TÍTULO: Diseño e implementación de un método para la recreación de gestos faciales a tiempo real con Kinect mediante el uso de blend shapes.
Design of a Kinect controlled real-time facial animation method using blend shapes

AUTOR: Alfredo Díaz García

TITULACIÓN: Grado en Ingeniería de Sonido e Imagen

TUTOR
Dr. Szirmay-Kalos László

UNIVERSIDAD: BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
CENTRO: Faculty of Electrical Engineering and Informatics
PAÍS: Hungría

Fecha de lectura: 20-07-2017

Calificación:

El Coordinador de Movilidad,
RESUMEN

Kinect es un dispositivo doméstico creado por Microsoft para ser usado como controlador en su consola de videojuegos. Pero debido a las posibilidades de uso que ofrece en otros campos, este dispositivo empezó a ser usado por los desarrolladores para sus propios proyectos. A la vista de esto, Microsoft decidió dejar a los usuarios usar todas las librerías propias de Kinect.

Desde entonces, Microsoft ha ido ampliando el número de funciones y posibilidades de las librerías. También crearon una nueva librería dedicada al reconocimiento facial, que recientemente mejoraron y publicaron como Kinect Face HD, esta librería mejora considerablemente el desempeño de Kinect en el reconocimiento facial.

La animación facial de un modelo 3D no es algo sencillo y rápido de hacer. La cara de un humano tiene 43 músculos. Para poder replicar fielmente las expresiones faciales, tenemos que intentar replicar el movimiento de todos estos músculos. El uso de blend shapes es una técnica muy usada para aumentar la velocidad a la que se crean las animaciones faciales. Esta técnica hace que les sea más fácil a los animadores crear estas expresiones únicamente mediante la modificación de unos pocos parámetros.

En este proyecto, vamos a usar estas tecnologías juntas. Crearemos un modelo 3D de una cabeza humana. Tras esto, crearemos los blend shapes que recrearán diferentes expresiones. Una vez hecho esto, usaremos Kinect para capturar los datos del usuario. Después, tendremos que estudiar la forma de usar la librería Face HD, dado que está muy mal documentada actualmente, y determinar cómo usar estos datos obtenidos para saber la expresión facial del usuario.

Una vez que tengamos los datos del usuario analizados y los diferentes blend shapes, se debe buscar la manera de transformar estos datos en los parámetros de los blend shapes.

Todo este proceso se va a realizar en tiempo real usando Unity 5.4, un motor de videojuego muy usado, para hacer trabajar todas las partes juntas. Elegimos este motor gráfico debido a que gracias a recientes actualizaciones, podremos usar las librerías de Kinect directamente en el.
Abstract

Kinect is a home device created by Microsoft for its video game console to serve as a controller. But due to the great possibilities it brings in other fields, this device was quickly used by the programmers for their own purposes and projects. After this, Microsoft decided to allow the users to use the Kinect library.

Since then, Microsoft has increased the number of functions and possibilities of the libraries. They also have created a new library dedicated to face tracking, that they recently evolved into another library called Face HD, that hugely improves the performance in face tracking.

The facial animation of a 3D model is not an easy task. The human face has 43 muscles in total. In order to replicate the facial expressions, we have to replicate the movement of all the muscles. Blend shapes is a widely-used technique to increase the speed in facial animation. It makes easier for animators to create realistic facial expressions with just the adjustment of a few parameters.

In this project, we are using these technologies together, we will create a 3D model of a human head. Then, we will create the different blend shapes recreating different expressions. After this, we will use the Kinect to capture the data of the user. Then we will study the Face HD library, that is not well documented, and determine how to use its output data in order to know what facial expression the user actually have.

Once we have the data of the user and the different blend shapes, we are going to find a way to analyse this data and transform it into blend shape weights.

All the process is going to be real-time using Unity, a well-known game engine, to make it work all together. In this engine, we can use Kinect libraries due to fresh updates.
DESIGN OF A KINECT CONTROLLED REAL-TIME FACIAL ANIMATION METHOD USING BLEND SHAPES

Autor: Alfredo Díaz García
Supervisor: Dr. Szirmay-Kalos László & Tamás Umenhoffer

July 2017
CONTENIDO

1 INTRODUCTION .................................................................................................................. 6
2 OBJECTIVES ........................................................................................................................ 7
3 RELATED WORK .................................................................................................................. 8
4 HARDWARE ........................................................................................................................ 10
  4.1 Kinect V2 ......................................................................................................................... 10
    4.1.1 Parts of the Kinect V2 .............................................................................................. 11
    4.1.2 Specifications ............................................................................................................ 11
    4.1.3 RGB camera ............................................................................................................. 12
    4.1.4 Depth Camera ........................................................................................................... 14
  4.2 Computer ....................................................................................................................... 15
5 SOFTWARE .......................................................................................................................... 18
  5.1 Autodesk Maya ................................................................................................................ 18
    5.1.1 Modelling Features ................................................................................................. 19
    5.1.2 Textures and Materials ............................................................................................ 19
    5.1.3 Animation Features ................................................................................................. 19
  5.2 Unity 5.4 ........................................................................................................................ 20
  5.3 Kinect SDK ..................................................................................................................... 21
    5.3.1 API .......................................................................................................................... 22
    5.3.2 Programming Kinect based applications ................................................................. 22
    5.3.3 Kinect Face SDK ..................................................................................................... 24
    5.3.4 Kinect Face HD ....................................................................................................... 28
6 SOLUTION ............................................................................................................................. 33
  6.1 INTRODUCTION .............................................................................................................. 33
  6.2 PHASE 1: 3D MODELLING ............................................................................................. 34
    6.2.1 Low Polygon modelling ............................................................................................ 35
    6.2.2 High polygon modelling ......................................................................................... 36
    6.2.3 Blend Shapes .......................................................................................................... 36
  6.3 PHASE 2: Kinect data analysis ......................................................................................... 38
  6.4 PHASE 3: Unity C# coding ............................................................................................. 44
To my fathers for letting me starting my degree at a late time
LIST OF FIGURES

FIGURE 1: PARTS OF THE KINECT ........................................................................................................ 11
FIGURE 2: THREE SENSORS CAMERAS ............................................................................................ 12
FIGURE 3: BAYER FILTER .................................................................................................................. 13
FIGURE 4: CCD AND CMOS ............................................................................................................... 14
FIGURE 5: TIME-OF-FLIGHT TIMING DIAGRAM ............................................................................ 15
FIGURE 6: COMPUTER USED IN THIS PROJECT ............................................................................ 17
FIGURE 7: AUTODESK MAYA EDITOR ............................................................................................... 18
FIGURE 8: UNITY ENVIRONMENT ..................................................................................................... 20
FIGURE 9: ACQUIRE AND OPEN THE SENSOR .............................................................................. 23
FIGURE 10: INITIALIZATION OF THE SOURCES ............................................................................ 24
FIGURE 11: ACCESS THE FRAMES .................................................................................................. 24
FIGURE 12: HEAD ANGLES ............................................................................................................... 26
FIGURE 13: AU’S IN BASIC FACE TRACKING ................................................................................... 28
FIGURE 14: FLOW CHART OF THE TRACKING SYSTEM (IMAGE SOURCE [1]) ............................... 32
FIGURE 15: LOW, MID AND HIGH POLYGON MODELS ................................................................. 35
FIGURE 16: OUR MODEL AND THE DOWNLOADED MODEL .......................................................... 36
FIGURE 17: EXAMPLE OF A MODEL WITH DIFFERENT BLEND SHAPES .................................. 37
FIGURE 18: DIFFERENCE BETWEEN RESOLUTIONS ...................................................................... 39
FIGURE 19: BIT DEPTH DIFFERENCE .............................................................................................. 40
FIGURE 20: FLOW CHART OF BLEND SHAPE WEIGHTS ............................................................... 44
FIGURE 21: KINECT LIBRARIES DECLARATION IN UNITY ......................................................... 45
FIGURE 22: Unity object inspector.................................................................46
FIGURE 23: Camera input of Kinect in Unity......................................................47
FIGURE 24: Example of OnApplicationQuit () function..................................48
FIGURE 25: Using values of other scripts in Unity..........................................49
FIGURE 26: Flowchart of the Face HD code......................................................50
FIGURE 27: Blend shapes created for the test....................................................51
FIGURE 28: Neutral face before and after calibration.......................................52
FIGURE 29: Smiling and surprise.................................................................52
FIGURE 30: Wink and eyebrows up..............................................................53
FIGURE 31: Cheek puff and mouth open.......................................................53
1 Introduction

Every day that passes, the graphics technologies are more widely used. Something that years ago was only used and improved for videogames, is now being used to make movies, create pieces of machinery or even create human organs. Also, because of the exponential grow of the computer technology, computer graphics have evolved due to the great capacities of this new computers.

With the emergence of devices like Kinect, tendency of creating games that uses the users body as a control also appears. Or even that tries to recreate the movement of the user with the movement of an avatar. This is how it started, but now it is common to see that the avatar animations, are made by using this technology and more realistic animations are created by capturing the user’s movement. This also applies to facial expressions, but facial expressions are much more complicated to recreate than the body movement.

Usually when we capture the movement of the body, we can easily do it by capturing the bones and articulations since it is quite easy to do. But with the face we cannot do that, in a human face we have 43 muscles and they are in charge of creating the facial expressions, so we have to look at these little changes in order to create a good animation.

In this project, we will learn about the facial animation and we will do an application that tries to recreate the facial animation in a 3D avatar model in real time, with the use of Kinect and with no facial markers.
2 OBJECTIVES

Engineer comes from the Latin words *ingeniare* (to contrive) and *ingenium* (cleverness). In the telecommunication degree, we acquire a lot of knowledge of this huge field that involves physics, electronics and informatics. But when we get out to the world as Engineers we are going to face the reality, that this knowledge is only the ground bases and that we have to learn by ourselves a lot of new skills. Because of this I decided to take for this project a field in which I have very little experience in order to also prepare myself to the outside world.

So, this project is not only important to create a good application, the objective is to create an application with new skills that we acquire while we do the project. Because of this, in this project one of the main objectives is to acquire skills that include:

- 3D modelling
- C# programming
- Kinect libraries usage
- Coding for Unity

Beside this, the final goal is to create an application that with the use of the Kinect libraries and sensor data and with blend shapes, recreates a facial expression of the user on a 3D model in real-time.
3 Related Work

Usually when we think about animation, and in this case facial animation, we can think about huge movies like Avatar for example. In these movies in some scenes, some actors are replaced by animated virtual characters. Usually, to create the animation of these 3D models, they use a tracking device that captures the movement of the real actor and translate it directly to the model. If we talk about the face, they capture some marked points on the face of the actor and link them to the 3D model face so it moves with the actor. When digitally replicating a real-world actor, the 3D model and the source model (the actor) have the exact same shape. Usually they only need to adapt the mesh of the model to make the correct vertices coincide with the markers in the actor’s face.

If we try to do the facial animation when the source and the model shape is completely different, we cannot just adapt the mesh to make the marker vertex positions coincide with the real marker positions, because due to the different shapes, the movement of those vertices have a different length or direction. So, in these cases we need to make a retargeting of the data that we capture in the input.

One possible option in these cases is to save the motion vectors of the movement of the markers in the input and then retarget these motion vectors to adapt to the new shape and its new curvature. In order to make this method work well, we need to track a lot of markers in the input so we have a lot of motion vectors to transform to make the expression more realistic. We also need a high processing power to make the operations work in real-time. We have to make thousands of operations with the vectors 30 times per second if for example we capture 30 Frames per Second.

Another option, the one that we are following is the use of parameterization of the face, for example creating parameters like mouth open or smiling. This method was
used at the beginnings of facial animation because it was easy to use with blend shapes, since you can create a blend shape for each parameter and use the parameters to directly control blend shape weights. The bad thing of this method was that the parameters that you could get were too basic, it was too difficult at the time to get more detailed information of the face movement in order to create more parameters. But with the advances in the technologies and the advent of devices like Kinect V2, now we have access to a huge quantity of information about the user in order to capture a higher quantity of parameters for our animations.
4 Hardware

4.1 Kinect V2

This device would be the main tool of this project, because we are using the information Kinect V2 provide us in order to achieve our goals.

This device is the second version of Kinect, that was launched for the videogame console Xbox 360. Kinect V2 was released in 2013 by Microsoft with the videogame console Xbox One. Kinect allows the user to interact with the videogame console without any controller.

Due to the capabilities of Kinect to recognize the movement and the gestures of the player, the developers started to try to get the raw information of Kinect, even though they shouldn’t have access to it. With this information, they made computer applications and started to develop their own games. This is why Microsoft launched an API and a plugin with the Kinect V2 that allows the user to use all the capabilities of the device and program any application they can imagine.
Figure 1: Parts of the Kinect

4.1.1 Parts of the Kinect V2
Kinect V2 have the following parts (Figure 1):

- RGB camera
- IR Depth Sensor
- Microphone

4.1.2 Specifications
- Colour camera: 1920 x 1080 @ 30 fps
- Depth Camera: 512 x 424
- Max Depth Distance: 4.5 m
- Min Depth Distance: 50 cm
- Horizontal field of view: 70 degrees
- Vertical field of view: 60 degrees
- Tilt Motor: no
- Skeleton Joints Defined: 26 joints
Full Skeletons Tracked: 6
USB Standard: 3.0
Supported OS: Win 8

4.1.3 RGB camera

Kinect uses a common high definition camera to capture the image. This camera has a resolution of 1920x1080 @ 30 fps. In order to capture the image, the light goes through a lens, then it is directed to a filter that separates colors. There are different types of camera, on the one hand we have the cameras with three sensors (Figure 2). In this type, the light is redirected by a prism to three different filters that captures only one of the three primary colors (red, green and blue). On the other hand, we have the cameras with one filter. This filter has different cells for each color, the name of this sensor is a Bayer sensor (Figure 3).

![RGB camera diagram](image)

Figure 2: three sensors cameras
Now the camera has the intensity of light that relates to each color but we need to capture this information. For this task we use intensity sensors, these sensors have many cells, the number of cells is what results in the resolution of the camera.

There are two types of sensors according with the technology used. CCD (Charge Couple Device) and CMOS (Complementary Metal Oxide Semiconductor). Both sensors are divided in cells. These cells capture the intensity of the light and transform it in voltage, the difference lies on the way the sensor gives the camera the information. In the CCD, the information of the voltage of every pixel is transferred to different shift registers (Figure 4). This information passes through a ADC in order to digitalize the value of all the cells at the same time. In the CMOS sensor, the value of the voltage is digitalized in every pixel and inside the sensor (Figure 4).
4.1.4 Depth Camera

The depth sensor consists of two parts an IR projector and an IR sensor which is a CMOS sensor. The two parts are placed in the same line in the Kinect, at a distance of 75mm. This colocation is to facilitate the depth calculations to the Kinect. These calculations are based in a stereo triangulation.

Stereo triangulation is based in the human eyes, we can sense depth because of the disparity \( d \) of a point in our left and right retinal images.

\[ d = x_{Left} - x_{Right} \]

But Kinect don't use exactly stereo triangulation, it uses a technology that is called Time-of-Flight. In this technology, you don't use two different cameras or IR sensors to get the stereo image.

Time-of-Flight is a method of measuring the distance between a sensor and a target surface. Is based on the difference of time between the emission of a signal and its return to the sensor. In Kinect V2 we use pulses of laser light as our emission signal.
What Kinects V2 does is taking a pixel and divide it in half. Both pixels are turned on and off quickly but with 180° phase delay. So, when one half is on, the other one is off. When the half is off it is rejecting the photons of the laser light, when it is on, it is absorbing the photons of the laser. The laser is also being pulsed in phase with the first half. In the Figure 5 we can see a timing diagram of the Time-of-Flight method.

![Diagram of Time-of-Flight method](image)

**Figure 5: Time-of-flight timing diagram**

The depth sensor is not 100 percent accurate, it has an error of 10 cm in distances of more than 4m and 2cm in distances of 2.5m or less.

### 4.2 Computer

Nowadays the computer has become one of the most important parts in every research or project. This project uses really demanding computer software. So having a powerful computer is really necessary for us.
In our case, we are working with Kinect, this device gives us huge amount of information. Some of this information is processed and light but what this project is going to use is almost all raw data. This means that the calculations of the data are made in the computer by the Kinect API. Since we are going to be working in real time, we will need a very powerful processor in order to provide sufficient performance.

This project uses 3D models and very powerful modelling and game engine programs, such as Maya and Unity. This is why we need good graphic card in order to visualize the data.

It is also important to have a big amount of RAM memory. All the raw data that the Kinect gives us every frame (30 frames per second) is going to be stored in this memory. So it is important that the computer don’t run out of it or the application will crash.

The computer used in this project is a MSI PE60 6QD (Figure 6). The specifications are:

- Processor: Intel I5-6300 HQ (quad-core)
- Graphic Card: NVidia GTX 950m
- Ram: 8Gb DDR4
Figure 6: Computer used in this project
5 SOFTWARE

5.1 Autodesk Maya

In the market, there are lots of modelling tools and programs (3Dmax, blender or Maya for example). In this project, we are going to use Autodesk Maya as our modelling tool. Maya allows us to use blend shapes in a very easy way. Also, it is the most commonly used modelling software for game and movie animations. In this brief section, we will describe the principal benefits and options of Maya.

Figure 7: Autodesk Maya editor
5.1.1 Modelling Features

There are four different fields in modelling: polygon, sub-division, NURBS and sculpting. Maya allows us to use any of these techniques. Is a very powerful software for expert modellers but also allows beginners modellers to start with it. We will use the polygon modelling technique, as it is the most widely used technique for interactive applications. Maya have a huge number of polygon modelling tools.

5.1.2 Textures and Materials

This is the strongest toolset of Maya, it has great texturing features and even better material and physics to make your model look more realistic.

Maya supports UV texture maps and has its own editor. We also can paint the map externally and import it to Maya again without any trouble. The result is an image file that covers the model as a skin. We can also give fine geometric detail to the model with a bump map.

Maya has a huge number of material simulation tools. There is no other 3D modelling software in the market that can compare when it comes to simulating fluids, cloth, skin, hair, fur and muscle. Maya also supports particles, rigid body dynamics and soft body dynamics.

5.1.3 Animation Features

Maya allow us to use IK (Inverse kinematics) and FK (forward kinematics). Also let us switch easily between the two of them. These are different types of skeletal animation controls. With this control, we would be able to create realistic motion
patterns that will be useful in the animation phase. The important thing for us is that Maya features blending which is ideal for animating facial expressions.

Maya also supports scripted animation, keyframe animation and procedural animation. We have a lot of options, we can animate in separate layers, use multiple audio tracks in our animations or even render in stereoscopic 3D.

5.2 Unity 5.4

Unity is a game engine that allows us to make applications or games for different platforms at the same time. It is an application where we can put together the parts of a game with a graphical preview and it's also comes with a code development enviroment (MonoDevelop). Unity is used when you have already created all the graphic content of the game, it unifies all of these content together, with the sound and animations and by coding the actuators generate a playable application.

![Figure 8: Unity environment](image-url)
Unity is the most popular Game engine for amateur developers, this is because it’s a free tool, it is easy to learn, and has great features. The Unity community is a very big and active community. We can see this by looking at the asset store, there are thousands of assets to buy and also for free.

Also, one of the good things about Unity is that the documentation is really complete. And as is it so widely used, there are lots of outside sources that can help us if we encounter trouble.

To finish, just say that for our project we are just using a 1 percent of what Unity really provide us. But it is a really powerful tool that as a developer we must have in our computer.

5.3 Kinect SDK

Kinect was released in 2010 as a complementary device for the Xbox 360. But early after it was launched people started to reverse engineer Kinect, in order to use the device data for their own applications. Seeing this, in 2011, Microsoft announced that they would release a non-commercial Kinect software development kit (SDK) for Windows on that year.

The SDK has been updated several times since then. With the release of Kinect V2 they also released a new SDK 2.0. This is the SDK that we will use, since it is the first one that also includes a library to program in unity environment.
5.3.1 API

The Application Programming Interfaces (API) is the more important part of the Kinect SDK. The SDK 2.0 gives us three different API sets to create Kinect applications. This sets are:

- Windows runtime API
- .NET API
- Native API

5.3.2 Programming Kinect based applications

In this section, we are going to explain briefly the data that you can get from Kinect. Also, we will explain how to access this data in a programming phase.

Kinect API works with Object Oriented Programming (OOP), such as java, C++ or C#. In this project, we will use C# because it is the programming language that best suites Unity. In OOP, we work with different classes that have their own methods.

5.3.2.1 Kinect Sensor Class

Windows supports one sensor; this class is the one that configures and gets access to the sensor data. First of all, you have to tell the computer that you want to get access to this sensor. After getting the sensor, you have to see if the sensor is open and activate it if it is not (Figure 9).
DESIGN OF A KINECT CONTROLLED REAL-TIME FACIAL ANIMATION METHOD USING BLEND SHAPES

```csharp
void Start () {
    _sensor = KinectSensor.GetDefault();
    if (_sensor != null) {
        if (!_sensor.IsOpen) {
            _sensor.Open();
        }
    }
}
```

**Figure 9: Acquire and open the sensor**

The sensor has also a few properties to see what is happening with it. These are the `open`, that tell us if the sensor is already open, and the `isAvailable` property, that tell us if a sensor is connected to the device.

5.3.2.2 Data Sources

The Kinect sensor have access to different sources:

- Colour
- Depth
- Body
- Infrared

These sources are controlled by three things (source type, Source Reader and Frame type). For accessing the data of these sources, we need to create these objects (Figure 10). Each of these objects have their own methods and information that you can find in [2].
Once we initialize the sources and readers we can have access to the data streams, we can acquire the captured frame of the reader (Figure 11). This frame contains all the data the reader can give us for the current frame.

```csharp
void Update () {
    _bodyReader.FrameArrived += BodyReader_FrameArrived;
    _faceReader.FrameArrived += FaceReader_FrameArrived;
}
```

**Figure 11: access the frames**

### 5.3.3 Kinect Face SDK

When 1.5 version of Kinect for Windows was released, Microsoft also released a new SDK for Kinect. This SDK was the Face Tracking Software Development Kit. This SDK allow us to create applications that can track human faces in real time.

The Kinect Face tracking analyzes the data of the Kinect camera and extracts the position of the head and some facial expressions in real time. We will explain in deep how Kinect manages to do this when we explain the Kinect Face HD, that is the evolution of this one, for now we will see the information that this SDK can give us.
5.3.3.1 Outputs

You can get the following data of the face tracking:

- Tracking status
- 2D points
- 3D head pose
- Animation Units

5.3.3.2 2D points

The basic face tracking can track 100 different 2D points; these points are returned in an array with X and Y coordinates. These coordinates refer to the image captured by the Kinect camera (1920 X 1080 in the Kinect V2). The face tracker also gives us the information about the position of the head (a box around it).

5.3.3.3 3D Head Pose

The X, Y and Z position of the user’s head are given to us in cartesian coordinates, being the Y up and Z the one that goes from the Kinect to the user.

The head pose is captured by three angles: pitch, roll and yaw (Figure 12). The angles are expressed in degrees from -180º to 180º.
5.3.3.4 Animation Units

The animation units give us information about the facial expression of the user. These animation units are going to be the base of this project. With the basic face tracking we have six AU’s. Each one has a value that goes from -1 to 1 we can see the six AU’s on Figure 13.
### Design of a Kinect Controlled Real-Time Facial Animation Method Using Blend Shapes

<table>
<thead>
<tr>
<th>AU Name and Value</th>
<th>Avatar Illustration</th>
<th>AU Value Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutral Face</td>
<td><img src="image1.png" alt="Neutral Face" /></td>
<td>(all AUs 0)</td>
</tr>
</tbody>
</table>
| AU0 – Upper Lip Raiser | ![AU0](image2.png) | 0 = neutral, covering teeth  
1 = showing teeth fully  
-1 = maximal possible pushed down lip |
| AU1 – Jaw Lowerer | ![AU1](image3.png) | 0 = closed  
1 = fully open  
-1 = closed, like 0 |
| AU2 – Lip Stretcher | ![AU2](image4.png) | 0 = neutral  
1 = fully stretched (joker’s smile)  
-0.5 = rounded (pout)  
-1 = fully rounded (kissing mouth) |
| AU3 – Brow Lowerer | ![AU3](image5.png) | 0 = neutral  
-1 = raised almost all the way  
+1 = fully lowered (to the limit of the eyes) |
Figure 13: AU’s in basic face tracking

5.3.4 Kinect Face HD

With the release of the new video game console Xbox one, they also released the new improved Kinect V2. The specifications of the device allowed to go a step further in the face tracking system. With the Kinect Face HD, we have access to over 1000 points of the user’s face and also, we can acquire 18 different animation units to analyse, this powerful face tracking library is the base of this project so we are going to briefly explain the base of the technology behind this library. We can find the full explanation of the research behind this tracking method in [1] of the bibliography.
5.3.4.1 Active Appearance Model (AAM)

The AAM allows us to reproduce images of surfaces that include non-rigid deformations. AAM is based in obtaining, during a training phase, a statistical model that describes the shape and appearance of the target object.

The AAM is a set of N characteristic points that create a mesh of the object they are represented by the vector \( s \).

\[
\mathbf{s} = (u_1, v_1, u_2, v_2, ..., u_N, v_N) \tag{1}
\]

Being \( u_i, v_i \), the coordinates of the vertex \( i \).

With the principal component analysis (PCA) over the training meshes we obtain a mesh \( s_0 \), and a subspace \( B_s = [s_1, ..., s_N] \) formed by \( n \) principal components with one dimension less than the training set.

We can get the instance of the model shape with the linear combination of the vectors of the base, \( B_s \) with the expression:

\[
\mathbf{s}(p) = s_0 + \sum_{i=1}^{n} p_i s_i \tag{2}
\]

We include the shape parameters in the model with the warp function \( W(x; p) \), this function transforms the interior points of a mesh (usually \( s_0 \)), were we define the appearance, to any mesh \( s(p) \) generated with (2):

\[
x' = W(x; p) \tag{3}
\]

With these elements, we obtain the model of linear appearance, that can generate an instance of an appearance with the lineal combination of the average and the base components, weighted with a set of parameters \( \lambda = (\lambda_1, \lambda_2, ..., \lambda_m) \).

\[
A(x; \lambda) = A_0(x) + \sum_{i=1}^{m} \lambda_i A_i(x) \tag{4}
\]
After this, we adjust the model. For this, we take an input image $I(x)$, and we find the set of parameters $p$ and $\lambda$ that minimise the quadratic error between the instance of the model generated with these parameters and the input image.

$$\sum_x [A_0(x) + \sum_{i=1}^m \lambda_i A_i(x) - I(W(x; p))]^2$$

This is the base of some face tracking systems and it works well in 2D tracking, we can also extend it to track 3D faces but is not really accurate enough to use it with confidence.

5.3.4.2 AAM face tracking with depth fitting

In later research, they added another constraint to the usual AAM 3D tracking that uses the depth data from a commodity RGBD camera (Kinect). This addition reduces significantly the 3D tracking errors.

We have already talked about the 2D AAM tracking in the last section. The 3D linear shape model is defined as:

$$s_{3D} = s_0 + \sum_{i=1}^L SU_i \bar{s}_i + \sum_{i=1}^M AU_i \bar{a}_i$$

$s_{3D} = 3D$ model in the camera space

$s_0 = mean \ 3D \ shape$

$s_i, \bar{a}_i = shape \ and \ animation \ deformation \ vectors$

$SU, AU = shape \ and \ animation \ deformation \ parameters$

What they did was extending these functions with a depth term. To do that, they formulate the depth term similarly to the energy function used in the Iterative Closest Point algorithm.
\[ E = w_{2D}E_{2D} + w_{2D3D}E_{2D3D} + w_{depth}E_{depth} + w_{temp}E_{temp} + w_{fseg}E_{fseg} + w_{reg}E_{reg} \]  

(7)

Where \( E_x \) are L2 terms and \( w_x \) are scalar weights. \( E_{depth} \) is the new term that they introduce to the equation. It minimized the Euclidean distances between face model vertices and the closest depth points in order to improve 3D fitting accuracy. We can see the processing flow of the tracking system on Figure 14.
DESIGN OF A KINECT CONTROLLED REAL-TIME FACIAL ANIMATION METHOD USING BLEND SHAPES

Figure 14: Flow chart of the tracking system (image source [1])
6 SOLUTION

6.1 INTRODUCTION

As we have seen in the previous chapters, there are many ways to make a facial animation. Also, there are ways of making a manual tracking of the face in order to get the data we want, and retarget it into the 3D model. But in this project, we want to use the capabilities of Kinect V2 by his own.

With the basic face tracking library of Kinect, you can already access to some basic but useful data. We can access the data of the position of the eyes, nose and mouth and you can identify some of the facial expressions with just some lines of code. But this information is not enough to replicate the facial expressions with high accuracy, so if we wanted to get more facial points or more information about the expressions, we needed to go through the raw data of the Kinect V2 sensor, and figure out a way to analyse this data to get the values we need.

With the release of the Kinect Face HD, the options for building a face tracking based application were hugely increased. Face HD is nowadays one of the most powerful face tracking libraries in the market. With it you can access the 3D position of over 1000 facial points in real-time.

But there is a problem, Kinect Face HD has a really bad documentation. It was made to provide advanced low-level functionality. This means that you have access to lot of raw data, but we are the ones that have to take this data and analyse what it means.
We are going to go through the phases of the project, this project was made for acquiring some new skills in areas that I don't dominate, and to create an application that make use of those skills. We can divide the project in three phases:

3D Modelling
Kinect Data analysis
Unity C# coding

6.2 Phase 1: 3D Modelling

3D Modelling is the process of creating a digital representation of any object or surface in a simulated 3D space. We make the modelling by manipulating the faces, edges or vertex of simple objects, such as planes or cubes. The limit of what you can make is your own ability and imagination.

Nowadays 3D Modelling is a great skill to have, this is because is used in a wide range of fields, such as Engineering, movie industry and of course games. Also in the medical field, they use 3D models of the organs and with the advance in 3D printing, they are starting to try to create printed functional organs based on these models.

For starting the modelling, we need a 3D modelling software, there are various in the market (blender, 3dmax, Maya) in this project we are using Autodesk Maya because of it blend shape tools that we will explain later. Maya allows us to create the basic objects in the 3D digital environment, and modifying these ones in our benefit.

In this project, the objective is to create a 3D head model with enough detail to have some facial expressions. This is a difficult task if you have never modelled because a human face has a lot of irregularities, and is not formed by simple polygons as a table would be for example. This is why we will learn how to model a head, but we will use a predesigned head for the rest of the project. We will use the skills we acquire to create the blend shapes of the predesigned model.
6.2.1 Low Polygon modelling

The models are made of a number of polygons that are connected between them in order to recreate a figure. The more polygons you have, the more realistic will the model be. In order to give form to the model we have to modify polygon edges and vertices.

What you see when you watch big animation movies or games, are very high polygon models, but you cannot really start modelling a high polygon object. If you do this it would be a gigantic task to move all the vertices of the thousands of polygons one by one. This is why we start the modelling with a very low polygon model (Figure 15) and then when we advance in the model, we add more details.

![Image of low, mid, and high polygon models]

**Figure 15: low, mid and High polygon models**

At the beginning, it is good to start with a cube with a resolution of 6, 4, 3 edges. In order to make faster modelling faster, we will only model half of the head, and later use the symmetry options to create the full head. The most difficult part of the model are the eyes, mouth, nose and ears. In a low polygon model, we have to create approximation of the form of this.
6.2.2 High polygon modelling

Once we have something like the low polygon model on Figure 15, we can start transforming the model into a high polygon model. This is where the experience on modelling is of more use. To create the high polygon model, we have to add new lines of definition to the low polygon model. We cannot add these lines randomly and you have to know exactly you want to achieve with every one you add.

Since the modelling was taking more time than expected and we were not getting very good results (Figure 16) we decided to use a free internet model for the next phases (Figure 16). We can see that there is a big difference, but you could get there by your own with more time.

![Figure 16: Our model and the downloaded model](image)

6.2.3 Blend Shapes

Morph target animation, per-vertex animation, shape interpolation or blend shapes are forms to call the same thing, we will use the name blend shape. This is a method of 3D animation that consist of creating a deformation of the model and saving the
positions of the new model. After this we use linear interpolation \([x]\) in order to create the intermediates positions of the vertex.

\[
y = y_a + (x - x_a) \left( \frac{y_b - y_a}{x_b - x_a} \right)
\]

[8]

To say it easy, we create a copy of the original model and with that base, we create a model with the facial expression we desire. Then we use linear interpolation to now the intermediate points and we can instantly create the model in the point of the expression we want. On Figure 17 we can see an example of different blend shapes of a model.

![Figure 17: Example of a model with different blend shapes](image)

One of the great tools that Autodesk Maya has, is the blend shapes tool. We only have to select the different expressions models with this tool and then select the original model. By doing this, Maya, will create a unique model with different sliders that controls the different interpolations of the blend shapes. This makes the creation of animations really fast and easy. This is because normally to do an animation, you have to modify the movement of the expression frame by frame in order to create
the animation. With the blend shapes done, we only have to change the parameter of the slider on the frames.

Another good thing about blend shapes nowadays is that you can easily export them to other program such as game engines like Unity. This is good for us since we are going to use Unity to create the Kinect application. Years ago, it was difficult to use blend shapes in Unity. Now you only have to import the correct package and you will have a new unity asset with the different blend shape parameters as object parameters.

Once we have the blend shapes of our model, we can go to the next phase.

6.3 Phase 2: Kinect data analysis

Kinect is a really powerful device that provides us several type of data about the input image. Since it is a camera created with the purpose of allowing us to control a game console with our body, we have a lot of tracking functionalities already created by the Microsoft developers. Of course, we can also get all the raw data from Kinect to analyse it by ourselves, but is better to use something already created to avoid losing too much time.

In our case, we don’t need the normal data that usually you get of Kinect. We are looking to get data about the face of the user. In order to make a good tracking of the body of a person with Kinect, we have to be at a minimum distance of 1.5-2 meters. This means that to get a good face tracking with a lot of different parameters, we would need a huge resolution of the camera.

An image in real life we can say that has an infinite number of pixels, so it’s impossible to get a digital photo with all of the information. In the digital photos we have a resolution, this resolution determines the number of points of information (Figure 18) that we have for example, with a HD camera (1920 x 1080) we have 2073600 pixels of information.
Beside the resolution, that is well-known by everybody, there is another important specification in a camera that influences the quality of the image, and therefore the quality of the information. This specification is the bit depth. This number specifies the resolution of the information in every pixel. It is the number of bits used to code the information of one pixel. For example, if we have a bit depth of 1 bit, we can only code the information as a 1 or a 0, so you only have two possibilities, black or white. The more bits you have, the more intermediate steps you get. This also gets the image size in the storage device bigger. On Figure 19 we can see a comparison between different grades of bit depth.

Figure 18: Difference between resolutions
The first Kinect, had a resolution of 1280x960. This made the detailed facial recognition really difficult. With the first Kinect, you could track the face and even get some data about it. You could know if the eyes or the mouth were opened or closed, or the head rotation. For this we used the Kinect Face Library.

At the first part of this phase, and in order to get used to the Kinect libraries, we used the Kinect face basic. We learned and implemented how to code for Kinect in order to get the information of the face. Since it has been a long time from the release of the Kinect Face, even though there is not much documentation of it in Microsoft Kinect websites, we were able to find a lot of information of how to use this libraries with outside sources. Once we had the results using the different parameters that Kinect Face gave, we determined that it was not accurate enough. This could be because since it was created to be used in Kinect V1 and Kinect V2, its prepared for a lower resolution and cannot effectively use the higher resolution of the Kinect V2.

Figure 19: Bit depth difference
We decided to try the last face tracking library, Face HD. This library was released for the Kinect V2 and makes use of the higher resolution and specifications of this device. But we encounter a problem while trying to work with the Kinect HD library. The documentation as in the face basic library, was insufficient to use the full potential of the library. Also, for the Face HD library, there was not good outside sources for our purposes. So we needed to found what all the raw information that we could get with it of the image. The Face HD give us access to over 1000 points of the face of the user in a 3D space, so we could recreate the face of the user, but we would need to do a retargeting in order to use it in another model and it would work better with a system of bones than with blend shapes. Finally we decided to use the Animation units.

The animation units are parameters that gives us information about some expressions in the face of the user. But even though they have a name that represents what expression it defines, some of them were difficult to know what they really told us. We made a practical research in order to get information, we code a program with all the animation units that sowed us the number for each one and we saw what triggered out the each one of them. In the table 1 you can see the results.

<table>
<thead>
<tr>
<th>Facial shape animation</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>JawOpen</td>
<td>Is to open the chin, not necessarily open the mouth</td>
</tr>
<tr>
<td>LipPucker</td>
<td>Pout</td>
</tr>
</tbody>
</table>

Table 1. Animation units and their meaning
<table>
<thead>
<tr>
<th>Animation Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JawSlideRight</td>
<td>Slide the chin to the right</td>
</tr>
<tr>
<td>LipStretcherRight</td>
<td>Mouth stretch to the right</td>
</tr>
<tr>
<td>LipStretcherLeft</td>
<td>Mouth stretch to the left</td>
</tr>
<tr>
<td>LipCornerPullerLeft</td>
<td>Pull the corner of the mouth to the left</td>
</tr>
<tr>
<td>LipCornerPullerRight</td>
<td>Pull the corner of the mouth to the right</td>
</tr>
<tr>
<td>LipCornerDepressorLeft</td>
<td>Pull left corner of the mouth down, sad</td>
</tr>
<tr>
<td>LipCornerDepressorRight</td>
<td>Pull right corner of the mouth down, sad</td>
</tr>
<tr>
<td>LeftcheekPuff</td>
<td>Inflate left cheek</td>
</tr>
<tr>
<td>RightcheekPuff</td>
<td>Inflate left cheek</td>
</tr>
<tr>
<td>LefteyeClosed</td>
<td>Left eye closed</td>
</tr>
<tr>
<td>RighteyeClosed</td>
<td>Right eye closed</td>
</tr>
<tr>
<td>RighteyebrowLowerer</td>
<td>Right frown</td>
</tr>
<tr>
<td>LefteyebrowLowerer</td>
<td>Left frown</td>
</tr>
<tr>
<td>LowerlipDepressorLeft</td>
<td>Pull down the left lower lip</td>
</tr>
<tr>
<td>LowerlipDepressorRight</td>
<td>Pull down the right lower lip</td>
</tr>
</tbody>
</table>

Once we discovered the function of every animation unit, we have to convert this value in the blend shape value. For this, we tried to simply put the number Kinect gives us between 0 and 1, into a number between 0 and 100. This obviously didn’t
work well. So, we needed to know how the number was determined, in the chapter 5 we explained briefly explained the theory behind the algorithm that uses Face HD to get the points and the animation units. To explain easily the problem, this animation units are decided by comparing the face in the frame with a database of different faces with the expressions. But every user has a different face shape so is not really accurate, and even if you have the eye closed, is probable that the eye closed animation unit is not going to be a 1, Kinect tries to deform the face in the input in order to adapt to the neutral face of the database, but this is not enough. What we did, was to create a pre-program that trained the program by doing the different expressions. For example, we keep the eye open, and we store the lowest plus 0.10 as the minimal value. Then we close the eye and we store the value minus 0.10 as the maximum value. Then in the program we these values as in the Figure 20.
6.4 Phase 3: Unity C# coding

For creating the 3D application that controls the model, we are using the software Unity 5.4. We are going to use the programming language C# because it is the most used and powerful. First of all, we have to know that Unity, since the 5.0 version, has its own Kinect library. You have to download these libraries from the Kinect webpage. You have to import this package with the libraries into your Unity project and then say in every script that uses Kinect that you are using these libraries (Figure 21).
We need to know how you use coding in Unity. In Unity, you have a huge 3D environment where we can put and create different objects. These objects are called Game Objects and have their own inside components. These components determine the behaviour of the Game Object. In Unity, we have a huge variety of components to add to our Game objects (Figure 22). With the basic ones, we can change the position or texture of the object. But we also can create physics for the interaction with other Game Objects. We can watch and control all of these components on the Unity inspector.
But we are not constrained to only these components. Unity also give us the opportunity of creating our own components. This is called Scripting, what we do is to create a script with code that controls the behaviour of the object as we wish. We can control internal parameters, trigger game events etc. And this is what we are going to use to create Kinect controls.

With the Kinect utilities for unity comes a few predesigned scenes with some useful objects and scripts that we can use for our own scene. The Scripts we are going to reuse are the ones related to the cameras. This object will show us the image that the different cameras are taking (Figure 23).

**Figure 22: Unity object inspector**
Once we have this setup we can include our model and start scripting the new managers. We need script that controls the head tracking system, with the Kinect Face libraries. Also, we need to create a controller of the blend shapes. We have already created the code for the Face tracking system in the second phase for visual studio. We can almost just copy that code, we only need to have in mind how the scripts work in Unity. We have a Start method that is only executed once. We will use this method to initialise components and open the different sensors that we need. We have to be careful because some sensors are no longer created by constructors with these libraries. In these cases, the main sensor usually has a create
method that acts as the constructor. After this, we have an Update method, this method is called in every frame. So, we will put here the code that analyse the frames of the camera for information. Then we need to create a public variable that contents the important information that we get. In this case, the different values of the animation units. In the code, we made for visual studio, we put all the code of every sensor and camera in the same script so we only opened the Kinect sensor once. Is important to remember that in Unity, we have separated the code in different scripts to use them in objects with different functions. For example, in the Figure 23 we see three Game Objects that show us the different images that Kinect send us. Each one of them has its own script that uses the Kinect sensor and open it. So, is important that in the code we include a line to avoid trying to open the Kinect sensor when is already open, we can see this line in the figure 8. Also, is possible that the system falls or that we close it in a bad way. So, we need to add a function that is called OnApplicationQuit () in every script that closes the sensor on these scenarios (figure 24).

```csharp
void OnApplicationQuit()
{
    if (_bodyReader != null)
    {
        _bodyReader.Dispose();
        _bodyReader = null;
    }
    if (_faceReader != null)
    {
        _faceReader.Dispose();
        _faceReader = null;
    }
    if (_sensor != null)
    {
        if (_sensor.IsOpen)
        {
            _sensor.Close();
        }
        _sensor = null;
    }
}
```

Figure 24: Example of OnApplicationQuit () function
With the code, we have from visual studio, we will have access to all the animation units that Kinect HD gives us. Also, we can have access to the data of the 1000 points that it tracks in the face. Now we need to create the program or the algorithm that adapts these values to a blend shape weight adapted to our face. We decided to do this also in real-time. For this, we create two public arrays of values, maximum and minimum values, and in every frame and for every animation unit we compare the incoming value with the maximum and minimum, if the incoming value is higher than the maximum, we store it as maximum also if its lower than the minimum we store it at minimum.

Now we just have to create a manager to set the values to the blend shapes, we need access to the arrays maximum, minimum and animation units' value of the face HD manager, in the Figure 25 we can see how to do this. With these values, we just implement the algorithm of the Figure 20 to create the blend shape weight.

```csharp
4 public class blendshapehd : MonoBehaviour {
5     facehd facehd;
6     public float[] aniUnits;
7     SkinnedMeshRenderer skinnedMeshRenderer;
8
9     void Awake ()
10     {
11         skinnedMeshRenderer = GetComponent<SkinnedMeshRenderer>();
12         facehd = GetComponent<facehd>();
13     }
14 }
```

**Figure 25: Using values of other scripts in Unity**

On Figure 26 we can see a flowchart of all the steps in the program.
Figure 26: Flowchart of the Face HD code
7 RESULTS

Finally, in this chapter we will talk about the results obtained with the process we detailed in the last chapter. As expected, when we start the program, the model shows unexpected behaviour, this is because you need to train it at the beginning. We have to make the different expressions so the algorithm takes our range of values and apply it to the blend shape weight. Once we pass this initial state, the program makes a good job to copy the expressions and translate them into the model. On Figure 27 we can see the blend shape we created for the test of the program. They are not great blend shapes, and we are sure that with more worked and detailed blend shapes, the results would be much better. In the Figures 28, 29, 30, 31, we can see different screenshots of a test session.

![Figure 27: Blend shapes created for the test](image-url)
Figure 28: neutral face before and after calibration

Figure 29: smiling and surprise
Figure 30: wink and eyebrows up

Figure 31: Cheek puff and mouth open
8 BIBLIOGRAPHY


