Munro, Nicole

Real Time Face Tracking

Graduation work 2018-19

Digital Arts and Entertainment

Howest.be

Contents

[2 Abstract 2](#_Toc528597435)

[3 Introduction 3](#_Toc528597436)

[4 Research 4](#_Toc528597437)

[4.1 Facial tracking in real time with c++ and other 3rd party libraries 4](#_Toc528597438)

[4.1.1 The different 3rd party libraries available 4](#_Toc528597439)

[4.1.2 The difference between the libraries 4](#_Toc528597440)

[4.1.3 Which one was chosen and why 6](#_Toc528597441)

[4.2 Sending the information from the facial tracker to our engine 6](#_Toc528597442)

[4.2.1 The different methods 6](#_Toc528597443)

[4.2.2 Which one was chosen and why 7](#_Toc528597444)

[4.3 Real time mesh manipulation 7](#_Toc528597445)

[4.3.1 Blend shapes 7](#_Toc528597446)

[5 Case study 8](#_Toc528597447)

[5.1 introduction 8](#_Toc528597448)

[5.2 Tracking a face using OpenCV 8](#_Toc528597449)

[5.2.1 Installation 8](#_Toc528597450)

[5.2.2 Tracking a face 8](#_Toc528597451)

[5.2.3 Facial landmarks 8](#_Toc528597452)

[5.3 Sending the information into Unreal 9](#_Toc528597453)

[5.3.1 By reading and writing to a file 9](#_Toc528597454)

[5.3.2 Creating a DLL 9](#_Toc528597455)

[5.4 Building a system to manipulate the face rig 9](#_Toc528597456)

[6 Conclusion 9](#_Toc528597457)

[7 References 10](#_Toc528597458)

[8 Appendices 11](#_Toc528597459)

# Abstract

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

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.

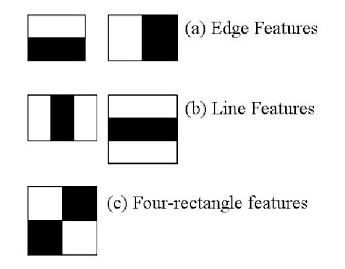
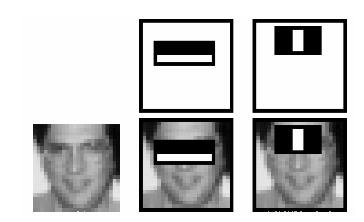


Figure 1: Line and edge feature detection

Figure 2 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.

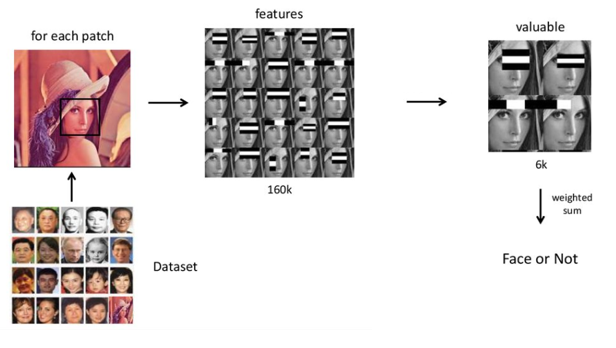
****

Figure 3 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.

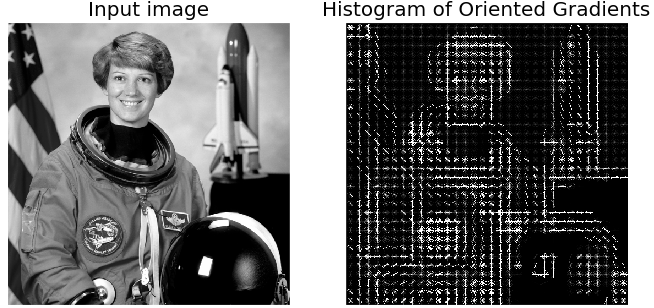


Figure 4 HOG algorithm for edge detection

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 there are high amounts of read and write operations happening. Contention describes two or more devices competing for the same resources.

#### Creating a DLL that links the Face Tracker application to the chosen engine

With this method the Face Tracker application would be a DLL that the engine loads up at the start of the game running, and the engine would be able to read the data in real time from the Face Tracker without any latency.

A dynamic link library (DLL) is used when a larger program needs to run smaller tasks at the same time. DLLs cannot be run on their own, but rather are called upon by other code that is already running. The advantage of using DLLs is that space is saved in RAM, because the files don’t need to be loaded together with the main program.

Creating a DLL also promotes modular architecture, with the DLL being separate it can easily be changed or rectified with no changes being made to the main program.

When creating functions (in the Face Tracker) that can be called by our chosen engine, these functions must be written in c. Thus, the Face Tracking system would be separated from the DLL functions that our engine calls because the Face Tracking system is written in c++.

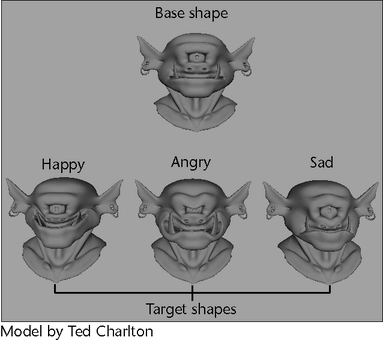
### Which one was chosen and why

In the case study both the options of reading and wring to a file, and creating a DLL are explored. In the end creating a DLL was chosen as it ran faster than the file reading/writing and it made the most sense for this project.

Building a DLL meant that we could easily change and manipulate the Face Tracking software without changing the code of the actual project.

## Real time mesh manipulation

### Blend shapes

In facial rigging and animation, animators can use controllers to create blend weights for a rig. Blend weights allow animators to easily create facial expressions – and gives them the ability to blend between them.

Blend weights of a facial rig can be exported to Unreal engine (called morph targets in Unreal) and can be manipulated in real time, for the purposes of this project the morph targets will be manipulated by the data that is given by the Face Tracker.

# Case study

## introduction

This part of the paper will discuss the steps taken and the entire pipeline of Real Time Facial Tracking and animations.

## Tracking a face using OpenCV

### Installation

To track a face the first step is to fork the OpenCV repository and its contributions, because it is within the contributions that the facial landmark detection algorithms exist. Instructions for building the binaries for OpenCV and its contributions are on the Github page of the contributions. [[4]](#_References)

The readme suggests downloading CMake to build the libraries. CMake is a tool designed to build software and control the software compilation process. [5]

Using CMake made it easy and manageable to build the libraries of OpenCV and therefore linking OpenCV and its libraries to Visual Studio wasn’t too difficult.

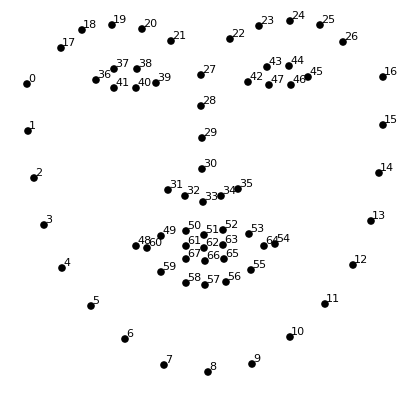
### Tracking a face

Once the installation process is done, the next step is to open and read the data from a webcam then use that camera data to detect if a face is in the screen. To do this, OpenCV uses it’s Haarcascade classifiers to determine if there is a face on the screen and if so, where it is.

### Facial landmarks

Once the face is tracked, the next step is to track the facial landmarks on the face itself. OpenCV has functionality for this, and it returns 68 tracked data points. [8]

With these tracked landmarks working, the next stage of the face tracking pipeline is to communicate the tracked points to the engine.

****

## Sending the information into Unreal

When it comes to reading the tracked data in Unreal, there are several methods that were found in the research stage of this project. Two of the methods were tested in the case study.

### By reading and writing to a file

The easiest method of sending the data to Unreal would be to write the information tracked to a csv file then to read that file every frame from Unreal.

The CSV file layout was simple: x,y(endline). By keeping the layout simple it made the data easier to parse in Unreal. Once the Face Tracker was writing the data to a file, we could read the data with Unreal and c++ [9].

The code for this was simple and it produced results quickly.

#### The issues

The game was running at 5FPS and the framerate kept dropping, because of reading and writing to a file.

Another issue was that the Face Tracker had to be started manually then the Unreal game would have to be started manually too. This was cumbersome and not ideal – starting the Face Tracker .exe from Unreal was also an option but it was not further researched. [10]

### Creating a DLL

## Building a system to manipulate the face rig

# Conclusion

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

# References

1. SearchStorage. 2018. What is I/O contention (input/output contention)? - Definition from WhatIs.com. [ONLINE] Available at: <https://searchstorage.techtarget.com/definition/I-O-contention-input-output-contention>. [Accessed 10 October 2018].
2. Lifewire. 2018. What Is a DLL File? (Dynamic Link Library). [ONLINE] Available at: <https://www.lifewire.com/what-is-a-dll-file-2625852>. [Accessed 11 October 2018].
3. tutorialspoint.com. 2018. DLL How to Write. [ONLINE] Available at: <https://www.tutorialspoint.com/dll/dll_writing.htm>. [Accessed 11 October 2018].
4. Paint blend shape weights | Maya 2018 | Autodesk Knowledge Network. 2018. Paint blend shape weights | Maya 2018 | Autodesk Knowledge Network. [ONLINE] Available at: <https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2018/ENU/Maya-CharacterAnimation/files/GUID-950716A4-19CF-4C14-96F1-FD6B4D48663C-htm.html>. [Accessed 15 October 2018].
5. Overview | CMake. 2018. Overview | CMake. [ONLINE] Available at: <https://cmake.org/overview/>. [Accessed 29 October 2018].
6. GitHub. 2018. GitHub - opencv/opencv\_contrib: Repository for OpenCV's extra modules. [ONLINE] Available at: <https://github.com/opencv/opencv_contrib>. [Accessed 03 October 2018].
7. OpenCV: Deep Neural Network module. 2018. OpenCV: Deep Neural Network module. [ONLINE] Available at: <https://docs.opencv.org/3.4.3/d6/d0f/group__dnn.html>. [Accessed 01 October 2018].
8. Facial Landmark Detection | Learn OpenCV. 2018. Facial Landmark Detection | Learn OpenCV. [ONLINE] Available at: <https://www.learnopencv.com/facial-landmark-detection/>. [Accessed 01 October 2018].
9. Unreal Engine. 2018. File Management, Create Folders, Delete Files, and More - Epic Wiki. [ONLINE] Available at: <https://wiki.unrealengine.com/File_Management,_Create_Folders,_Delete_Files,_and_More#Create_Text_Files>. [Accessed 16 October 2018].
10. api.unrealengine.com. 2018. No page title. [ONLINE] Available at: <https://api.unrealengine.com/INT/API/Runtime/Core/GenericPlatform/FGenericPlatformProcess/CreateProc/index.html>. [Accessed 29 October 2018].

# Appendices