Ain Shams University
Faculty of Computer & Information Sciences
Computer Science Department

Facial Expression Motion Capture

By:

Islam Hussien Sallam
Basel Saied Mohamed
Osama Amgad Lahzy
Asmaa Alaa Abdelhi
Aya Saeed Mohamed

Under Supervision of :

[Dr. Maryam Nabil]
[Lecturer],
Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

[Yoman Ahmed]
[Teaching Assistant],
Computer Science Department,
Faculty of Computer and Information Sciences,

July 2020

Ain Shams University
Faculty of Computer & Information Sciences
Computer Science Department

Facial Expression Motion Capture

By:

Islam Hussien Sallam
Basel Saied Mohamed
Osama Amgad Lahzy
Asmaa Alaa Abdelhi
Aya Saeed Mohamed

Under Supervision of :

[Dr. Maryam Nabil]
[Lecturer],
Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

[Yoman Ahmed]
[Teaching Assistant],
Computer Science Department,
Faculty of Computer and Information Sciences,

Acknowledgement

On the submission of this report, Facial Expression Motion Capture, we would like to thank ALLAH for granting us the power, faith, and blessings throughout this project, and we are really grateful that we could manage to complete our project within time.

This project cannot be completed without the effort, hard work, dedication, and cooperation from our group members.

We would like to express our gratitude to our supervisor, Dr. Maryam Nabil, for her support, guidance, motivation, and help all the time, whose knowledge, patience, and understanding added considerably to our experience.

We also want to thank our teaching assistant, Yomna Ahmed, who has helped us since day one. Special thanks for her help, support and devotion.

We would also like to thank our families and friends for the support they provided us through our entire life, without their love, and encouragement we would not have reached this stage of our life and we would not have had succeeded.

Finally, we are grateful to everyone who helped us, or contributed to this project.

Abstract

Through decades of research we have learned that capturing the shape and motion of real human face can lead to realistic virtual characters. We know that Mastering facial animation is a long-standing challenge in computer graphics. We aim to track the motion of any face on the camera and construct a 3D-facial expression of that character. Audiences are particularly trained to look at faces and identify these subtle characteristics. We aim to make this real by providing that project as a support for that kind of AI applications.
This idea is helpful to the area of computer graphics. We can apply lots of applications from this idea as the face can describe the emotions of a character, convey their state of mind and hint at their future actions. So, we can apply this approach to video games and video animations to make it more realistic. We think This research can be helpful to make analysis of a character throw its emotions.

In our system, we used face detection algorithms, landmarks detection, pose estimation and produced the animated results using 3D-model rigging. The results that we got about this project it can capture the performance of the human face in real time from a video or a camera person and apply this human performance on 3D model.

Table of Contents

List of Figures

List of Abbreviations

Abbreviation	What the abbreviation stands for
DLT	Direct Linear Transform
HOG	histogram of oriented gradients
PnP	Perspective-n-Point

Chapter One:

Introduction

1- Introduction

1.1 Motivation

As computer vision engineers and researchers we have been trying to understand the human face since the very early days. Motion Capturing of the face performance is tracking the face motions in real time. Motion Capture can be helpful to the area of computer graphics. We can apply lots of applications from this idea as the face can describe the emotions of a character, convey their state of mind and hint at their future actions. So, we can apply this approach to video games and video animations to make it more realistic. We think This research can be helpful to make analysis of a character show its emotions.

The portrayal of high-quality animation is of great importance for the creation of appealing characters in both realist and stylised contexts in feature films and video-games. The massive use of computer graphic for the recreation of fantastic and imaginary creatures in filmmaking and visual effects makes very demanding the investigation of new methods and efficient techniques that create believable animation, specifically when working within realistic contexts Ping et al. [1]. Believable animation is achieved through the correct portrayal of emotion and the illusion of thought process in the characters, giving adequate traits and reactions to them in terms of body and facial motion Lasseter [2]. A key feature for believability in animation is the facial expression and the acting of the characters. The sensitivity of the human eye towards expressions makes the face the most observed feature and the most prone to be subject of critique, given the high amount of subtleties that can be found in each expression Orvalho et al. [3]. Also, the complexity of the structure and the muscles of the face makes it hard to simulate it properly Cong et al. [4]. Facial animation, therefore, presents itself as a challenging area within animation both for the artist and the technical departments. To achieve good results in this matter, great effort is put in the articulation process of the face (commonly known as rigging), so animators can have enough control to articulate the desire expressions in the characters. Recent advances in this area have reached levels in which the portrayal of complex facial expressions has been achieved successfully, both in realistic and stylised context (see Figure 1). However, current rigging techniques for the face require a considerable amount of effort and resources when working with realism and subtle expressions and, in many cases, they result in cumbersome controllers difficult for animators to handle, getting in the way of the animation process when trying to achieve certain expressions or convey subtle changes such as micro-expressions Orvalho et al. [3]. Finding better techniques to facilitate this workflow in a more fluid and organic manner is an important subject to study in computer animation.

Figure 1. Still images from Coco (Disney Pixar 2017), Avengers: Infinity War (Marvel Studios 2018) and Hellblade: Senua's Sacrifice (Ninja Theory 2017).

1.2 Problem Definition

The face can describe the emotions of a character, convey their state of mind and hint at their future actions. Capturing these features is difficult because it requires lots of phases to be done.

Each of these phases has its own challenges such as: face detection, facial landmarks, pose estimation, Kalman filter, eye-blinking detection, Facial expression features extraction and 3d facial model expression control

These challenges need knowledge of:

1. Computer vision
2. Unity development
3. Socket programming
4. 3d modeling and rigging

1.3 Objective

We aim to make a program that uses the video camera to capture Facial expressions. Then analyzes these expressions. After That the program generates a 3D tracker that mimic this movements and Apply this movements to the 3D model that we have. Our second goal was to apply lots of concepts in computer science like Socket programming, computer vision and 3D-model manipulation.

Project Main Objectives:

1- Make the video camera capture face performance.
2- Analyze the face performance.
3- Making a 3D tracker that mimic this movements.
4- Apply this movements to the 3D model that we have.

1.4 Time Plan

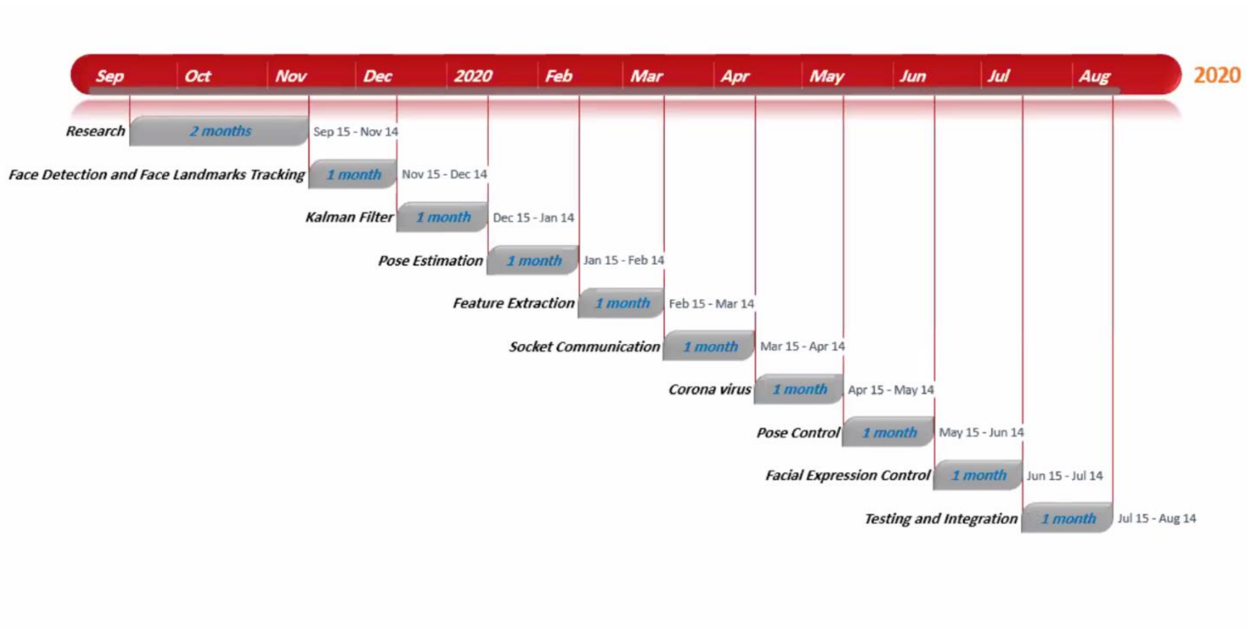


Figure 2. Time Plan

1.5 Document Organization

Chapter 2: cover a detailed description of our project and its main phases. It also covers a project survey and similar systems.

Chapter 3: Includes the system architecture and its users and the diagrams of the system.

Chapter 4: Includes the implementation details of the project and the challenges that we solved.

Chapter 5: it has the user manual and screen shots of the project testing and exe runs and an installation guide.

Chapter Two:

Background

2- Background

2.1 detailed description and scientific background

The most obvious application of facial analysis is Face Recognition. But to be able to identify a person in an image we first need to find where in the image a face is located. Therefore, face detection — locating a face in an image and returning a bounding rectangle / square that contains the face was a hot research area. In 2001, Paul Viola and Michael Jones pretty much nailed the problem with their paper titled [5] In the early days of OpenCV and to some extent even now, the killer application of OpenCV was a good implementation of the Viola and Jones face detector.
Once you have a bounding box around the face, the obvious research problem is to see if you can find the location of different facial features (e.g. corners of the eyes, eyebrows, and the mouth, the tip of the nose etc) accurately. Facial feature detection is also referred to as "facial landmark detection", "facial key-point detection" and "face alignment" in the literature.
Once you know a few landmark points, you can also estimate the pose of the head. In other word you can figure out how the head is oriented in space, or where the person is looking.

Since the location of the detected landmarks are quite noisy which would make pose estimation extremely unstable, a Kalman filter is used to track the landmarks, which means the history information is used rather than just relying on the detected location from current frame alone.

In computer vision the pose of an object refers to its relative orientation and position with respect to a camera. You can change the pose by either moving the object with respect to the camera, or the camera with respect to the object.

The pose estimation problem described in this project is often referred to as Perspective-n-Point problem or PNP in computer vision but we will use the DLS algorithm rather than the PNP in this problem the goal is to find the

pose of an object when we have a calibrated camera, and we know the locations of n 3D points on the object and the corresponding 2D projections in the image.
After that we apply results to a 3D model that has blend-shapes to make it track the performance of the face.

2.2 survey of the work

The beginning of computer facial animation research has its origins in 1972, in the work of Parke [6] with the proposal of the first computer generated animated face, in a period when computer graphics was starting to develop and to gain more attention. This pioneer attempt would open a new branch of research, leading to the development of new techniques for facial animation such as the work of Gillenson [7], who proposed the first interactive system to create 2D facial images, and Platt [8] on the first facial model controlled by muscle simulation. These highlights would lead to the development of further research and the creation of the first computer animated shorts depicting facial animation, such as Tony de Peltrie [9] and Tin Toy Disney Pixar [10]. Since then, many approaches have been developed in relation to the creation of more realistic facial animation.
In recent years, the techniques for the recreation of realistic facial animation have reached high levels of fidelity, with examples such as the work of Alexander et al. [11], with the photorealistic recreation of the animated face of an actor, or Ichim et al. [12] on the use of physics-based simulation for the realistic animation of facial muscles and flesh. The state-of-the-art of this field offers now a wide range of solutions adapted to each of the possible scenarios in which facial animation plays a key role, both in the entertainment industry and in more academic and scientific applications. Continuous research effort is being invested on achieving more accurate results and better performance and expressivity in digital facial models.

The development within computer facial animation has brought up a wide range of techniques for the recreation of digital faces. The following sub-sections cover some of the most common methods used nowadays: bone-based with blend shapes, physics based and motion capture based. 6 Bone-Based and Blend shapes In general terms, facial animation can be considered a special case for body animation, and in many cases the methods and solutions employed for body rigging can be applied to facial rigging. As such, a common practice in facial rigging involves the use of a bone based rig to set the deformation of its main features, such as the cheeks, the eyelids and the jaw. Along with this, blend shapes allow the artists to sculpt certain features and details of the face that are difficult to achieve with bones, and these two techniques in combination are usually enough to reproduce a believable set of expressions with high efficiency McLaughlin et al. [13]. In a higher level of detail, a related approach for facial animation consists on the use of a dictionary of blend shapes, sculpting in the model all the necessary expressions for the performance, and then creating the animation by shape interpolation Orvalho et al. [3] based on techniques such as bilinear and spline interpolation Arai et al. [14]. A pioneer work in film is the case of the creature Gollum, from The Lord of the Rings New Line Cinema [15], in which a total of 675 blend shapes were used to reproduce all the range of possible expressions for the animation of his face Orvalho et al. [3]. Though this method can achieve high fidelity in the performance and expressivity of the character, it is restricted to the range of sculpted expressions, and requires high computational resources. Physics-Based and Muscle Systems Physics-based techniques rely on heavy calculations and precise models for the animation of the face, attempting to simulate the elastic properties of facial tissue and muscles. The use of muscle systems is one of the most accurate methods within this area of facial animation, consisting on the simulation of the whole underlying muscle anatomy of the face, and calculating how it deforms the skin tissue on top Parke and Waters [16]. In recent years, the film industry has started using these techniques more frequently, with remarkable examples such as Kong Warner Bros [17], in which muscle systems were used to help in the animation of the expressions Cong et al. [18], Cong et al. [4]. Physics-based methods have the advantage of being able to reproduce almost any achievable expression on the face, but it requires a considerable effort for the correct sculpting of the facial muscles and the proper simulation of the deformation of the skin, being this the main reason why its use is not very widespread yet. Muscle systems have been used beyond the entertainment industry, and in the case of medical science additional techniques have attempted to simulate the face in a realistic manner for its use in surgery training and medical research. In this domain, physic-based systems are widely used, and the simulation of the face in layers (bone, muscle, fat and skin) is bringing new levels of realism and accuracy Murai et al. [19]. Motion Capture Based Motion capture has become another common approach for the creation of computer facial animation, based on the collection of expression and performance from real actors. The recreation of movement from motion capture usually relies on the use of surface deformation techniques applied to the facial model and driven by the data collected. Among those, RBF is one of the most commonly used for its robustness and efficiency Man-dun et al. [20]. A remarkable example of this technology can be seen in the real-time motion capture performance for Hellblade: Senua’s Sacrifice Ninja Theory [21]. However, this technology still requires considerable amount of efforts to set it up and clean it to make it ready for use, and facial nuances and micro expressions can still be difficult to achieve.

Technique	Pros	Cons
Bone-based	• Fast and efficient. • Intuitive (artist friendly). • Good for stylized models.	• Limitation in details • Difficult to achieve realism by itself.
Blend shapes	• Can bring a wide range of accurate expressions. • Works well together with bone based systems • Works well for both stylized and realistic models.	• Computationally expensive. • Needs large collection of blend shapes to work well. • High level of artist effort.
Physics-based	• High level of accuracy (anatomically correct). • Able to simulate anatomical properties such as fat and skin tissue. • Good for realistic models.	• Computationally expensive. • Difficult to achieve realistic results. • High level of technical effort.
Muscle systems	• High level of accuracy (anatomically correct). • Works well together with additional physics-based simulations (i.e. fat jiggling). • Good for realistic models.	• Difficult to achieve realistic results. • Needs large collection of muscles to work well. • High level of technical and artist effort.

Motion capture	• Based on actor performance. • Able to capture subtle movements. • Works well for both stylized and realistic models.	• Difficult to transfer data without losing accuracy. • Might introduce noise during capture process. • High level of technical effort.

Figure 3. Pros and cons of different facial animation techniques.

2.3 similar systems

This project has similar ones like the paper of [22] present the first real-time high-fidelity facial capture method. The core idea is to enhance a global real-time face tracker, which provides a low-resolution face mesh, with local regressors that add in medium-scale details, such as expression wrinkles. Their main observation is that although wrinkles appear in different scales and at different locations on the face, they are locally very self-similar and their visual appearance is a direct consequence of their local shape. The researchers therefore train local regressors from high-resolution capture data in order to predict the local geometry from local appearance at runtime. they propose an automatic way to detect and align the local patches required to train the regressors and run them efficiently in real-time. Their formulation is particularly designed to enhance the low-resolution global tracker with exactly the missing expression frequencies, avoiding superimposing spatial frequencies in the result. Their system is generic and can be applied to any real-time tracker that uses a global prior, e.g. blend-shapes. Once trained, the online capture approach can be applied to any new user without additional training, resulting in high-fidelity facial performance reconstruction with person-specific wrinkle details from a monocular video camera in real-time. A description of the Architecture of this method is shown in figure 4 below

Another similar system [23] present a novel image-based representation for dynamic 3D avatars, which allows effective handling of various hairstyles and headwear, and can generate expressive facial animations with fine scale details in real-time. They develop algorithms for creating an image-based avatar from a set of sparsely captured images of a user, using an off-the-shelf web camera at home. An optimization method is proposed to construct a topologically consistent morphable model that approximates the dynamic hair geometry in the captured images. They also design a real-time algorithm for synthesizing novel views of an image-based avatar, so that the avatar follows the facial motions of an arbitrary actor. Compelling results from our pipeline are demonstrated on a variety of cases.

Another similar system [24] present a real-time performance-driven facial animation system based on 3D shape regression. In this system, the 3D positions of facial landmark points are inferred by a regressor from 2D video frames of an ordinary web camera. From these 3D points, the pose and expressions of the face are recovered by fitting a user-specific blend shape model to them. The main technical contribution of this work is the 3D regression algorithm that learns an accurate, user- specific face alignment model from an easily acquired set of training data, generated from images of the user performing a sequence of predefined facial poses and expressions. Experiments show that our system can accurately recover 3D face shapes even for fast motions, non-frontal faces, and exaggerated expressions. In addition, some capacity to handle partial occlusions and changing lighting conditions is demonstrated.

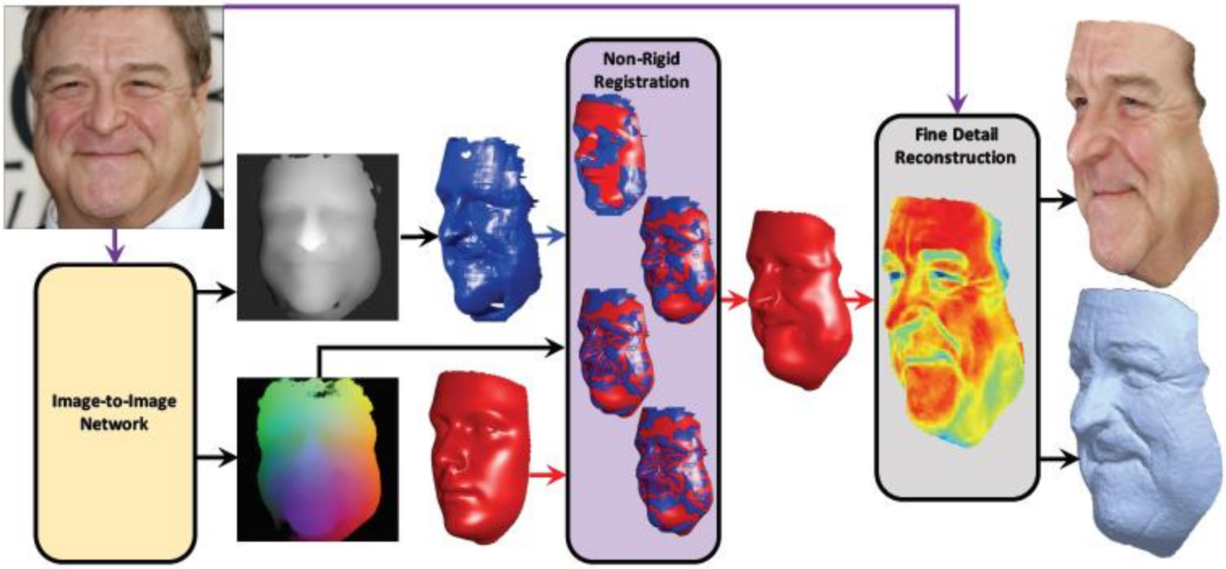


Figure 4. architecture of the high fidelity method

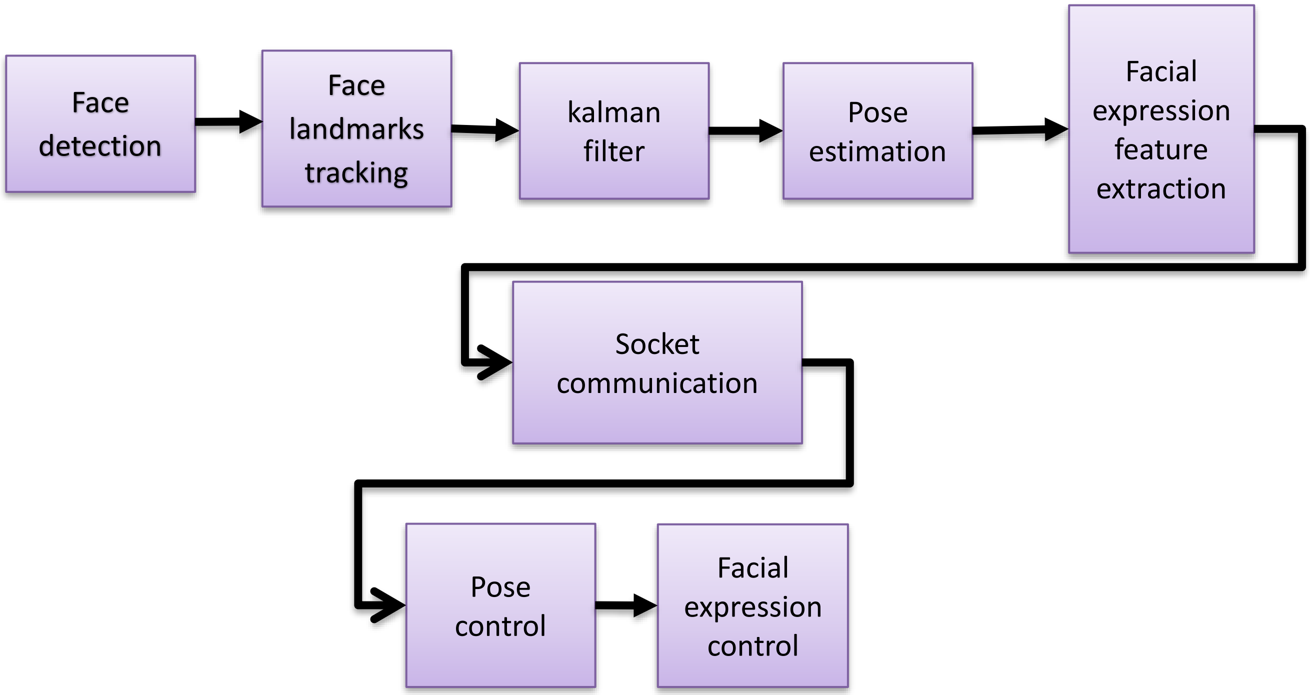
Chapter Three:

Analysis and Design

3- Analysis and Design

3.1 System Overview

3.1.1 System Architecture



we have 2 main phases in the system, detection and extraction part and motion control part, we communicate between them by using socket communication detection and extraction part:

- face landmarks tracking
- kalman filter
- pose estimation
- facial expression feature extraction

motion control part:

- pose control
- facial expression control

detection and extraction part

face landmarks tracking: for estimating and measure the head position, orientation and the facial expression features, 68 landmarks are got first using Dlib and OpenCV. when the location of the detected landmarks are noisy it would make pose estimation unstable. So, we use kalman filter for stability.

kalman filter: we use it to track landmarks , we have two kalman filters are applied to the x-coordinate and y- coordinate of the 68 landmarks , its result will be more smooth and stable.

pose estimation: we use monocular camera through PNP measurement to estimate the position and orientation of an object in 3D scene, we choose 5 feature points to get 3D representation.

facial expression feature extraction: we use EYE_ASPECT_RATIO for detection features. This measure is very basic and straight forward. Nonetheless, it doesn't have great rotational invariance, since the denominator would change a ton unfortunately when the head shakes starting with one side then onto the next. Along these lines, after a progression of tests I develop a more vigorous measure for the eyes' transparency, and furthermore build comparable measures to portray the state of the mouth.

motion control part

pose control : the position is fixed in two models , then we can only control the rotation of it , the four parameters are sent to kalman filter again before the computed quaternions are applied to the model to ensure the smooth motion.

- facial expression control:
1.dealing with noise: by using two steps
step 1: model a dynamic system by using newtons low.
step 2: apply incomplete derivative PD control.

2.blend shape function: to make facial expression of the virtual character more realistic we use blinking function (the most representative and important one), shocked function and mouth deformation function

3.1.2System Users

A. Intended Users:

we are focused on a specific category of people who can handle the most extreme benefit of our system, the system intended users are:

- *game developers*
- *graphic designers*

B. User Characteristics

No extra experience is needed but as the user continues using the application they gain more benefits from it.

3.2 System Analysis & Design

3.2.1 Use Case Diagram

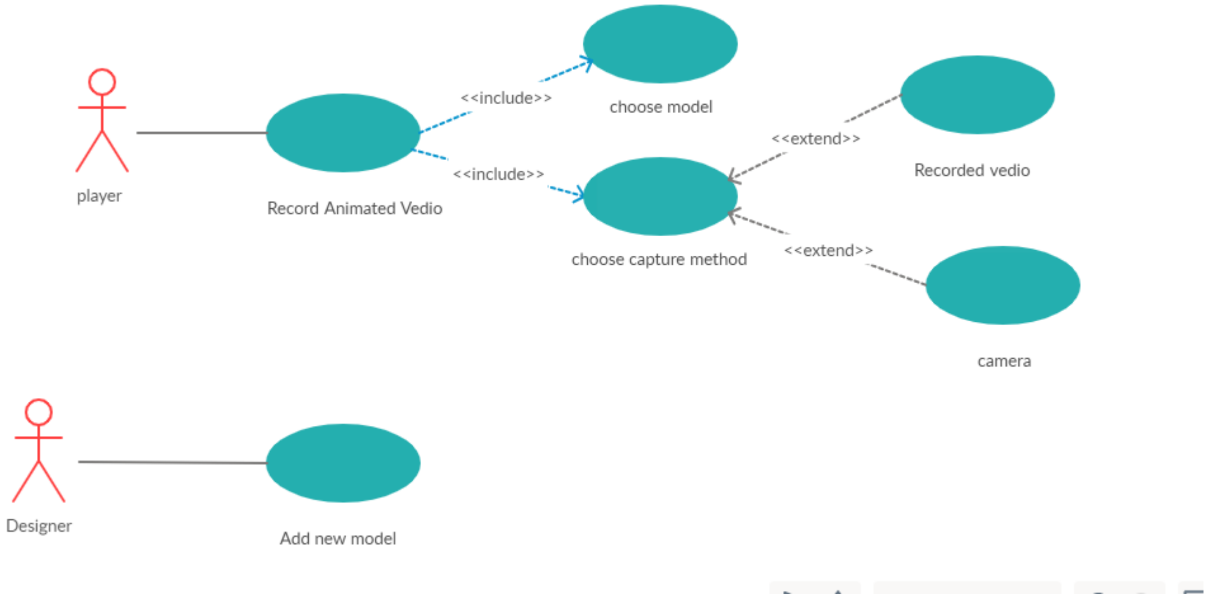


Figure 5. Use Case Diagram

3.2.2 Class Diagram

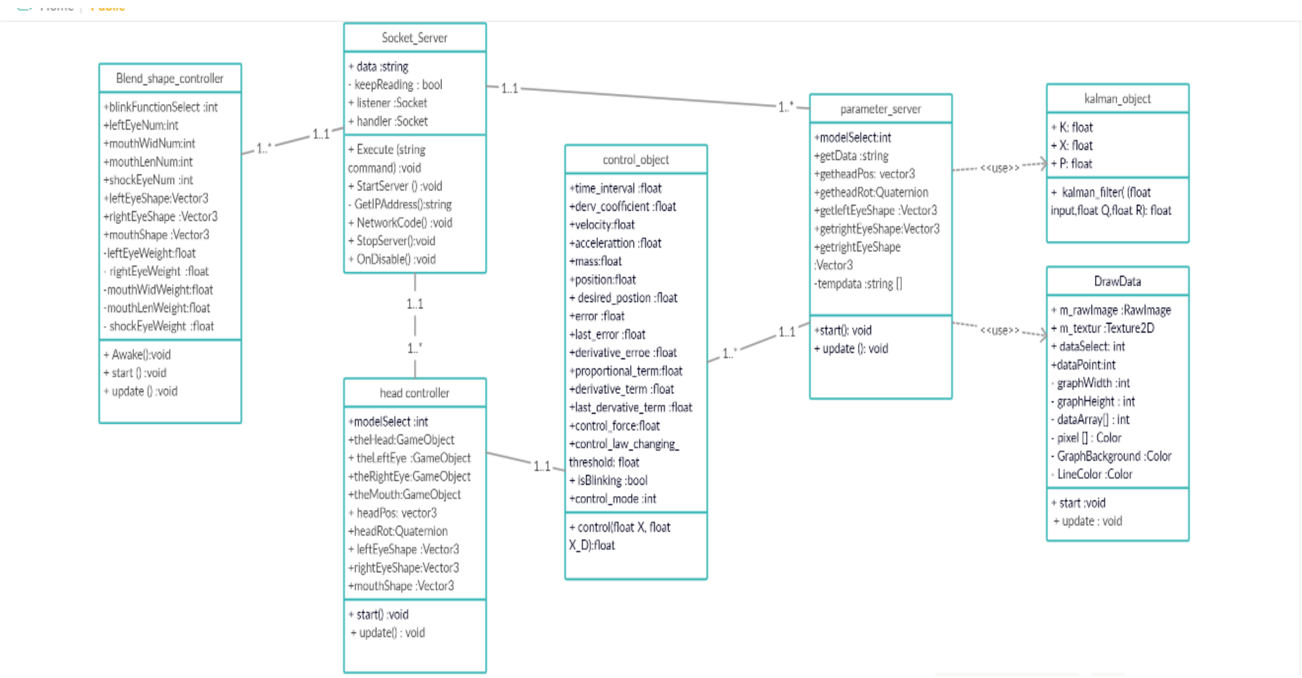


Figure 6. Class Diagram

Socket Server Class
Class Attributes:
network data (IP Address)
Class Methods:
start server connection, connect to the network using IP Address, and finally stop the server

----
Blend Shapes Controller Class
Class Attributes:
Face features data (leftEyeShape, rightEyeShape, mouthShape, shockEyeWeight, mouthLenNum)
Class Methods:
skinned Mesh Rendering, Blinking, set weight to each attribute, Apply deformation weights to each Attributes

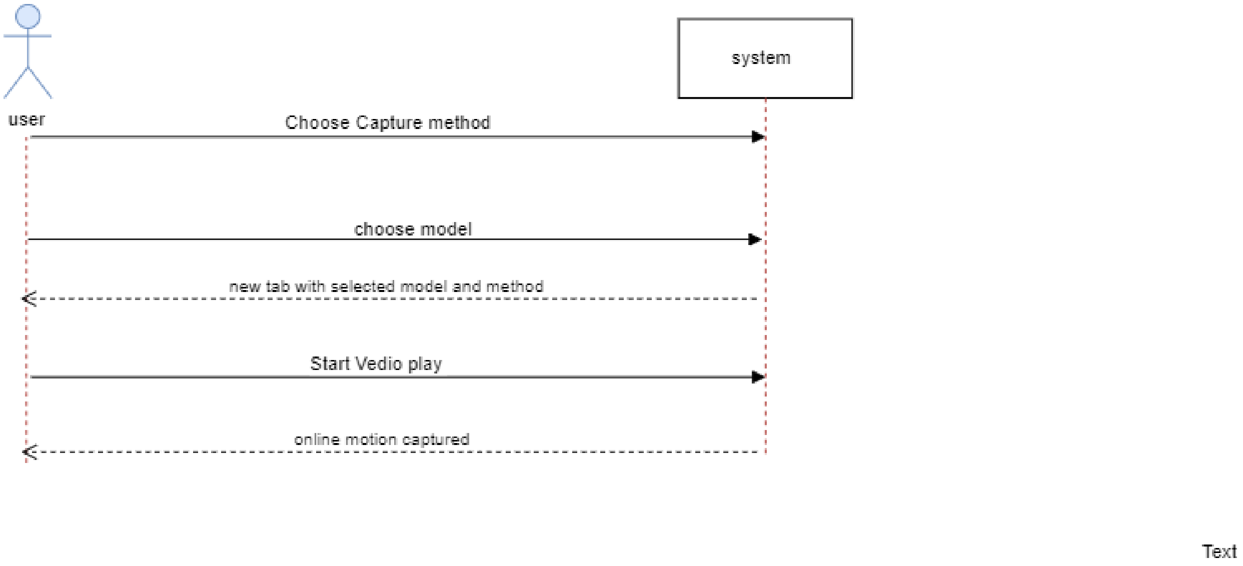
----
Draw Data Class
Class Attributes:
features datapoints, lines, background and texture
Class Methods:
use data points to draw the features on model, adding background color or texture frame by frame

----
Parameter Server Class
Class Attributes:
Kalman objects, Control objects, Head parameters
Class Methods:
select the model and initialize it, get data from socket and update for selected model
----
Control objects Class
Class Attributes:
data of the object over the time
Class Methods:
control object changes over the time using a threshold

Kalman Object Class
Class Attributes:
mean and variance for data points
Class Methods:
use the mean and variance to calculate the missing points due to instability

----
Head Controller Class
Class Attributes:
head and its objects positions
Class Methods:
Apply position and rotation changes and facial expression shapes changes,

3.2.3 Sequence Diagram



Chapter Four:

Implementation and Testing

4- Implementation and Testing

The system can be divided into two parts: Detection and extraction Part (Python part) and Motion Control Part (Unity part).

4.1 Detection and extraction Part (Python)

4.1.1 Face Detection

Detecting the largest face and getting its bounding box points by a dlib's HOG face detector function.

4.1.2 Face landmarks tracking

To estimate the head position, orientation and the facial expressions features, 68 face landmarks are got first using Dlib and OpenCV. Using The model file 'shape_predictor_68_face_landmarks.dat' for face landmarks detection which was created by davis king for part of the dlib example programs, and was trained on the iBUG 300-W face landmark dataset.

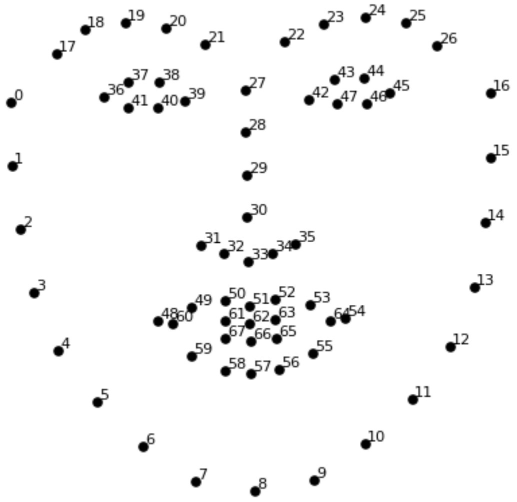


Figure 7. 68 Face landmarks

Since the location of the detected landmarks are quite noisy which would make pose estimation extremely unstable, a **Kalman filter** is used to track the landmarks, which means the history information is used rather than just relying on the detected location from current frame alone.

Specifically, two Kalman filters are applied to the x-coordinate and y-coordinate of the 68 points, respectively.

To make the tracking result even more smooth, the state vector of the Kalman filter is then sent to a **mean filter** which has a window size of 5 (frames).

4.1.3 Pose estimation

The position and orientation of an object in 3D scene can be estimated using a monocular camera through **PnP** measurement. Rather than write a piece of our own, we decided to use the built-in function offered by OpenCV since it has different algorithms to choose. For this part of implementation we used pose estimation algorithm [25].

What is pose estimation?

In computer vision the pose of an object refers to its relative orientation and position with respect to a camera. You can change the pose by either moving the object with respect to the camera, or the camera with respect to the object.

The pose estimation problem described in this tutorial is often referred to as **Perspective-n-Point** problem or PNP in computer vision. As we shall see in the following sections in more detail, in this problem the goal is to find the pose of an object when we have a calibrated camera, and we know the locations of **n** 3D points on the object and the corresponding 2D projections in the image.

How to mathematically represent camera motion?

A 3D rigid object has only two kinds of motions with respect to a camera.

1. **Translation** : Moving the camera from its current 3D location (X, Y, Z) to a new 3D location (X', Y', Z') is called translation. As you can see translation has 3 degrees of freedom — you can move in the X, Y or Z direction. Translation is represented by a vector $\mathbf{t}$ which is equal to $(X' - X, Y' - Y, Z' - Z)$.
2. **Rotation** : You can also rotate the camera about the X, Y and Z axes. A rotation, therefore, also has three degrees of freedom. There are many ways of representing rotation. You can represent it using Euler angles (roll, pitch and yaw), a 3×3 rotation matrix, or a direction of rotation (i.e. axis) and angle.

So, estimating the pose of a 3D object means finding 6 numbers three for translation and three for rotation.

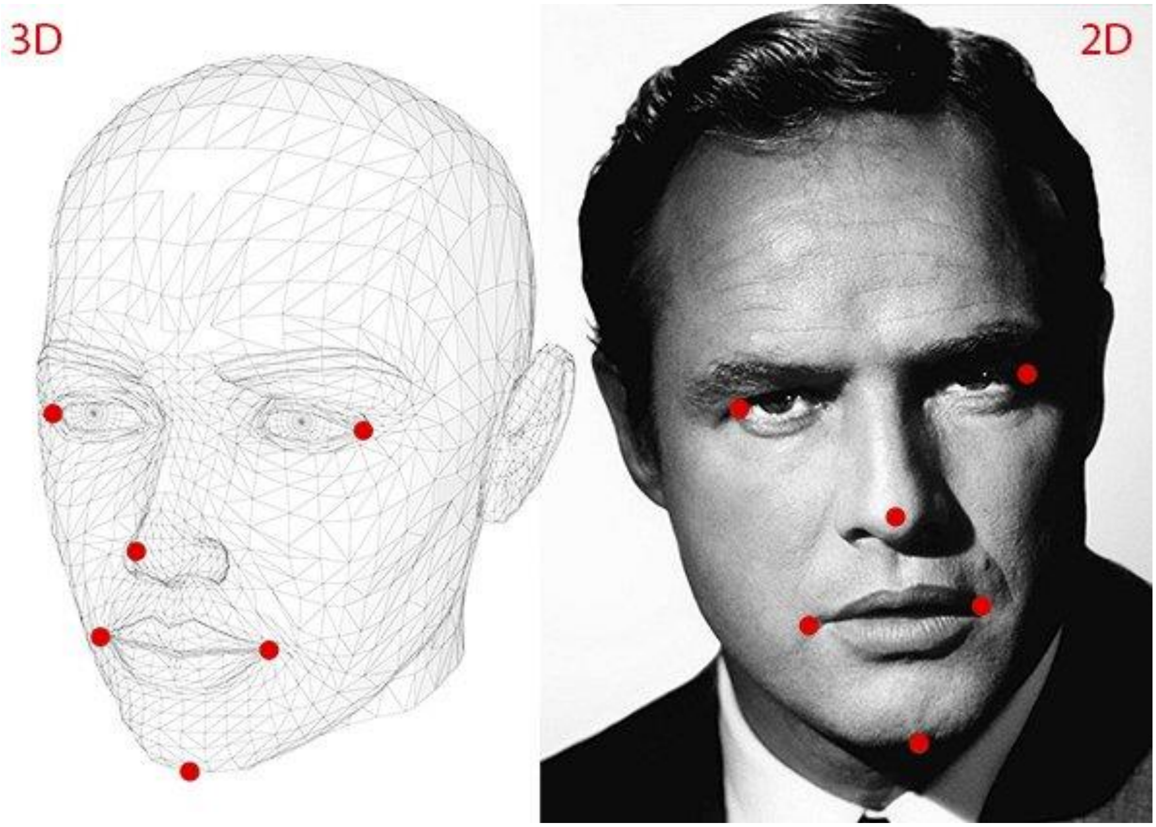


Figure 8. 3d location of detected 2d points

What do you need for pose estimation?

To calculate the 3D pose of an object in an image you need the following information

1. **2D coordinates of a few points** : You need the 2D (x,y) locations of a few points in the image. In the case of a face, you could choose the corners of the eyes, the tip of the nose, corners of the mouth etc. Dlib's facial landmark detector provides us with many points to choose from. In this project, we will use the tip of the nose, the chin, the left corner of the left eye, the right corner of the right eye, the left corner of the mouth, and the right corner of the mouth.
2. **3D locations of the same points** : You also need the 3D location of the 2D feature points. You might be thinking that you need a 3D model of the person in the photo to get the 3D locations. Ideally yes, but in practice, you don't. A generic 3D model will suffice. Where do you get a 3D model of a head from? Well, you really don't need a full 3D model. You just need the 3D locations of a few points in some arbitrary reference frame. In this project, we are going to use the following 3D points.
    1. Tip of the nose : (0.0, 0.0, 0.0)
    2. Chin : (0.0, -330.0, -65.0)
    3. Left corner of the left eye : (-225.0f, 170.0f, -135.0)
    4. Right corner of the right eye : (225.0, 170.0, -135.0)
    5. Left corner of the mouth : (-150.0, -150.0, -125.0)
    6. Right corner of the mouth : (150.0, -150.0, -125.0)

    Note that the above points are in some arbitrary reference frame / coordinate system. This is called the World Coordinates (a.k.a Model Coordinates in OpenCV docs) .

3. **Real parameters of the camera**: As mentioned before, in this problem the camera is assumed to be calibrated. In other words, you need to know the focal length of the camera, the optical center in the image and the radial distortion parameters. So you need to calibrate your camera. Of course, for the lazy dudes among us, this is too much work. Can We supply a hack? Of course, We can! We are already in approximation land by not using an accurate 3D model. We can approximate the optical center by the center of the image, approximate the focal length by the width of the image in pixels and assume that radial distortion does not exist. Boom! you did not even have to get up from your couch!

How do pose estimation algorithms work?

There are several algorithms for pose estimation. The first known algorithm date back to 1841. It is beyond the scope of this post to explain the details of these algorithms but here is a general idea.

There are three coordinate systems in play here. The 3D coordinates of the various facial features shown above are in **world coordinates**. If we knew the rotation and translation i.e. pose, we could transform the 3D points in world coordinates to 3D points in **camera coordinates**. The 3D points in camera coordinates can be projected onto the image plane i.e. **image coordinate system** using the intrinsic parameters of the camera focal length, optical center etc.

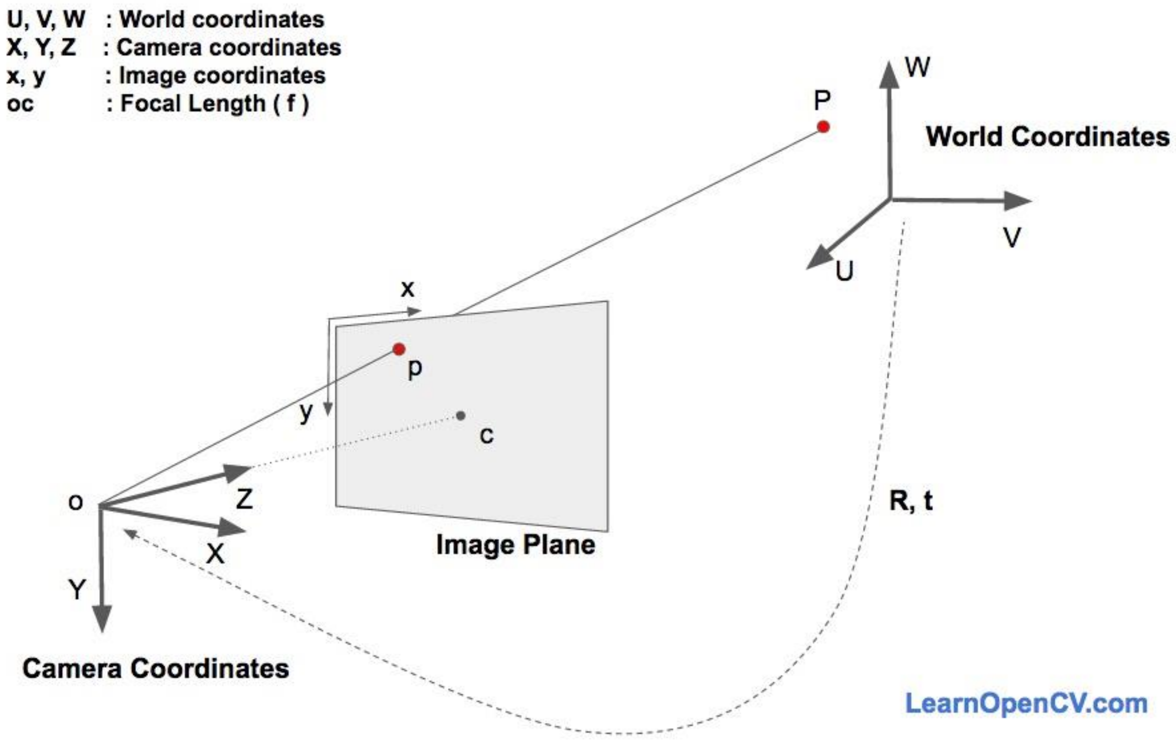


Figure 9. world Coordinates vs Camera Coordinates

Let’s dive into the image formation equation to understand how these above coordinate systems work. In the figure above, o is the center of the camera and plane shown in the figure is the image plane. We are interested in finding out what equations govern the projection p of the 3D point P onto the image plane.

Let’s assume we know the location (U, V, W) of a 3D point P in World Coordinates. If we know the rotation $\mathbf{R}$ (a 3×3 matrix) and translation $\mathbf{t}$ (a 3×1 vector), of the world coordinates with respect to the camera coordinates, we can calculate the location (X, Y, Z) of the point P in the camera coordinate system using the following equation.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathbf{R} \begin{bmatrix} U \\ V \\ W \end{bmatrix} + \mathbf{t}$$

$$\Rightarrow \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = [\mathbf{R} \mid \mathbf{t}] \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}$$

(1)

In expanded form, the above equation looks like this

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}$$

(2)

If you have ever taken a Linear Algebra class, you will recognize that if we knew sufficient number of point correspondences (i.e. (X, Y, Z) and (U, V, W)), the above is a linear system of equations where the r_{ij} and (t_x, t_y, t_z) are unknowns and you can trivially solve for the unknowns.

As you will see in the next section, we know (X, Y, Z) only up to an unknown scale, and so we do not have a simple linear system.

Direct Linear Transform:

We do know many points on the 3D model (U, V, W), but we do not know (X, Y, Z). We only know the location of the 2D points (x, y). In the absence of radial distortion, the coordinates (x, y) of point p in the image coordinates is given by

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = s \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

(3)

where, f_x and f_y are the focal lengths in the x and y directions, and (c_x, c_y) is the optical center. Things get slightly more complicated when radial distortion is involved and for the purpose of simplicty we are leaving it out.

What about that s in the equation? It is an unknown scale factor. It exists in the equation due to the fact that in any image we do not know the depth. If you join any point P in 3D to the center o of the camera, the point p, where the ray intersects the image plane is the image of P. Note that all the points along the ray joining the center of the camera and point P produce the same image. In other words, using the above equation, you can only obtain (X, Y, Z) up to a scale s.

Now this messes up equation 2 because it is no longer the nice linear equation we know how to solve. Our equation looks more like

(4) $$s \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}$$

Fortunately, the equation of the above form can be solved using some algebraic wizardry using a method called Direct Linear Transform (DLT). We can use DLT any time we find a problem where the equation is almost linear but is off by an unknown scale.

However, please note that We have modified some part of the implementation to be compatible with my specific application, which includes:

- Rather than using the 5 **feature points** mentioned in the blog, I choose another set of points since they are more stable than the original ones when the facial expressions become exaggerated
- The rotation vector is converted to **quaternions** to adapt to the Unity applications
- The algorithm to solve PnP is set to **DLS** rather than the default one

4.1.4 Facial expression features extraction

Several studies have been done to extract facial expression features from face landmarks. One of the mostly used feature for eye-blinking detection is called Eye-Aspect-Ratio as shown below:

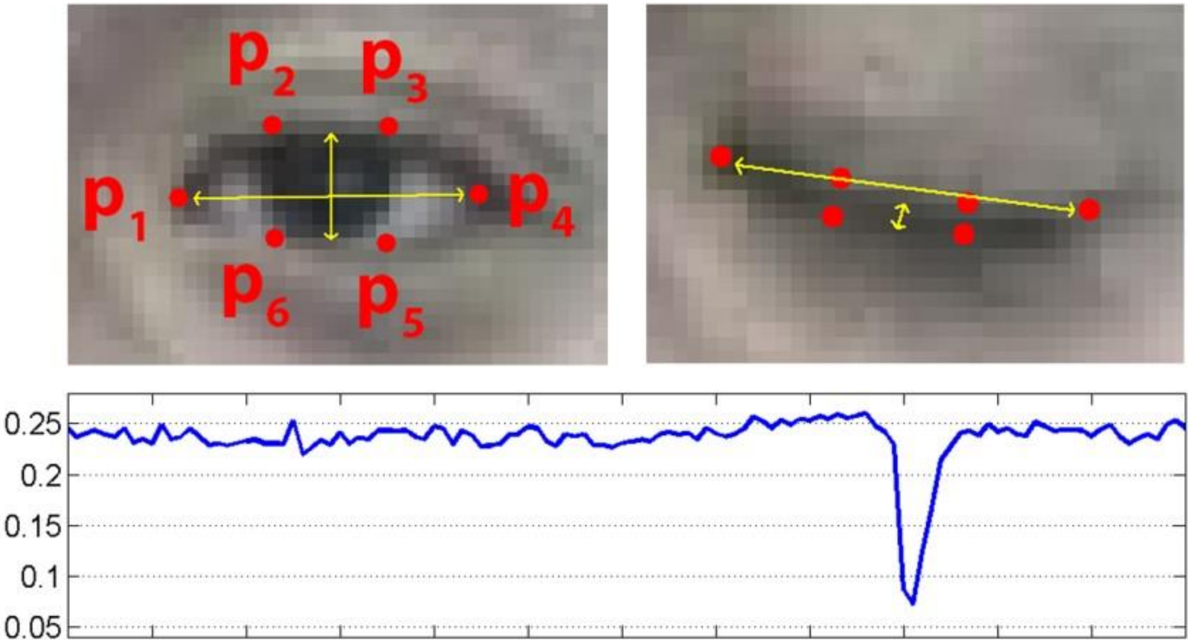


Figure 10. Eye Blinking technique

$$EAR = \frac{||P_2 - P_6|| + ||P_3 - P_5||}{2||P_1 - P_4||}$$

Figure 11. Eq of Eye Blinking technique

This measure is quite simple and straight forward. However, it does not have good **rotational invariance**, since the denominator would change a lot undesirably when the head shakes from one side to another. Thus, after a series of tests We construct a more robust measure for the eyes' openness, and also construct similar measures to describe the shape of the mouth.

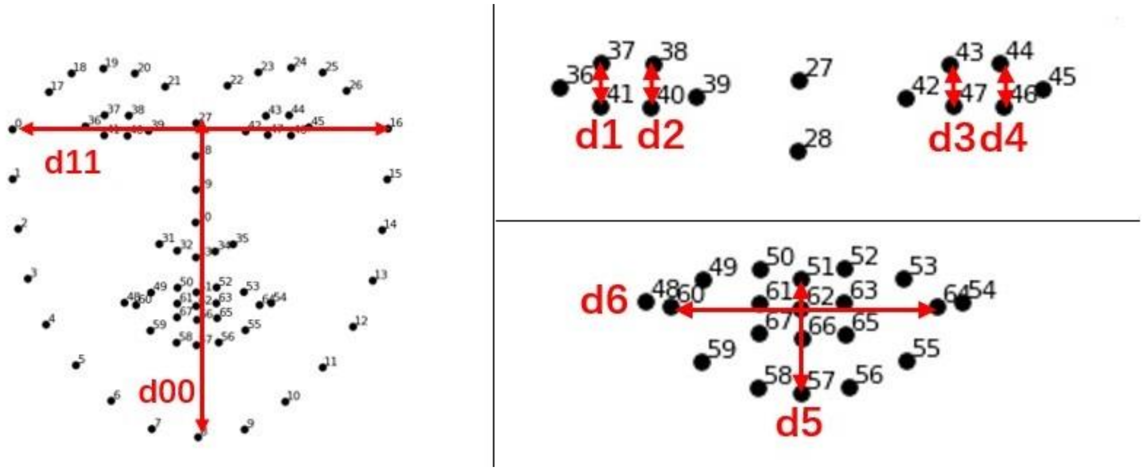


Figure 12. Aspect-Ratio rotational invariance

$$d_{ref} = \frac{d_{00} + d_{11}}{2} = \frac{\|P_{27} - P_8\| + \|P_0 - P_{16}\|}{2}$$

$$d_1 = \|P_{37} - P_{41}\|$$

$$\vdots$$

$$d_4 = \|P_{44} - P_{46}\|$$

$$leftEyeWid = 6 \times (\frac{d_1 + d_2}{2d_{ref}} - 0.02)$$

$$rightEyeWid = 6 \times (\frac{d_3 + d_4}{2d_{ref}} - 0.02)$$

$$mouthWid = 1.27 \times (\frac{d_5}{d_{ref}} - 0.13) + 0.02$$

$$mouthLen = \frac{d_6}{d_{ref}}$$

Figure 13. Eq of Aspect-Ratio rotational invariance

The core idea is to define a reference distance which is insensitive to the rotation.

4.2 Motion Control Part (Unity C#)

After the position, rotation and feature vectors are got at the Python front-end, they are then transferred to the C# back-end through socket, which is described in section 4.3. With this collection of information, several steps are taken here to actually make the virtual character move up properly. The implementation details are slightly different between the two models. Most of time we will only take the model 2 as the example because it has more generality.

4.2.1 Pose control

In fact, theoretically you can directly apply the estimated position vector and rotation quaternions to the model if you use something like model 1 (pure head). However, since we have not implemented inverse kinematics to the more generalized model like model 2, for these models the position is fixed, which means you can only control the rotation of it.

Furthermore, since the setting of world coordinate system are different in OpenCV and in Unity, a new, fixed **rotation transformation** is applied to the quaternions.

Before the computed quaternions are applied to the model, the four parameters are sent to a **Kalman filter** again to ensure the smooth motion.

4.2.2 Facial expression control

4.2.2.1. Dealing with noise

As is mentioned in 4.1.1, the landmarks used for facial expressions are un-filtered due to some considerations, as a result the measures we got here (left Eye W id, right Eye W id, mouth W id, etc.) are quite noisy. To solve this problem, we think up an interesting method that turn it into a **control problem**.

Specifically, we take the **input measure** as the **desired position** of a **mass**, and an **input force** given by **incomplete derivative PD control** is attached to the mass. Then the **actual position** of the mass is used as the **output measure**.

Speaking from another perspective, what we have done is actually constructing a **second-order mass-spring-damper system** with a low pass filter to model this process.

One of the test results is shown in the figure below. On the left is the original signal and on the right is the processed signal.

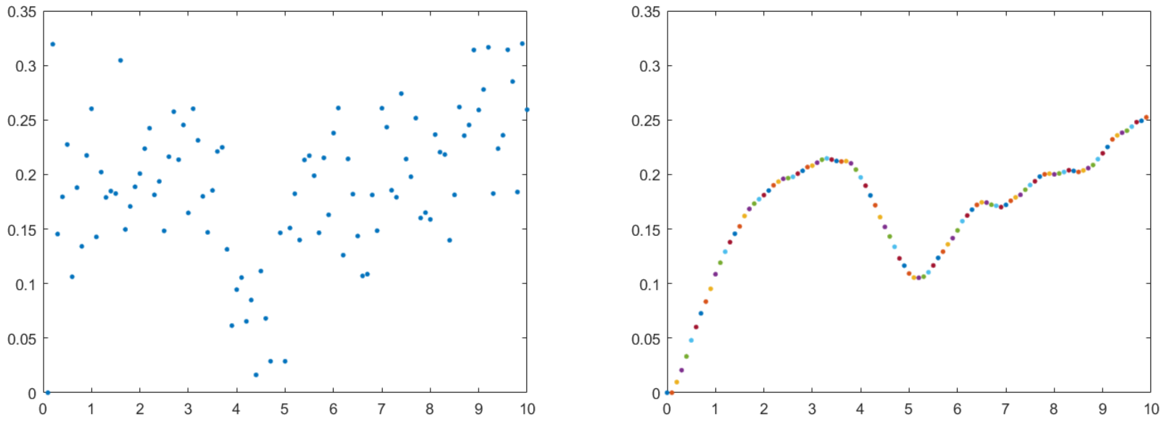


Figure 14 non-processed vs processed signal

The method can be divided into two steps.

Step 1. Model a dynamic system with Newton's law

```
a = F/M; // Update acceleration
v = v + a*T; // Update velocity
x = x + v*T; // Update position
```

Where **T** is the time interval, which is set to 0.1 (s), and the mass **M** is set to around 1-5 (kg). Note that these two parameters should be compatible with each other.

Step 2. Apply incomplete derivative PD control

After the input measure is passed to **x_d**, run the following lines:

```
e = x_d - x;      // Update error
de = (e - e_1)/T; // Compute the derivative of error
p_out = KP*e;     // Proportional term
d_out = (1-ALPHA)*KD*de + ALPHA*d_out_1; //Derivative term

e_1 = e;          // Update last error
d_out_1 = d_out;  // Update last derivative term
```

Where **KP**, **KD** are parameters for the PD controller and **ALPHA** is the incomplete derivative coefficient. The response characteristic of the controller can be altered by tuning these parameters. To ensure robustness, the following relation is always kept to make sure that the system is **overdamped**:

Here are two frequency response diagrams of the system with ALPHA = 0 and ALPHA = 0.7, respectively. They can reflect how much the high-frequency noise would affect the output.

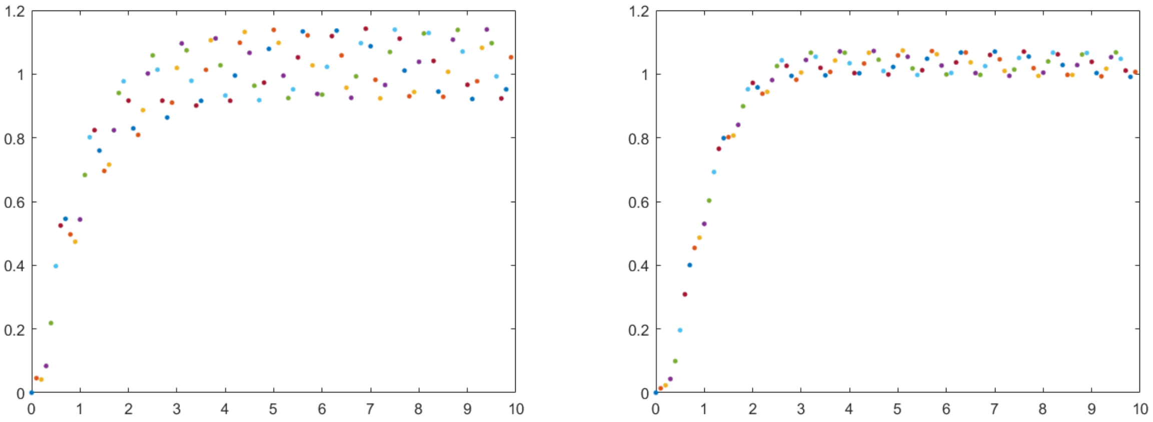


Figure 15. frequency with Alpha = 0 vs Alpha -0.7

More details of the implementation can be referred to ParameterServer.cs.

4.2.2.2 Blend shape functions

To make the facial expression of the virtual character more realistic, we write customized deformation functions for the blend shapes of model 2: **Blinking function**, **Shocked function** and **Mouth deformation function**. Since the function may vary according to the specific model, we will just list the most representative and important one here, which is the eyes' **Blinking function**. It has two versions at present.

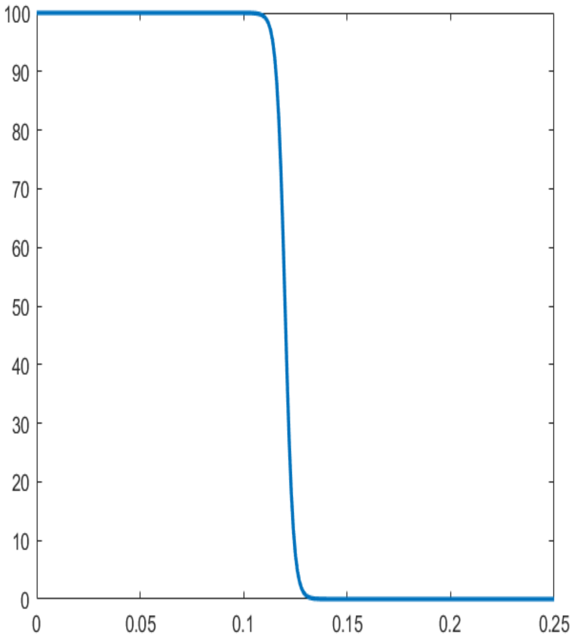
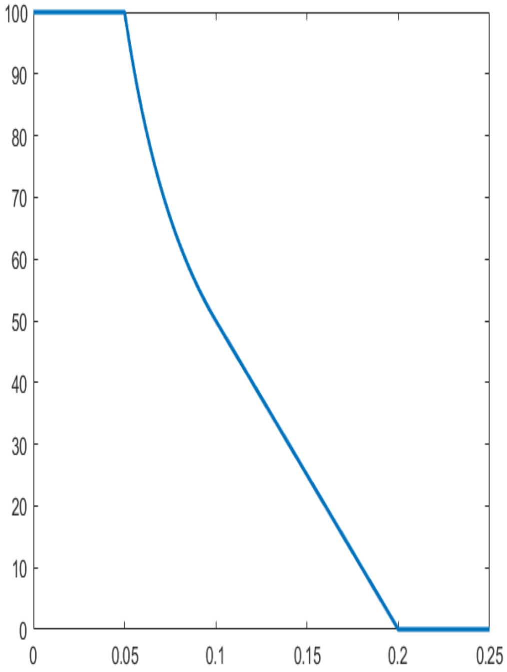
Version 1: A parameterized sigmoid function	**Version 2: A piecewise function**
	
$w = 100 - \dfrac{100}{1 + \mathrm{e}^{-500(m-0.12)}}$	$w = \begin{cases} 100 & m < 0.05 \\ 5/m & 0.05 \leq m \leq 0.1 \\ 100 - 500/m & 0.01 \leq m \leq 0.2 \\ 0 & m \geq 0.2 \end{cases}$

Figure 16. Deformation Functions

Where *m* is the processed measurement. *w* is the weight applied to the blend shape controller, ranging from 0 (no deformation) to 100 (max deformation).

4.2.2.3 A little trick:

When tuning the parameters, there is always a contradiction between robustness and sensitivity. Especially when controlling the shape of the eye, it is reasonable to keep it smooth, which requires a longer response time, but that would also make the detection of blinking more challenging. To solve this problem, I use a small trick here.

- **In the dynamic system part:** While keep the system as smooth as you can, **force** the "position", that is, the measure, **to be zero** when the original measure is lower than a pre-set threshold.
- **In the blend shape part:** Use the same threshold as the upper bound for 100 weight (eye fully closed).

The following figure demonstrates the difference of the system response without and with this trick. T1, T2 and T3 are the eye-closed duration of the original response, while T is eye-closed duration of the new response.

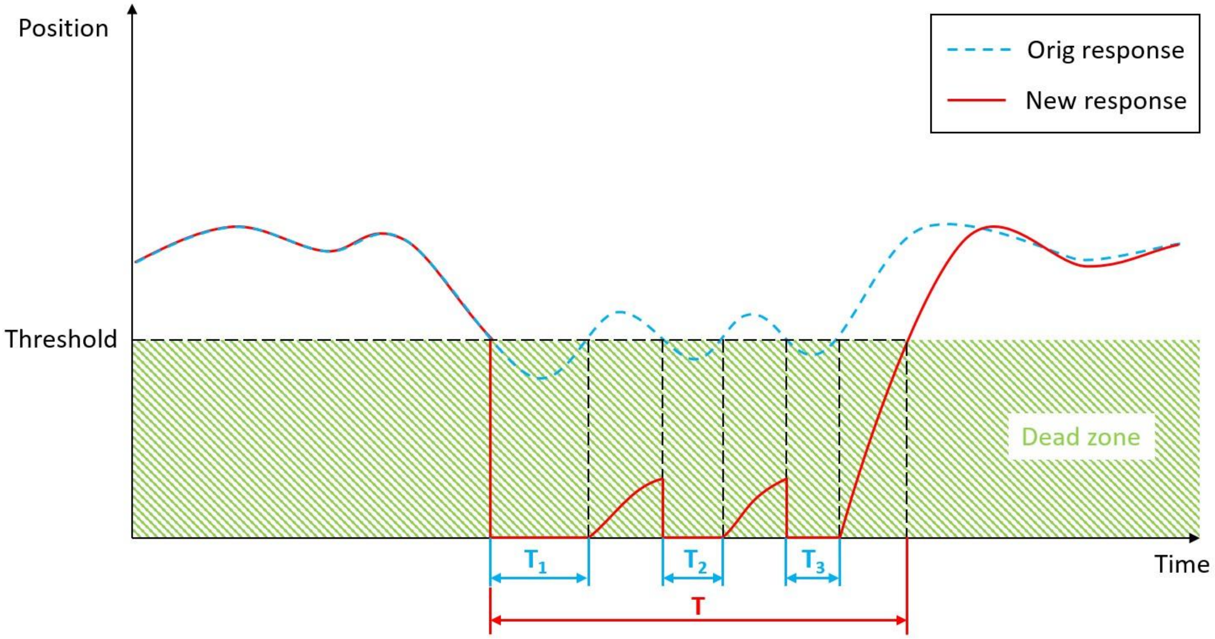


Figure 17. original response vs new response

4.3 Socket communication

The communication between the front-end and the back-end is made possible using **Socket**. Specifically, a **Unity C# server** and a **Python client** are setup to transfer data through a TCP/IP connection. The details of this part of implementation can be referred to SocketServer.cs.

The socket endpoint is set at:

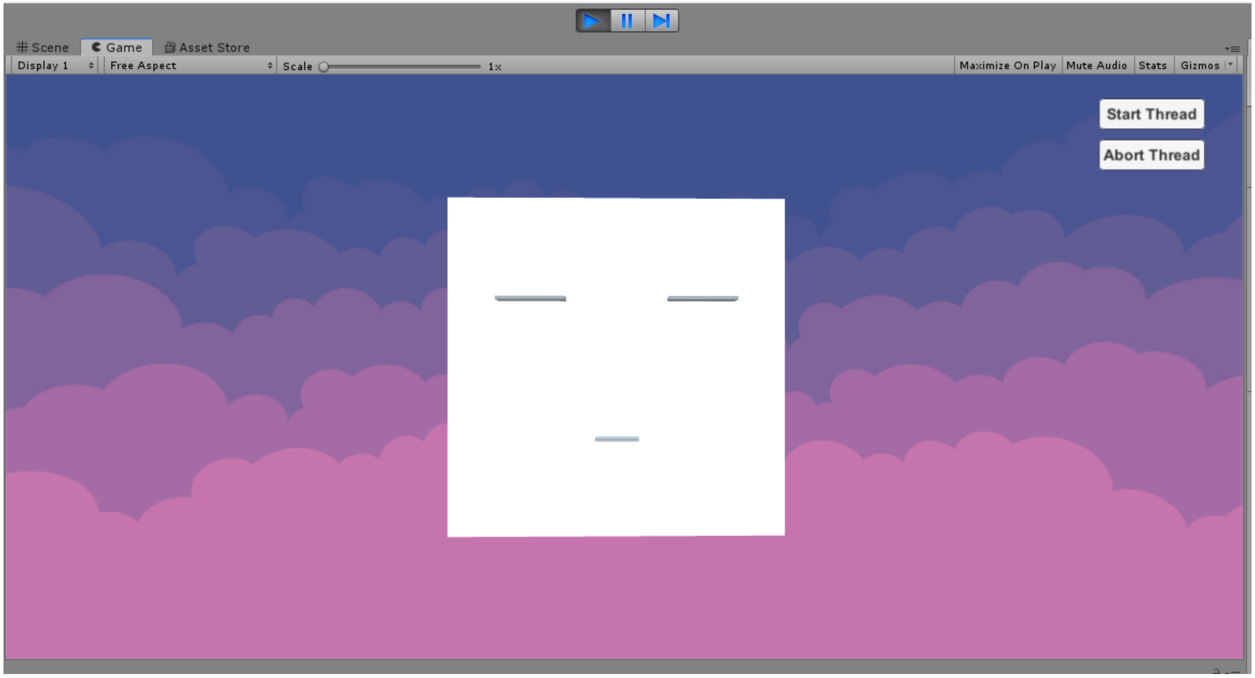
- IP address: 127.0.0.1
- Port: 1755

Chapter Five:

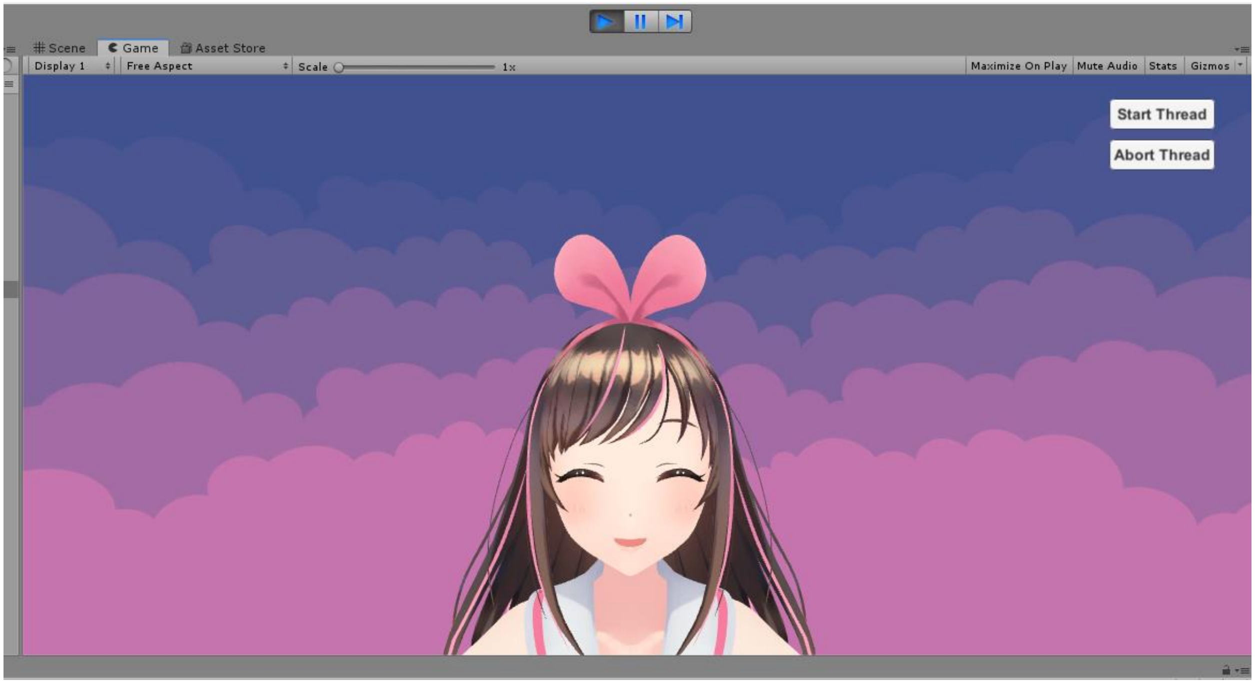
User Manual

5- User Manual

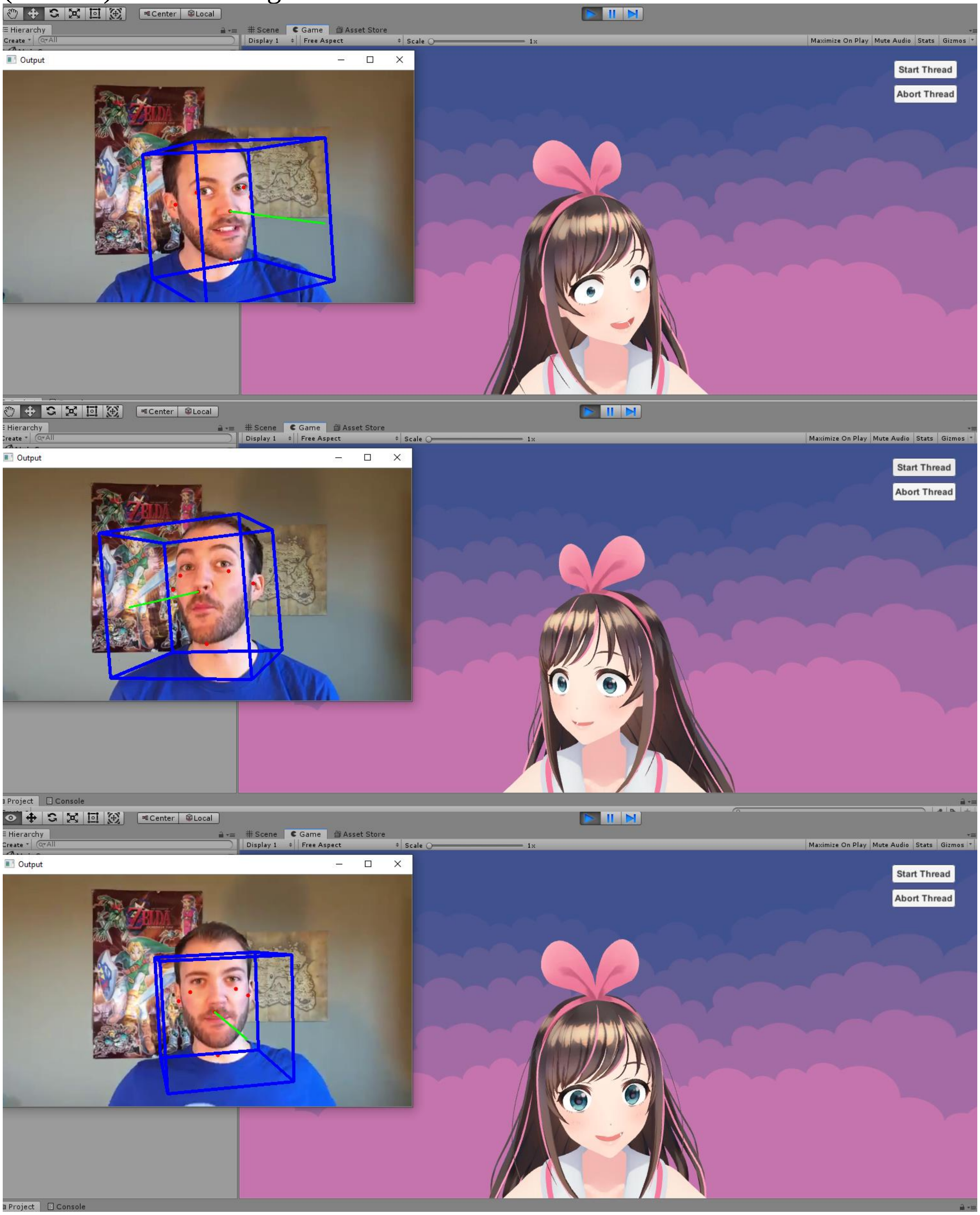
Model1 (Cube)



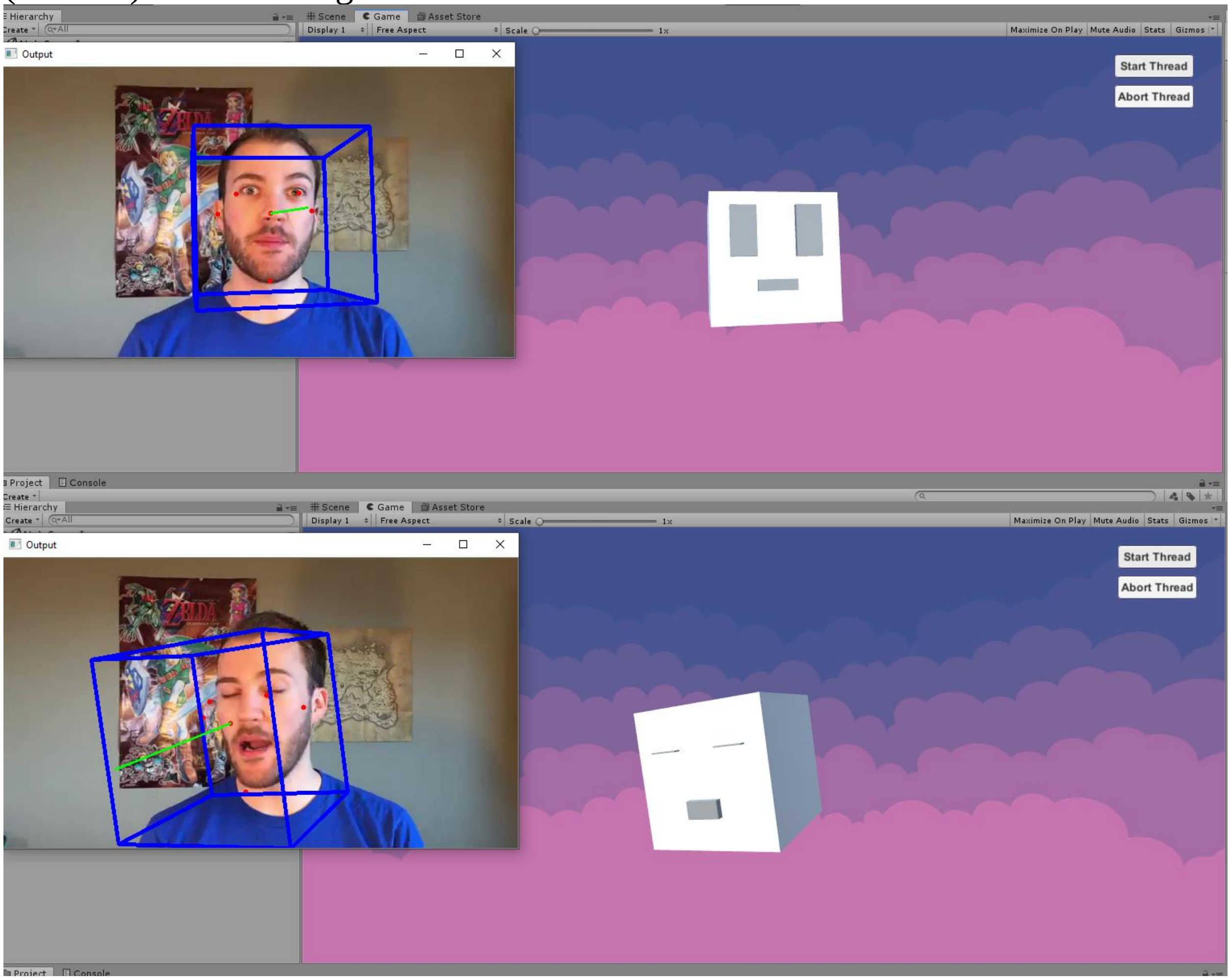
Model2



(Model2) Run folowing the Camera Person

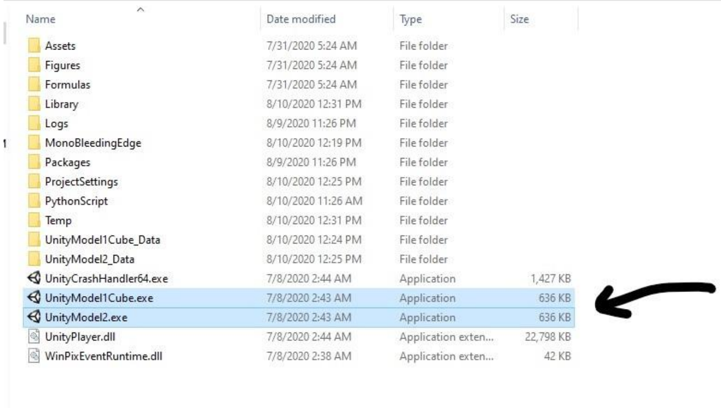


(Model1) Run folowing the Camera Person



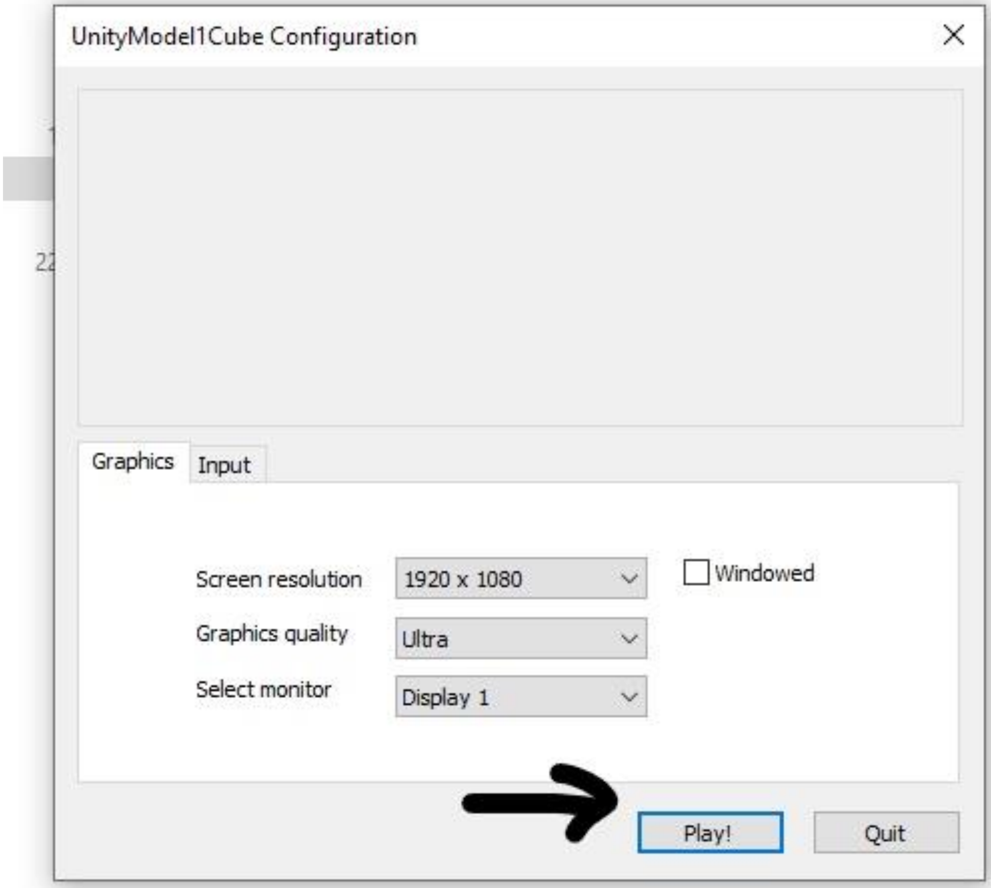
Run Steps:

1- Open the Project Folder.
2- Execute any of the 2 .exe files each one for a different model.



Name	Date modified	Type	Size
Assets	7/31/2020 5:24 AM	File folder	
Figures	7/31/2020 5:24 AM	File folder	
Formulas	7/31/2020 5:24 AM	File folder	
Library	8/10/2020 12:31 PM	File folder	
Logs	8/9/2020 11:26 PM	File folder	
MonoBleedingEdge	8/10/2020 12:19 PM	File folder	
Packages	8/9/2020 11:26 PM	File folder	
ProjectSettings	8/10/2020 12:25 PM	File folder	
PythonScript	8/10/2020 11:26 AM	File folder	
Temp	8/10/2020 12:31 PM	File folder	
UnityModel1Cube_Data	8/10/2020 12:24 PM	File folder	
UnityModel2_Data	8/10/2020 12:25 PM	File folder	
UnityCrashHandler64.exe	7/8/2020 2:44 AM	Application	1,427 KB
UnityModel1Cube.exe	7/8/2020 2:43 AM	Application	636 KB
UnityModel2.exe	7/8/2020 2:43 AM	Application	636 KB
UnityPlayer.dll	7/8/2020 2:44 AM	Application exten...	22,798 KB
WinPixEventRuntime.dll	7/8/2020 2:38 AM	Application exten...	42 KB

3- Click on the play button.



UnityModel1Cube Configuration

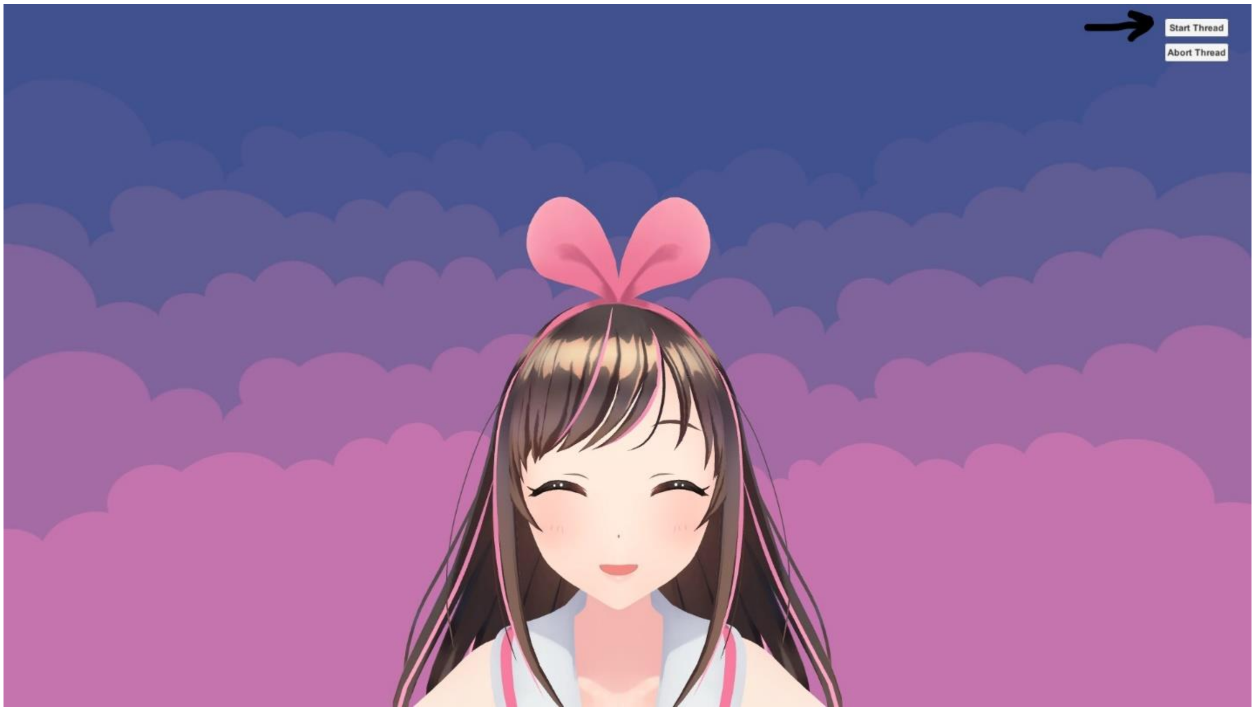
Screen resolution: 1920 x 1080

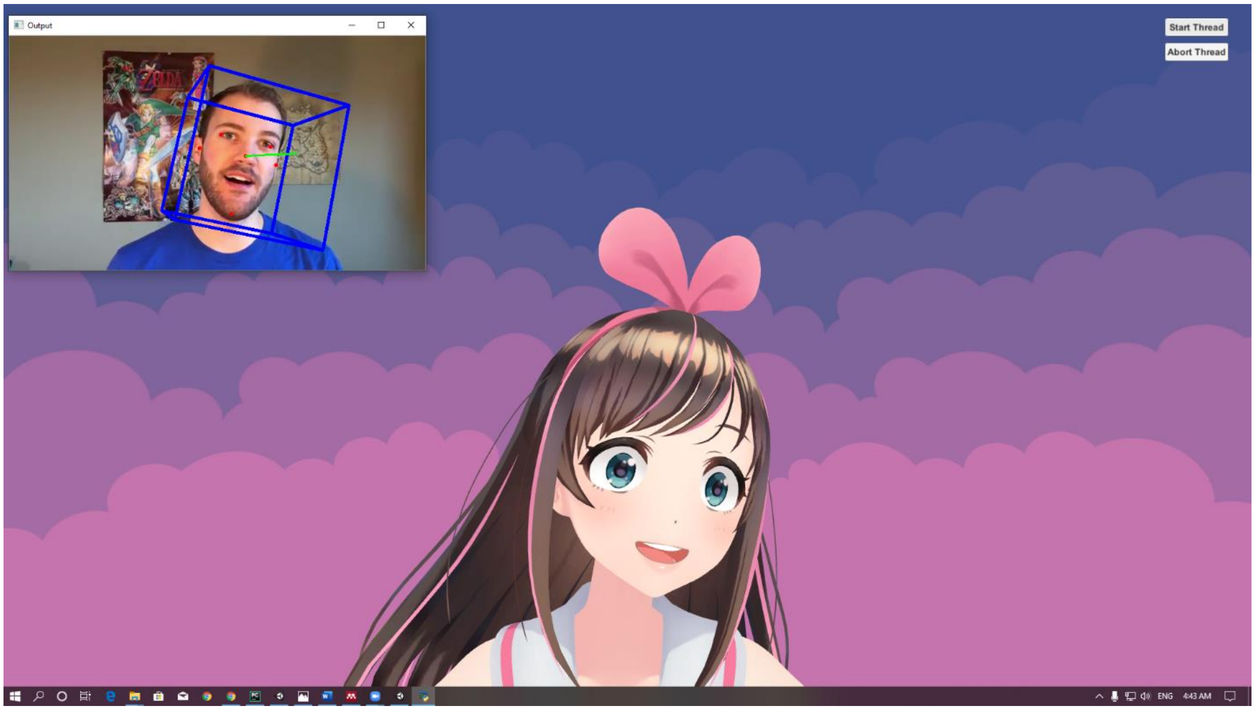
Graphics quality: Ultra

Select monitor: Display 1

□ Windowed

4- Click on the Start Thread button





Installation Guide:

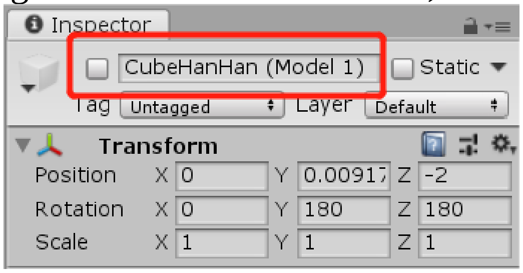
Programs Required: Unity 2018, Python 3.6 environment with Dlib, OpenCV

1- Configure the environment. You may use pip to install the required packages.

   For installing the library from Terminal activate your environment and then type:
   Pip install dlib
   Pip install opencv-python
   Pip install opencv-contrib-python
   Pip install socket
2- Open the folder as a project in Unity.
3- Drag "MainScene" from Assets -> scenes to the Hierarchy and remove the "Untitled" scene.
4- Press the Play button to run the project.
5- Press the "Start Thread" button on the UI to start the C# socket server. An output window should pop up and the Python client would start to communicate with the sever. Now the Python script would be running in the background to extract features from the video stream captured by camera and send them to Unity. The virtual character comes to life!
6- Press the Play button again to stop the program.

Model selection

This system now has two characters model with customized parameter settings. To change the character model, select the corresponding Game Object in the Scene hierarchy and unhide it by clicking on the toggle next to the object's name in the inspector, while the other Game Object should be hide. For example, if you want to change from model 1 to model 2, the settings should look like this:



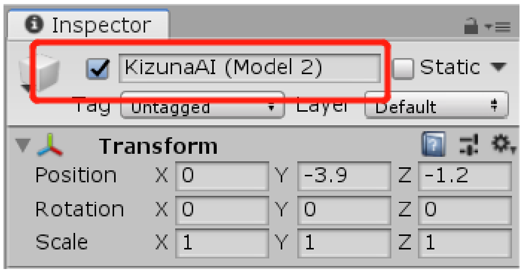
Inspector

☐ CubeHanHan (Model 1) ☐ Static

Tag Untagged Layer Default

Transform

Position	X 0	Y 0.00917	Z -2
Rotation	X 0	Y 180	Z 180
Scale	X 1	Y 1	Z 1



Inspector

☑ KizunaAI (Model 2) ☐ Static

Tag Untagged Layer Default

Transform

Position	X 0	Y -3.9	Z -1.2
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

6- Conclusion and Future Work

6.1 Conclusion

we introduced a computer vision approach for 3D facial performance captured from 2D video. our testing show strong , real-time tracking results that compare to related techniques, our framework significantly advances the state of the art in facial animation by providing a very simple and easy system for users who have no experiences with his kind of applications.

In this work we present the first method capable of capturing facial performances in real-time at high accuracy, Through Combining real-time facial performance tracking with high-fidelity reconstruction opens up a variety of new applications. The proposed method is both robust and accurate since it combines a global face prior to track the overall shape with local regressors to produce the local detail. In doing so we present a number of novel technical contributions, including a generic model that describes local detail on human faces and an automatic technique to extract them such that they can be applied at runtime without interference. Our method has a few limitations which open up for future research. The optical flow employed to improve tracking accuracy of the global model can fail when there is fast motion or motion blur. While any introduced error will be bounded by the global tracker and thus cannot cause the system to diverge, it might lead to a subtle local shift in tracked texture. This could potentially be addressed by relating the current frame back to the neutral expression. Changing illumination and especially hard shadows can also impact the local regressor. If the illumination during runtime differs substantially from the illumination during wrinkle training, the resulting wrinkles can be inaccurate.

6.2 Future Work

This work is A result of our trials within the area of computer animation.

In general terms, the goals of this project were to explore the area of motion capture for facial animation and the use of efficient algorithms for the generation of surface deformations.

The results of this research show some positive outcomes when dealing with the development and use of transferred motion capture in 3D facial models. Also, when making the evaluation, there are some limitations in the qualitative comparison. A more robust perceptual experiment could bring more significant results

there are possibilities to optimize further the algorithms to bring more accurate facial deformation and to be more computational efficient.

there are possibilities to optimize the Filters Used to track the landmarks to get more accurate points, and Filters used to smooth the result of tracking.

we can add new physical engine for other components like hair and headwear

new blinking function may be added

this change will result to improve the performance of the project.

References

[1] “Ping, H.Y., Abdullah, L.N., Sulaiman, P.S. and Halin, A.A., 2013. Computer Facial Animation: A Review. International Journal of Computer Theory and Engineering, 5(4), 658.”

[2] “Lasseter, J., 1987. Principles of traditional animation applied to 3D computer animation. ACM Siggraph Computer Graphics, 21 (4), 35–44.”

[3] “Orvalho, V., Bastos, P., Parke, F.I., Oliveira, B. and Alvarez, X., 2012. A Facial Rigging Survey. Eurographics (STARs), 183–204.”

[4] “Cong, M., Lan, L. and Fedkiw, R., 2017. Muscle simulation for facial animation in Kong: Skull Island. ACM SIGGRAPH 2017 Talks, 21.”

[5] P. Viola and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features,” 2001.

[6] “Parke, F.I., 1972. Computer generated animation of faces. Proceedings of the ACM annual conference, 1, 451–457.”

[7] “Gillenson, M.L., 1974. The interactive generation of facial images on a CRT using a heuristic strategy. Thesis (PhD). The Ohio State University.”

[8] “Platt, S.M., 1980. A system for computer simulation of the human face. University of Pennsylvania.”

[9] “Tony de Peltrie, 1985 [film]. Directed by Pierre Lachapelle et. al. Canada: University of Montreal.”

[10] “Tin Toy, 1988 [film]. Directed by John Lasseter. USA: Disney Pixar.”

[11] “Alexander, O., Rogers, M., Lambeth, W., Chiang, J.Y., Ma, W.C., Wang, C.C. and Debevec, P., 2010. The digital Emily project: Achieving a photorealistic digital actor. IEEE Computer Graphics and Applications, 30(4), 20-31.”

[12] “Ichim, A.E., Kadleček, P., Kavan, L. and Pauly, M., 2017. Phace: Physics-based face modeling and animation. ACM Transactions on Graphics (TOG), 36(4), 153.”

[13] “McLaughlin, T., Cutler, L. and Coleman, D., 2011. Character rigging, deformations, and simulations in film and game production. ACM SIGGRAPH 2011 Courses, 5.”

[14] “Arai, K., Kurihara, T. and Anjyo, K., 1996. Bilinear interpolation for facial expression and metamorphosis in real-time animation. The Visual Computer, 12(3), 105–116.”

[15] “The Lord of the Rings, 2001 [film]. Directed by Peter Jackson. New Zealand-USA: New Line Cinema.”

[16] “Parke, F.I. and Waters, K., 2008. Computer facial animation. New York: AK Peters/CRC Press.”

[17] “Kong: Skull Island, 2017 [film]. Directed by Jordan Vogt-Roberts. USA:

Warner Bros."

[18] "Cong, M., Bhat, K.S. and Fedkiw, R., 2016. Art-directed muscle simulation for high-end facial animation. ACM SIGGRAPH Symposium on Computer Animation, 119–127."

[19] "Murai, A. et al., 2017. Dynamic skin deformation simulation using musculoskeletal model and soft tissue dynamics. Computational Visual Media, 3(1), 49–60."

[20] "Man-Dun, Z., Xin-Jie, G., Shuang, L., Zhi-Dong, X., Li-Hua, Y. and Jian-Jun, Z., 2014. Fast Individual Facial Animation Framework Based on Motion Capture Data. Journal of Donghua University (English Edition), 1(3), 8."

[21] "Hellblade: Senua's Sacrifice, 2017 [game]. Directed by Tameem Antoniades. UK: Ninja Theory."

[22] C. Cao, D. Bradley, K. Zhou, and T. Beeler, "Real-time high-fidelity facial performance capture," *ACM Trans. Graph.*, vol. 34, no. 4, 2015, doi: 10.1145/2766943.

[23] C. Cao, H. Wu, Y. Weng, T. Shao, and K. Zhou, "Real-time facial animation with image-based dynamic avatars," *ACM Trans. Graph.*, vol. 35, no. 4, 2016, doi: 10.1145/2897824.2925873.

[24] C. Cao, Y. Weng, S. Lin, and K. Zhou, "3D shape regression for real-time facial animation," *ACM Trans. Graph.*, vol. 32, no. 4, 2013, doi: 10.1145/2461912.2462012.

[25] "Head Pose Estimation using OpenCV and Dlib | Learn OpenCV." https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/ (accessed Aug. 12, 2020).