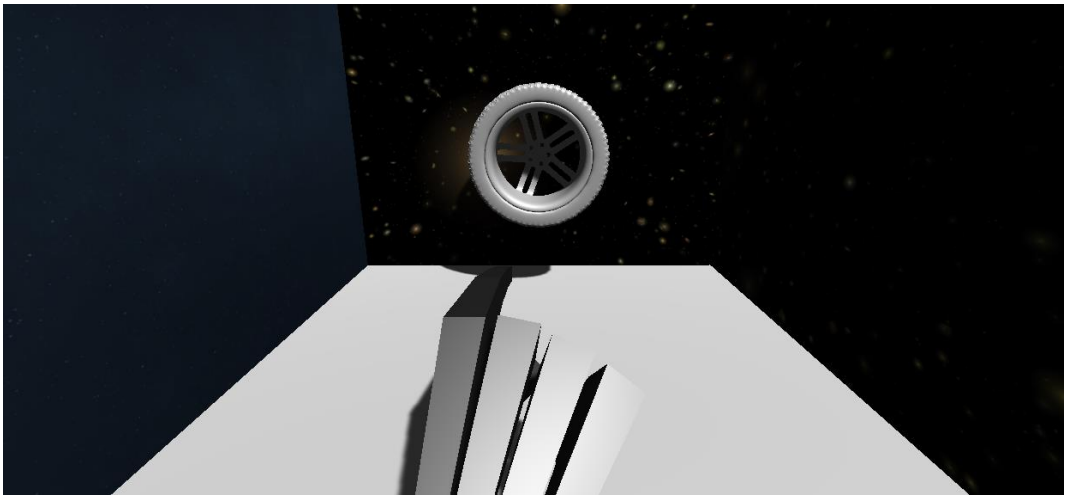


Fig. 6.10: Visualització de l'escena 3 en mode play

- **Procediment**

Com es pot observar, l'escena no està composta de GameObjects molt complicats. S'ha dissenyat aquest SandBox, que és com una caixa que abasteix la zona de cobertura de les mans, i s'ha importat un model de roda des de la pàgina web de t3dm [9]. Com en les dos escenes anteriors, s'ha afegit el controlador de Leap Motion amb un model de mans **PolyHand1** (*Leap Motion* > *Prefabs* > *HandModelsNonhuman*).

- **Scripts**

- *GameController*: *GestureRecognition_CIRCLE*

Aquest és l'script, com en l'escena anterior, que permet reconèixer quan s'està realitzant un cercle amb un dit o qualsevol eina. Un cop s'inicia el gest, el software de Leap Motion actualitzarà el progrés fins que acabi el gest. Aquest gest és continu, i acaba quan el dit en cercles o l'eina es para o es mou massa lent.

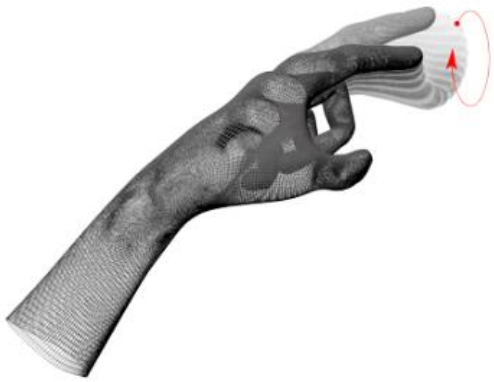


Fig. 6.11: Representació del gest Circle [8]

Per poder reconèixer gests, s'han d'habilitar per que es reconegui com a tal, amb la instrucció:

```
controller.EnableGesture(Gesture.GestureType.TYPECIRCLE);
```

el qual el MinRadius està expressat en *mm* i el MinArc està expressat en *radiants*, per tant un *float*pi*.

A continuació es mostra l'script GestureRecognition_CIRCLE:

```
using UnityEngine;
using System.Collections;
using Leap;

public class GestureRecognition_CIRCLE : MonoBehaviour
{
    Leap.Controller controller;

    void Start()
    {
        controller = new Controller();
        controller.EnableGesture(Gesture.GestureType.TYPECIRCLE);

        controller.Config.SetFloat("Gesture.Circle.MinRadius", 1.5f);
        controller.Config.SetFloat("Gesture.Circle.MinArc", 6f);
        controller.Config.Save();

    }

    void Update()
    {

        Frame frame = controller.Frame();

        foreach (Gesture gesture in frame.Gestures())
        {
            switch (gesture.Type)
            {
                case (Gesture.GestureType.TYPECIRCLE):
                    {
                        CircleGesture circle = new CircleGesture(gesture);
                        string clockwiseness;
                        float turns = circle.Progress;
                        GameObject Rueda = GameObject.Find("Rueda");
                        if (circle.Pointable.Direction.AngleTo(circle.Normal)
<= Mathf.PI / 2)
                        {
                            clockwiseness = "clockwise";
                            Rueda.transform.Rotate(0, -5 * turns, 0);
                        }
                        else
```

```
                        {
                            clockwiseness = "counterclockwise";
                            Rueda.transform.Rotate(0, 5 * turns, 0);
                        }
                        print(clockwiseness);
                        break;
                    }
                }

            }
        }

}
```

Codi 5: GestureRecognition_Circle, reconeix el gest Circle

6.4.4. Escena 4: Point and shoot

- **Objectiu**

En aquesta escena, es tracta bàsicament de disparar balins mitjançant un gest, anomenat ScreenTap, el qual es reconeix quan la punta del dit empeny cap endavant i després salta cap enrere fins aproximadament la posició original, com si fes la intenció de tocar la pantalla. El dit ha de fer una petita pausa abans de començar el gest, no es pot fer molt seguidament perquè sinó li costa de detectar al Leap Motion.

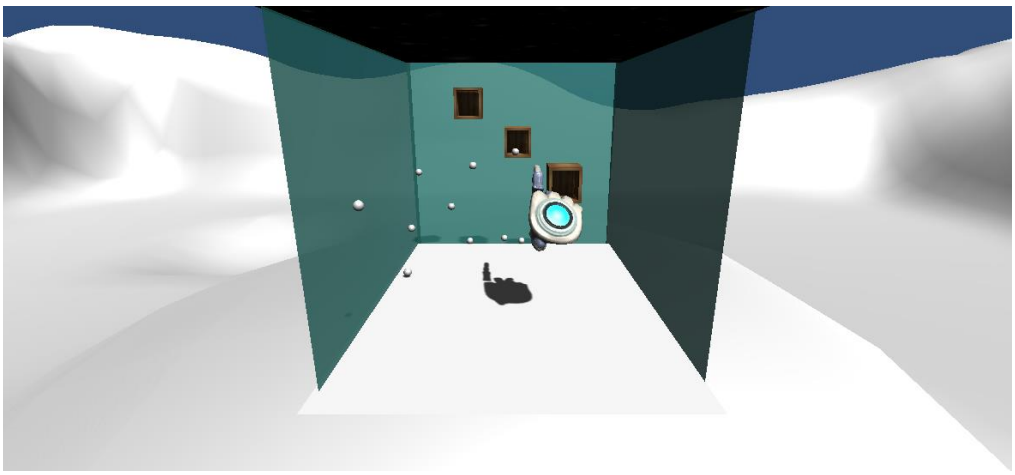


Fig. 6.12: Visualització de l'escena 4 en mode play

- **Procediment**

Per la creació d'aquesta escena, s'ha tornat a crear un escenari tancat semblant al de les altres escenes, que és on estaran ubicades les mans quan s'executi l'aplicació, i de fons s'ha utilitzat el *GameObject > 3D Object > Terrain* per crear aquests relleus com si es volguessin simular unes muntanyes nevades. S'han incorporat 3 caixes de fusta a la paret

del fons. S'ha creat també un prefab Sphere, el qual es farà servir cada cop que es dispari una bala. Aquest fet s'explicarà conjuntament amb l'script.

- **Scripts**

- *GameController*: *GestureRecognition_TYPESCREENTAP*

En aquest script es programa bàsicament el reconeixement del gest Screen Tap. En l'Start() es crida el Controller (tal i com a les altres escenes) i s'habilita la classe Gestures, i concretament el tipus de gest *TYPESCREENTAP*. En l'Update() es crida la classe Frame, que guarda les instàncies de l'escena gravades pel controlador Leap Motion. I per tant, en aquestes instàncies es detectarà el gest programat. Després, s'accedeix al dit *índex* i concretament a l'os *bone3,* per que les bales es dispari des d'allà, cada cop que es fa el gest.

Fig. 6.13: Representació del gest ScreenTap

```
using UnityEngine;
using System.Collections;
using Leap;

public class GestureRecognition_TYPESCREENTAP : MonoBehaviour
{
    Leap.Controller controller;
    public GameObject obj;
    public GameObject shot;
    public Transform shotSpawn;
    public float fireRate;
    private float nextFire;

    void Start()
    {
        controller = new Controller();
        controller.EnableGesture(Gesture.GestureType.TYPESCREENTAP);
    }

    void Update()
    {
```

```
        Frame frame = controller.Frame();

        foreach (Gesture gesture in frame.Gestures())
        {

            switch (gesture.Type)
            {

                case (Gesture.GestureType.TYPESCREENTAP):
                    {
                        ScreenTapGesture screentap = new
ScreenTapGesture(gesture);
                        Vector pokeDirection = screentap.Direction;
                        Vector pokeLocation = screentap.Position;
                        GameObject go =
GameObject.Find("PinchingHand(Clone)/index/bone3");

                        print("type screentap");
                        if (Time.time > nextFire)
                        {
                            nextFire = Time.time + fireRate;
                            Instantiate(shot, go.transform.position,
Quaternion.identity);
                            //  GetComponent<AudioSource>().Play();

                        }
                        break;
                    }
            }

        }
    }
}
```

Codi 6: GestureRecognition_TypeScreenTap, reconeix el gest ScreenTap

- *Sphere : Done_Mover*

Aquest script està adjuntat al gameObject sphere. L'objectiu és que es dispari una bala cada cop que es realitza el gest ScreenTap. Per a això, s'ha hagut de definir una variable pública, *speed*, que és la velocitat a la qual sortirà la bola. Llavors, també s'ha de definir una força, que s'aplica contínuament al llarg del vector de la força. El mode ModeForce, permet que el tipus de força es canviï a una acceleració, impuls o canvi de velocitat. La força només pot ser aplicada a un cos rígid actiu, i si un GameObject està inactiu, el AddForce no té cap tipus d'efecte.

```
using UnityEngine;
using System.Collections;
```

```
public class Done_Mover : MonoBehaviour
{
    public float speed;

    void Start()
    {
        GetComponent<Rigidbody>().AddForce( transform.forward * speed );
    }

}
```

Codi 7: Done_Mover, script que serveix per disparar les bales

7. Planificació temporal i pressupost

7.1. Fases de desenvolupament

En aquest apartat, es desglossaran les hores dedicades a cada fase del projecte des del seu inici fins al final. La gràfica inclou les següents fases:

1. **Aprenentatge** (120h): que correspon a les hores invertides en l'aprenentatge de la API de Unity 3D i en la API del Leap Motion.
2. **Programació** (60h): correspon a les hores dedicades en la creació dels scripts de la App.
3. **Investigació** (64h): correspon a la recerca d'informació sobre l'estat de l'art de les aplicacions i les noves tecnologies relacionades amb la realitat virtual.
4. **Aplicació** (75h): correspon a les hores de disseny de les escenes.
5. **Memòria** (110h): correspon a les hores invertides en la redacció de la memòria del treball.
6. **Reunions** (8h): reunions amb el tutor del projecte.

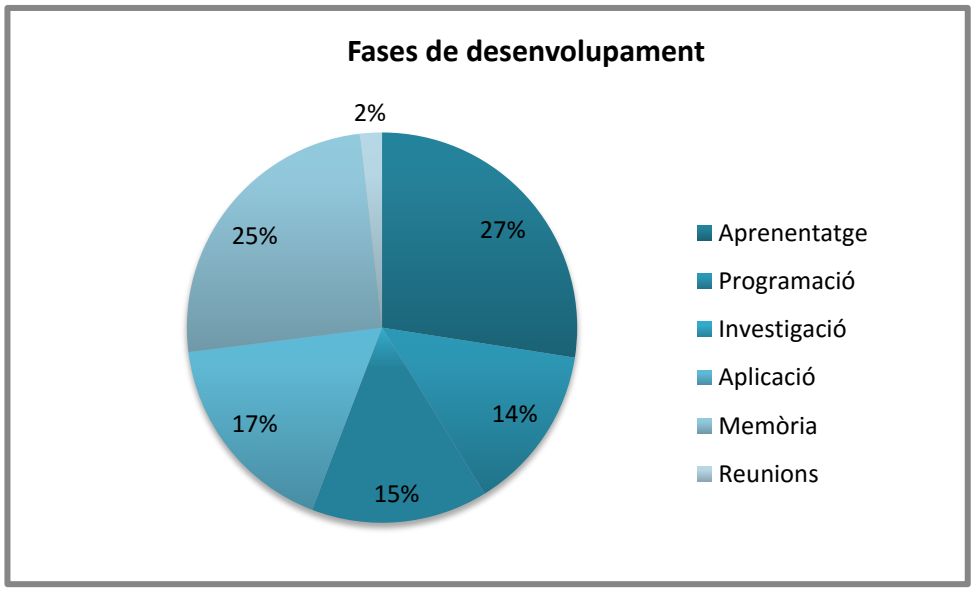


Fig. 7.1: Fases de desenvolupament del projecte

Per tant, nombre total d'hores que s'han dedicat per la realització d'aquest projecte són 437h.

7.2. Pressupost

Per a la realització d'aquest projecte s'ha necessitat un matemàtic o enginyer (ja sigui industrial, informàtic, telecomunicacions, etc.) per al desenvolupament del software, programació i creació de l'aplicació.

S'estima un cost de 15€ per hora de treball de l'enginyer.

Respecte al Software necessari per la realització del projecte, es necessiten els següents programes:

- Programa base per la creació de l'aplicació: *Unity 3D v.5.4.0* versió personal (gratuït).
- Entorn de desenvolupament integrat (IDE): Microsoft Visual Studio (gratuït).
- Paquet de Microsoft Office, que inclou Word, Power Point i Excel.

Respecte al Hardware necessari, es necessiten els següents aparells:

- Ordinador de sobretaula amb el sistema operatiu Windows 10.
- Dispositiu Leap Motion Controller.

Els costos del projecte estan descrits a la taula següent:

Concepte	Cost unitari	Quantitat	Cost [€]
Enginyer	20,00 €/h	437 h	8.740,00
Subtotal			**8.740,00**
Programa Unity 3D v5.4.0	0,00 €/u	1 u	0,00
Microsoft Visual Studio	0,00 €/u	1 u	0,00
Paquet de Microsoft Office		1 u	479,00
Subtotal			**479,00**
Ordenador de sobretaula			800,00
Dispositiu Leap Motion			69,99
Consum d'electricitat			70,00
Subtotal			**939,99**
Total			**10.158,99**

8. Impacte medi ambiental

Realitzar un estudi d'impacte ambiental serveix per recollir avaluacions de com afecta el projecte en diferents aspectes rellevants de caire social o físic i utilitzar aquesta informació per solucionar els problemes abans de prendre decisions definitives.

Generalment, s'associa un impacte ambiental a un canvi advers. Tanmateix, el mot impacte es refereix a tot allò que provoca un canvi en el medi. Aquest treball s'ha basat en el disseny d'una aplicació utilitzant un programa de disseny 3D i un sensor que s'ha connectat a l'ordenador, per tant analitzem quins han estat els aspectes negatius i els positius de la realització del projecte:

- **Aspectes negatius**

Els aspectes negatius de l'impacte sobre l'entorn són gairebé escassos. Un dels aspectes negatius que es podria considerar és el consum energètic que deriva de l'ús de l'ordenador per al disseny de l'aplicació i de l'ús del dispositiu Leap Motion connectat a ell. No obstant, si tenim en compte els materials usats per a la seva construcció, metalls, plàstics, minerals, etc. han estat extrets del medi ambient i per tant aquesta activitat sí que li afecta. Llavors, també s'hauria de considerar el transport dels components fins les fàbriques on s'acoblen els equips, que generen contaminació per la crema de combustibles per als vehicles. Un cop s'ha acoblat l'equip, s'exporta a altres països, i per tant apareix novament la contaminació deguda al transport.

D'altra banda, la vida útil d'un ordenador es troba entre 4-8 anys (depenent de l'ordinador), i al cap dels anys es substitueix. Llavors, els materials si no es reciclen, passen a ser residus sòlids als abocadors, afectant novament al medi ambient.

- **Aspectes positius**

Els aspectes positius i l'impacte positiu que té aquest treball és molt més gran que el que pugui afectar negativament. Ha causat un impacte positiu per diverses raons. Ha servit per demostrar que enginyers industrials són capaços de dissenyar i programar una aplicació relacionada amb el camp de la realitat virtual, que no han estudiat ni tractat mai durant la carrera. A més, el treball servirà per continuar desenvolupant aplicacions més complexes utilitzant el dispositiu Leap Motion, i futurs treballs de final de grau realitzats al Centre de Realitat Virtual, podran aprofitar-lo com a base d'inici.

Per tant, es pot dir que té un impacte més bé social i formatiu positiu que no negatiu de cara al medi ambient.

Conclusions

El repte d'aquest projecte ha estat investigar les noves tecnologies d'interacció persona-computador, amb la funció de traslladar a l'usuari a una nova experiència propera a la realitat. Es va iniciar el treball sense tenir gairebé coneixement sobre el món de la realitat virtual i les tecnologies vinculades a ella, i s'ha finalitzat, havent desenvolupat una aplicació força completa i complexa a la vegada, utilitzant aquestes tecnologies.

La primera conclusió que se'n pot extreure és que tant jo com els meus companys que hem realitzat diferents treballs però tots relacionats amb la realitat virtual (sistema ideat pel CRV de la facultat de Matemàtiques de Barcelona), hem estat capaços desenvolupar un projecte amb aquestes característiques, una base ben informada i documentada tècnicament que probablement serveixi per futurs estudis o treballs de fi de grau, amb la finalitat de que segueixin amb la mateixa línia d'investigació, realitzant projectes encara més complexos.

S'ha pogut crear una aplicació en la qual l'usuari és capaç d'interactuar amb l'escena mitjançant la representació de les seves mans. Gràcies a això, l'usuari pot moure objectes de dins l'escena i col·locar-los on desitgi; i també pot gesticular amb les mans o els dits i fer que es produeixi una certa acció. S'ha estat capaç de controlar les API's tant del Unity 3D com la del Leap Motion, no diríem que a nivell expert però sí lo suficient com per programar una aplicació d'aquesta envergadura. La realitat virtual, avui en dia, és un sistema molt nou que té una versatilitat molt gran i cada cop són més els camps on l'ús d'aquesta podria afavorir la interacció entre persona i màquina. No estem parlant només del món industrial, sinó que també ja s'està introduint en altres vessants, com en aplicacions per a la medicina.

.

Agraïments

En primer lloc, m'agradaria agrair al tutor del meu TFG, el professor Toni Susin, l'oportunitat de realitzar aquest treball sota la seva directriu, juntament amb tres alumnes més que han realitzat treballs enfocats en el mateix àmbit de la Realitat Virtual. Agrair la seva disposició i ajuda brindada durant la realització del projecte, especialment quan vaig patir la lesió al genoll a principis de Novembre. I també agrair la possibilitat d'haver treballat al Centre de Realitat Virtual de la facultat de Matemàtiques, i tenir l'oportunitat de descobrir un nou àmbit tan apassionant com és la realitat virtual.

En segon lloc, agrair a Jordi Moyés tota la seva ajuda, qui s'ha brindat a la nostra disposició sempre que ho hem necessitat, tant per al desenvolupament del projecte com per la facilitat d'accés a les sales del CRV sempre que hem necessitat anar-hi.

Finalment, donar un agraïment especial al meu tiet Ignasi Iriondo, per tot el suport que m'ha donat durant la realització del projecte i en els moments que més ho he necessitat.

Bibliografia

Referències bibliogràfiques

[1] «Unity Technologies. Lloc web de Unity 3D,» [En línia]. Available: https://unity3d.com/es.

[2] «Web Oficial de la API de Leap Motion developers,» [En línia]. Available: https://developer.leapmotion.com/documentation/v2/unity/index.html.

[3] «Web oficial de Leap Motion,» [En línia]. Available: https://www.leapmotion.com/. [Últim accés: Gener 2017].

[4] D. Zax, «Review sobre Leap Motion (última edició Maig 2012). Títol: Leap 3D Out-Kinects Kinect,» [En línia]. Available: https://www.technologyreview.com/s/427981/leap-3d-out-kinects-kinect-video/.

[5] «Manual de Unity3D,» [En línia]. Available: http://docs.unity3d.com/es/current/Manual/.

[6] «Què és una Interficie de programació d'aplicacions API,» [En línia]. Available: http://www.ticbeat.com/tecnologias/que-es-una-api-para-que-sirve/. [Últim accés: 2 Gener 2017].

[7] «Developer guide Unity 3D - set up a project,» [En línia]. Available: https://developer.leapmotion.com/documentation/v2/unity/devguide/Project_Setup.html.

[8] «Leap Motion documentation v2 - Gestures,» [En línia]. Available: https://developer.leapmotion.com/documentation/v2/csharp/api/.

[9] «The 3D Models - 3d models for free,» [En línia]. Available: http://tf3dm.com/.

Bibliografia complementària

- Weichert, F; Bachmann, D; Rudak, B; Fisseler, D. *Analysis of the Accuracy and Robustness of the Leap Motion Controller*. (2013). Última visita 07-01-2017

- YOUTUBE. "*An inside view of the Leap Motion controller*". Última visita 25-10-2016

- Terdiman, Daniel. *"Leap Motion: 3D hands-free motion control, unbound"*. *Cutting Edge*. cnet. Última visita 21-01-2017.

- Lang, Ben. "Orion is Leap Motion's Overhauled VR Hand Tracking Engine". *Road to VR*. Última visita 18-11-2017.

- Hart, Brian. "'Diplopia' Becomes 'Vivid Vision', Trials Get Underway - Interview with CEO James Blaha - Road to VR". *Road to VR*. Última visita 22-12-2016

- Lee, Kevin (2013-12-06). "Leap Motion bounding to more HP desktops, all-in-ones with new keyboard". *Techradar*. Última visita 25-10-2016.

- Scott Hayden. "Winners of Leap Motion '3D Jam' Game Jam Contest Announced - Road to VR". *Road to VR.* Última visita 22-12-2016

- *"Leap Motion seals HP deal to embed gesture control technology"* Última visita 16-12-2016

- Wing, Anthony "Leap Motion Hand Tracking Gets More Realistic To Solve Real World Problems". *Forbes*. Ultima visita 12-12-2016

Annexos

Annex 1 : Codi *MouseLook*

```
using UnityEngine;
using System.Collections;

[AddComponentMenu("Camera-Control/Mouse Look")]
public class MouseLook : MonoBehaviour {

	public enum RotationAxes { MouseXAndY = 0, MouseX = 1, MouseY = 2 }
	public RotationAxes axes = RotationAxes.MouseXAndY;
	public float sensitivityX = 15F;
	public float sensitivityY = 15F;

	public float minimumX = -360F;
	public float maximumX = 360F;

	public float minimumY = -60F;
	public float maximumY = 60F;

	float rotationY = 0F;

	void Update ()
	{
		if (axes == RotationAxes.MouseXAndY)
		{
			float rotationX = transform.localEulerAngles.y +
Input.GetAxis("Mouse X") * sensitivityX;

			rotationY += Input.GetAxis("Mouse Y") * sensitivityY;
			rotationY = Mathf.Clamp (rotationY, minimumY, maximumY);

			transform.localEulerAngles = new Vector3(-rotationY,
rotationX, 0);
		}
		else if (axes == RotationAxes.MouseX)
		{
			transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityX,
0);
		}
		else
		{
			rotationY += Input.GetAxis("Mouse Y") * sensitivityY;
			rotationY = Mathf.Clamp (rotationY, minimumY, maximumY);

			transform.localEulerAngles = new Vector3(-rotationY,
transform.localEulerAngles.y, 0);
		}
	}
```

```
	void Start ()
	{
		// Make the rigid body not change rotation
		if (GetComponent<Rigidbody>())
			GetComponent<Rigidbody>().freezeRotation = true;
	}

}
```

Annex 2 : Codi *CharacterMovement*

```
#pragma strict
var speed : float = 7; //player's movement speed
var gravity : float = 10; //amount of gravitational force applied to the player
private var controller : CharacterController; //player's CharacterController
component
private var moveDirection : Vector3 = Vector3.zero;

function Start () {
	controller = transform.GetComponent(CharacterController);
}

function Update () {
	//APPLY GRAVITY
	if(moveDirection.y > gravity * -1) {
		moveDirection.y -= gravity * Time.deltaTime;
	}
	controller.Move(moveDirection * Time.deltaTime);
	var left = transform.TransformDirection(Vector3.left);

	if(controller.isGrounded) {
		if(Input.GetKeyDown(KeyCode.Space)) {
			moveDirection.y = speed;
		}
		else if(Input.GetKey("w")) {
			if(Input.GetKey(KeyCode.LeftShift)) {
				controller.SimpleMove(transform.forward * speed * 2);
			}
			else {
				controller.SimpleMove(transform.forward * speed);
			}
		}
		else if(Input.GetKey("s")) {
			if(Input.GetKey(KeyCode.LeftShift)) {
				controller.SimpleMove(transform.forward * -speed *
2);
			}
			else {
				controller.SimpleMove(transform.forward * -speed);
			}
		}
		else if(Input.GetKey("a")) {
```

```
                    if(Input.GetKey(KeyCode.LeftShift)) {
                        controller.SimpleMove(left * speed * 2);
                    }
                    else {
                        controller.SimpleMove(left * speed);
                    }
                }
                else if(Input.GetKey("d")) {
                    if(Input.GetKey(KeyCode.LeftShift)) {
                        controller.SimpleMove(left * -speed * 2);
                    }
                    else {
                        controller.SimpleMove(left * -speed);
                    }
                }
        }
        else {
                if(Input.GetKey("w")) {
                    var relative : Vector3;
                    relative = transform.TransformDirection(0,0,1);
                    if(Input.GetKey(KeyCode.LeftShift)) {
                        controller.Move(relative * Time.deltaTime * speed *
2);
                    }
                    else {
                        controller.Move(relative * Time.deltaTime * speed);
                    }
                }
        }

}
```