Munro, Nicole

My Graduation Work Title

Graduation work 2018-19

Digital Arts and Entertainment

Howest.be

Contents

[2 Abstract 2](#_Toc526348092)

[3 Introduction 3](#_Toc526348093)

[4 Research 4](#_Toc526348094)

[4.1 Facial tracking in real time with c++ and other 3rd party libraries 4](#_Toc526348095)

[4.1.1 The different 3rd party libraries available 4](#_Toc526348096)

[4.1.2 The difference between the libraries 4](#_Toc526348097)

[4.1.3 Which one was chosen and why 4](#_Toc526348098)

[4.1.4 Should we use existing facial tracking software, what is out there 4](#_Toc526348099)

[4.1.5 Is there a way to make our facial tracker faster and more reliable 4](#_Toc526348100)

[4.2 Sending the information from the facial tracker to Unreal 4](#_Toc526348101)

[4.2.1 The different methods 4](#_Toc526348102)

[4.2.2 What we chose 4](#_Toc526348103)

[4.2.3 Why we chose it 4](#_Toc526348104)

[4.3 Real time mesh manipulation 4](#_Toc526348105)

[5 case study 5](#_Toc526348106)

[5.1 introduction 5](#_Toc526348107)

[5.2 Tracking a face using OpenCV 5](#_Toc526348108)

[5.2.1 Installation 5](#_Toc526348109)

[5.2.2 Tracking a face 5](#_Toc526348110)

[5.2.3 Facial landmarks 5](#_Toc526348111)

[5.3 Getting the information into Unreal 6](#_Toc526348112)

[1. Shading 7](#_Toc526348113)

[2. Lighting 7](#_Toc526348114)

[6 Conclusion 9](#_Toc526348115)

[7 References 10](#_Toc526348116)

[8 Appendices 11](#_Toc526348117)

# Abstract

**An abstract explains the aim of the paper in very brief, (the methods, results, etc.). Maximum length 150 words**

This paper will explain the topic of real-time facial feature detection and application to a rigged facial model. A single camera will be used to track the face and third-party libraries will be implemented to find these facial features. A facial tracker app will be built that smoothly tracks the face and it’s features, this tracker will write the values of the tracked points to a file that will be read by an engine, real-time, and then those tracked points will be applied to bones and blend-weights in the rig.

# Introduction

**In the introduction, you write the background of your topic, explain the purpose of the paper more broadly, and explain the hypothesis, and the research question(s).**

Real-time facial feature and detection has become more broadly used for video game animations. This is the case because video games and movies want to be more realistic and that can be achieved by capturing real human expressions. Capturing these facial animations real-time is also important because we no longer want to wait for an animation to render out to see if it’s good enough for the game.  
This paper will go over the steps taken to track a face and it’s features real time and applying those tracked features to a facial rig in real-time.  
This paper will also explain the details of the libraries we will be using for facial feature tracking, what library we ended up going with and why. We will also discuss the different ways to communicate the tracked points from the facial tracker to Unreal engine. Then how we chose to manipulate the facial rig.

Our first research question would be how we could go about tracking a face and it’s features. Once we have determined what 3rd party libraries exist for facial tracking, we can do more research to determine which one to use and why. Research will also be done on what happens behind the scenes with these different libraries.

Once we have the knowledge of what libraries we need to use then we can go about researching how we can take the information from the facial tracker application to the engine we are using.

Then we are going to take the information from the facial tracker app and apply it to the bones in the face of the rig. Here we must research what the best way is to manipulate a rig in real time.

# Research

## Facial tracking in real time with c++ and other 3rd party libraries

### The different 3rd party libraries available

When it comes to real time face tracking and face recognition many libraries and existing software bases their programs on two c++ toolkits. These two are OpenCV and Dlib, and they are either used in conjunction or separately.

There are also face tracking applications available that use OpenCV and Dlib that work out of the box. Examples of the ones found were OpenFace, FaceTracker and OpenFaceTracker. These were all free to use accurate face tracker applications.

In this part of the paper, the different libraries will be discussed, and, in the end, we will conclude which library was chosen and why.

### The difference between the libraries

#### OpenCV

OpenCV (Open Source Computer Vision Library) has a focus in real time applications and is used frequently for facial tracking and facial feature tracking. Most facial tracking applications that are found online use OpenCV.

OpenCV uses the Haarcascade Classfier which is a machine learning algorithm, and a cascade function is trained from images. The algorithm first takes a lot of images with a face then a lot of images without a face to train itself. Once it can recognize a face we can move on to recognizing features. The way it works is that each feature is a value obtained by subtracting the sum of pixels under the white rectangle from the sum of pixels under the black rectangle. These are called kernels.

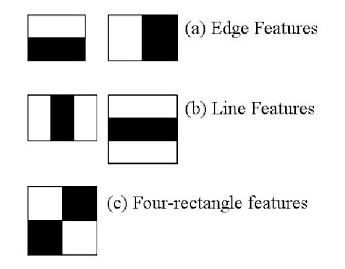
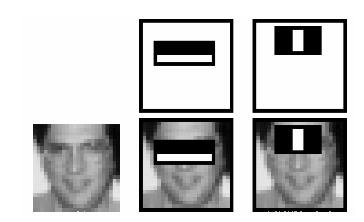


Figure : Line and edge feature detection

Figure The different kinds of detection

Now there is the issue that we need all possible sizes and locations of the kernels to detect the features of the face, which is a lot of computation even for a 24x24 image we have 160 000 features. Thus, the solution was to come up withan Integral image, which reduces the operations of each pixel to just 4 pixels.

There is another issue with this system and that is that for most of an image, the data does not contain facial data. So, we discard any data that is not a face and we only apply the feature checking on parts of the image that contain a face.

Enter the concept of Cascade of Classifiers: the features of the face are grouped into different stages and are applied. So, there are 1, 10, 15 and 50 features in the first 5 stages. If on the first stage no features are detected the image has failed and it will not move on to the next stage.

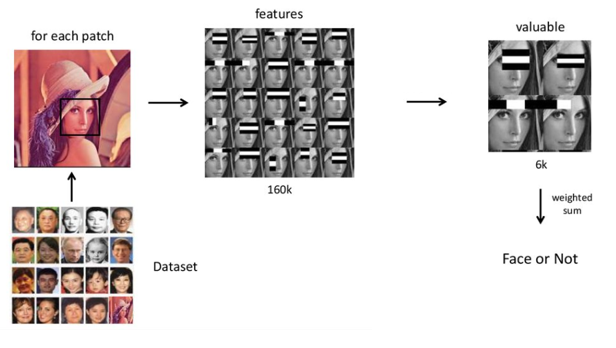
****

Figure The workflow of the Haarcascade algorithm

This algorithm is fast, running at ~100ms on the CPU. But it is very inaccurate and creates many false positives.

#### Dlib

Dlib uses machine learning algorithms and it’s face detector is based on histogram of oriented gradients (HOG) and linear SVM. With HOG the intensity of gradients or edge directions can describe the local object appearance. A histogram of gradient directions can be complied by dividing the image into small connected regions called cells and the pixels in the cells would be used to calculate this histogram.

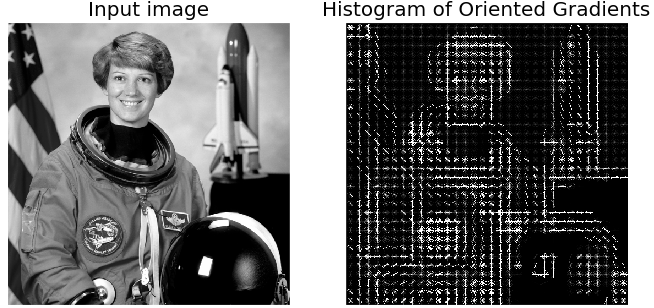


Figure HOG algorithm for edge detection

The concept of linear SVM (support vector machine) is that it is given a set of training examples, and the training algorithm assigns new examples to one category or the other, thus making it a non-probabilistic binary linear classifier.

### Which one was chosen and why

Ultimately, the facial tracking application would be built using OpenCV.

OpenCV has extensive documentation online, with an active community of people using the library to create software applications.

The reason why it was chosen was because there is a very active OpenCV community with people posting on forums and using chat services to talk to each other about issues. OpenCV also has functionality to easily use the GPU instead of the CPU for landmark detections, which is inherently faster.

## Sending the information from the facial tracker to our engine

To start manipulating a mesh real time, the data from the Face Tracking application should be sent to the engine being used.

### The different methods

#### Creating the Face Tracker in the chosen engine

Instead of creating the Face Tracker application in c++ in a separate Visual Studio solution, the engine that being used could be linked to OpenCV and the landmark detection could be built into it.

#### Sending the tracked data to the engine via reading and writing to a file every frame

The Face Tracker application could write the tracked data in a logical manner to a .csv file every frame, then the engine would read from that file and parse the data.

This is an issue for several reasons, one being how slow it is to read and write to a file. Every tick the Face Tracker application would have to open the file and re-write the data, and the engine of choice would read that data and parse it – and computationally this can be incredibly slow.

Another issue with reading and writing to the same file every frame is I/O contention, this causes degradation of performance. I/O contention is created when a disk competes for

When reading and writing to a hard drive every single frame I/O contention is created. I/O contention is what happens when the hard drive on your computer is competing for I/O resources.

### What we chose

### Why we chose it

## Real time mesh manipulation

# case study

## introduction

In this case study I will go over the steps I took to make the whole real-time facial animation process work.

## Tracking a face using OpenCV

### Installation

The first thing I had to do was download OpenCV from Github and compile the libraries with CMake. Once we had the complied libraries we could link them to Visual Studio and start building an application using the library’s functionality.

### Tracking a face

Once we were done with the installation process we could move on to opening the webcam and reading information from it. The information I found on the internet for reading from a webcam used the GUI functionality from Visual Studio which was incompatible with the version of Visual Studio that I had and OpenCV.

Once I found a workaround for this I started with face tracking and eye tracking, when that started working I could move on to landmarks

### Facial landmarks

This is where the real fun begins, to implement facial landmarks, we had to download OpenCV Contributions then link the libraries of the original OpenCV and OpenCV\_contrib. This proved to be a challenge as there is no real documentation on how to do this besides the brief instructions on the OpenCV\_contrib Github.

We had to take the original source code for OpenCV and use CMake to build the libraries with OpenCV contrib. Once those libraries were built (all of them had to be the same version) we could then link them to Visual Studio.

The way you link the libraries is also important because if they are linked incorrectly you cannot use the functionality from OpenCV contributions. I ran into some weird issues that I eventually had to go to an OpenCV forum to ask for help – the libraries that I linked were different versions and that caused some issues.

Once that issue was resolved we had facial landmark detection running at a reasonable fps and with pretty good accuracy.



## Getting the information into Unreal

## Shading

## Lighting

# Conclusion

**repeat the main topics, discuss your main findings, discuss the end result.**

# References

Casey Raes (2014), Processing (second edition).

Saccade. (n.d.). In Wikpedia. Retrieved November 6 2016 from <https://en.wikipedia.org/wiki/Saccade>

Sarah Northway (2016) A year in VR Northway [Powerpoint slides] from <https://www.gdcvault.com/play/1024631/A-Year-in-VR-A>

# Appendices