

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

DETECTION OF FACIAL FEATURES FROM 3D FACE MODEL ACQUIRED BY THE KINECT SENSOR

BAKALÁŘSKÁ PRÁCE

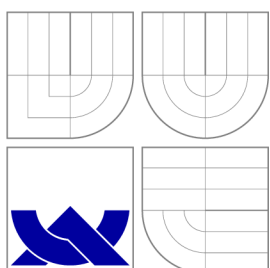
BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

MICHAL DANKO

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE RYSŮ OBLIČEJE V 3D SNÍMKU OBLIČEJE POMOCÍ SENZORU KINECT

DETECTION OF FACIAL FEATURES FROM 3D FACE MODEL ACQUIRED BY THE KINECT

SENSOR

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

MICHAL DANKO

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. ŠTĚPÁN MRÁČEK

BRNO 2013

Abstrakt

Obsahem této bakalářské práce je zkoumání a navrhnutí metody pro detekci rysů tváře (nosu, očí a úst). Detekce probíhá na 3D modelech získaných pomocí přístroje Kinect. Kromě návrhu a implementace aplikace jsou v dokumentě zahrnuty i dosažené výsledky experimentů detekce na různých vzorcích a jejich vyhodnocení.

Abstract

The subject of this bachelor thesis is study and design of facial features detection (nose, eyes and mouth). The detection is applied on 3D models acquired by Kinect device. Besides the design and implementation of application, this document also includes experimenting with the application on the set of various models and evaluation of the results.

Klíčová slova

Kinect, detekce, rysy tváře, 3D, C++, OpenCV

Keywords

Kinect, detection, facial features, 3D, C++, OpenCV

Citace

Michal Danko: Detection of Facial Features From 3D Face Model Acquired by the Kinect Sensor, bakalářská práce, Brno, FIT VUT v Brně, 2013

Detection of Facial Features From 3D Face Model Acquired by the Kinect Sensor

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Štěpána Mráčka.

.....

Michal Danko

May 16, 2014

Poděkování

Ďekuji svému vedoucímu práce za poskytnutí odborné pomoci a lidem, kteří mně svolili zachytit je jako 3D modely: mé rodině, Timotejovi Gabrišovi a Matúšovi Blahovi.

© Michal Danko, 2013.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Contents

1	Introduction	2
2	The basics of 3D data	3
2.1	Acquisition of 3D data	3
2.2	Representation of 3D data	5
2.3	Processing of 3D data	6
3	Kinect	10
3.1	Hardware	11
3.2	Software	12
4	Curvatures and surface	13
4.1	Mean and Gaussian curvature	14
4.2	Surface types	15
5	Design of application	16
5.1	Data processing	16
5.2	Specification of surface types	16
5.3	Detection of specific features	17
5.4	Special cases	20
6	Implementation	25
6.1	Used technology	25
6.2	Facial features detecton implementation	25
6.3	User interface	27
7	Experiments	28
7.1	Testing detected features	28
7.2	Testing special features	33
8	Conclusion	34
A	CD contents	37

Chapter 1

Introduction

People will always look for something that can make their life easier and better. One of the things that can improve everyday life in many ways is biometrics [1]. It refers to identification of humans by their both physical and behavioral characteristics.

In 21st century, demand for products of biometrics is rising and it has a great use in different industries. Mostly in medicine, science, security, law department, engineering, entertainment industry, etc. More specifically, it includes everything from touchscreen on smart phone, through eye-detection sensor to robotics. Every year, biometrics plays a bigger role in these departments. There are new methods, more effective and more complex devices being invented.

One of the things that biometrics relates to is face recognition and facial features detection. It is used in video surveillance systems, human computer interfaces and image database management. Mostly used devices are especially various types of cameras, e.g. video camera, web camera, Kinect, etc. They work with data which can be either 2D (photos) or 3D models. Nowadays, these 3D devices are affordable even for common users. The simpler ones doesn't cost much more than a few hundreds of dollars.

Theoretical part is divided into three chapters: The second chapter explains the basics of the representation, acquisition and procession of 3D data. Introductory to both hardware and software part of Kinect, the device used to acquire data set, is the subject of the chapter three. The chapter four consists of principles of curvatures and their types, surface types determination used in selected method for detection. The main design of application is in the fourth chapter. Including explanation of used methods for model data processing and facial features detection. The implementation of used methods and testing itself is described in the chapter number five. All discussed matter, results of experiments and ideas for improvement are summarized in the conclusion.

Chapter 2

The basics of 3D data

Nowadays, 3D is a widely used term. Most people know it in relation with entertainment industry: 3D movies, special visual effects, video games, animations. However, what does it mean?

Generally, 3D describes objects in a three-axis coordinate system. In 2D, each axis represents height and width. And in addition to these two, 3D includes a third dimension, depth.

Although, this concept is also related to computer graphics. The third dimension allows:

- As it was said before, the depth of object, in terms of size.
- The location of an object related to the initial coordinates (commonly $[0,0,0]$).
- The distance of an object from the rendered camera.
- The rotation in z-axis.

Compared to real world, 3D model can be expressed only mathematically, by a modelling. Therefore, in digital space, any 3D object is represented by a 3D model.

2.1 Acquisition of 3D data

There are several ways how to acquire 3D data:

- Acquisition from 2D data - methods based on images, e.g. stereo photogrammetry [2].
- Acquisition from scanners - special scanning devices, e.g. structured light scanner [3].
- Acquisition of combination of previous methods.

Structured light scanner

Structured light scanner is a 3D scanning device consisting of projector and a camera system. It projects light patterns on the target, which illuminates its 3D surface. The line of illumination created by projection is distorted in different perspectives. This distortion can be used to recreate the exact original 3D surface, therefore it is captured by the scanners' camera system.

This method uses patterns, which consist of numerous narrow bands of light. It allows to acquire multiple samples at same time. There are two most frequently used methods of

light projection. Infrared light, which main advantage is its non-disturbing effect. However, the segmentation of the image and the identification of nearby stripes is more difficult. The other method is using a visible light color camera [3].

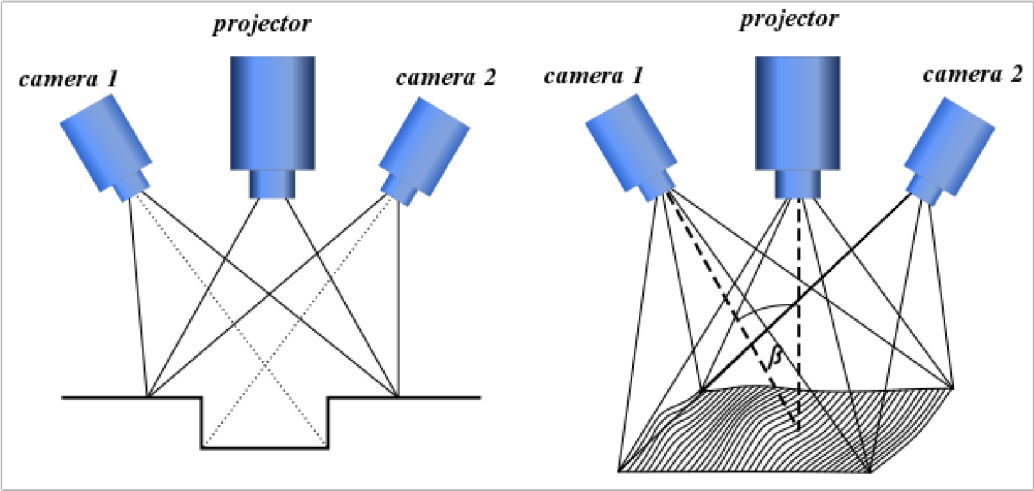


Figure 2.1: Structured light scanner, projector and 2 cameras for avoiding obstructions [3]

2.2 Representation of 3D data

3D data can be represented by set of points in coordinate 3D system. This set is called point cloud and consists of triplet of points, each representing one coordinate ($[x,y,z]$ coordinates). However, this representation is very simple - point cloud data files consist only from rows with coordinates and the final model is just a set of dots. If there is not enough points, it can be unclear, which object is that model representing. Therefore, these points are connected into greater shapes and polygonal or triangle meshes.

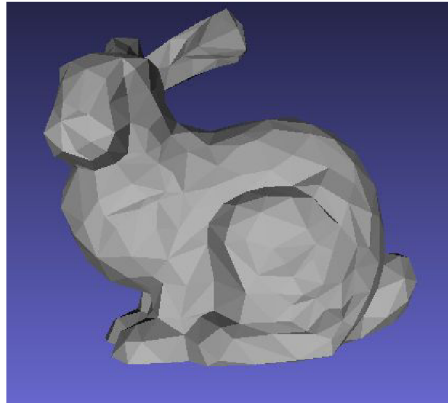


Figure 2.2: 3D model of a bunny

Objects represented by mesh contain different types of mesh elements. The most basic are vertices, points, which are used as corners. Then there are edges, created by connection of two vertices. After that, sets of edges form faces. Finally, polygonal mesh is consisting of coplanar sets of faces. If mesh consists of three edges, it's called a triangle mesh and it's equivalent to face [4].

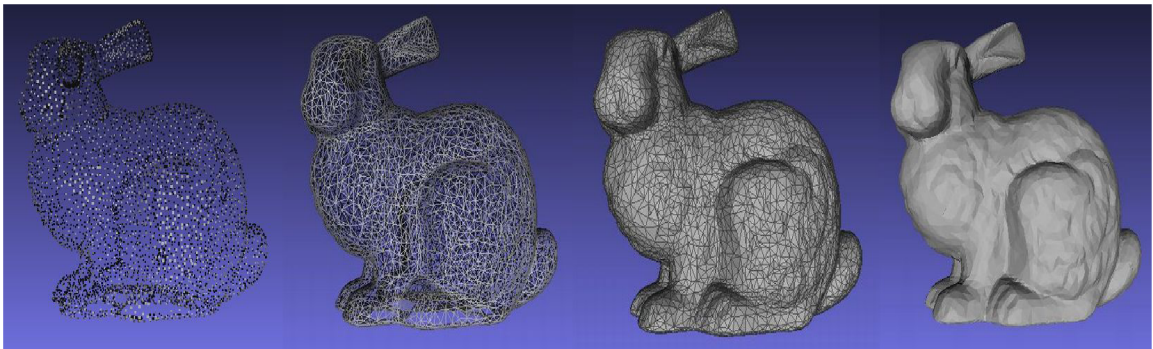


Figure 2.3: Polygon mesh elements, from the left: vertices, edges, faces and polygon model

2.3 Processing of 3D data

Due to inaccurate devices, acquired data are imperfect, often with holes, missing textures and noise. Therefore, they need to be processed, edited to improve their quality.

This can be done by creating an application using graphic libraries, e.g. OpenGL, Direct3D. Or it can be done by using one of many existing 3D model editors and application, whom many of them are free or open source. They offer modelling, texturing, rendering and far more ways of processing.

MeshLaB

MeshLaB, is an open source designed for visualisation and editing 3D data, available for Linux, Windows, MacOSX and mobiles with iOS and Android. MeshLab is actively developed by the a small group of people at the Visual Computing Lab at the ISTI - CNR institute, a large group of university students and many developers from the rest of the world [5].

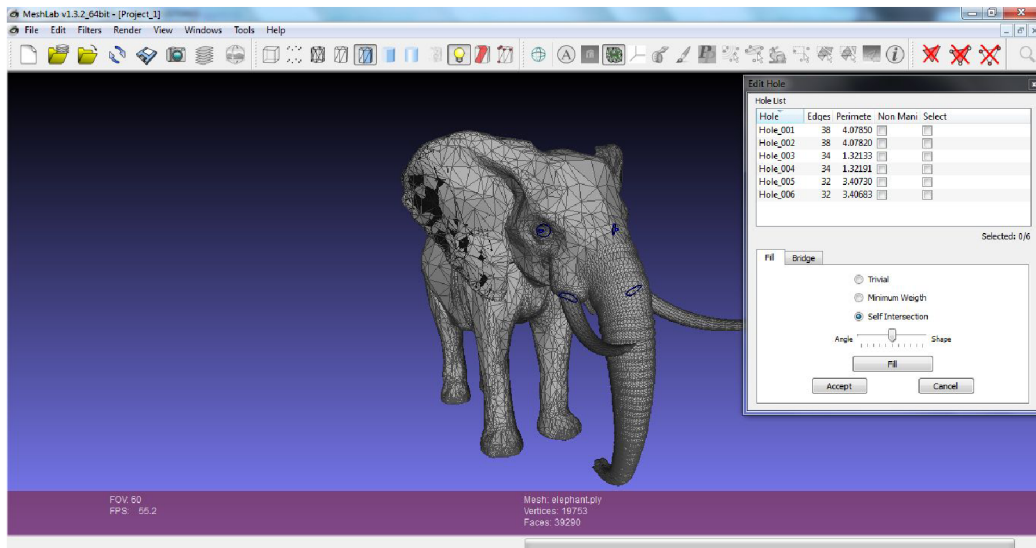


Figure 2.4: MeshLab sample

Colour space

Also, 3D models are often transformed into 2D data, so that important information can be extracted and subsequently used. Then they can be easily processed as images. Images are represented in colour space. It is a model, which describes how are basic colours represented by values. The combination of these values gives a specific colour.

The most common color space is RGB, where R stands for red, G for green and B for blue colour. This colour model is additive, which means that with zero values colour is black and with adding colour faction brightens.

Another colour space is CMYK, which consists of four colour values: cyan, magenta, yellow and black (key). These values refer to four printing inks. This colour space is

subtractive, because the result subtracts, absorbs the some wavelengths of light. There are many different colour spaces as YIQ, HSV, HSL, etc. [6].

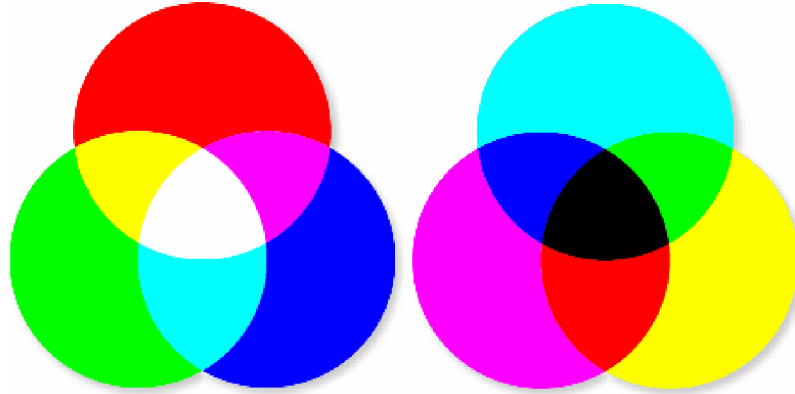


Figure 2.5: Colour spaces: RGB on the left, CMYK on the right [6]

Then there is a grayscale color space, which consists only from shades of gray, or intensity of black and white color. These images are also known as monochromatic. Because every value of pixel of this color space carries only intensity information, it can be used to represent the z-coordinate of 3D model.

Therefore, this grayscale image can be used as a depth map of 3D model. Depth map is an image, which carries information about distance of surface from the observing point. It is also known as Z-buffer, depth buffer and Z-depth. In this particular case, depth map can be helpful in determining maximal and minimal depth values of pixels on the face. Which helps to find approximate areas on the face, which facial features actually are in.

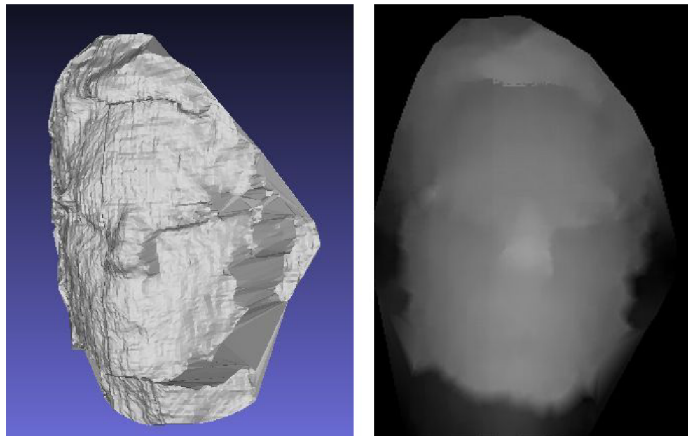


Figure 2.6: Example of depth map (right) created from 3D face model (left)

As said before, in order to reduce unnecessary noise and enhance their quality, data sets are often modified. Models captured by Kinect usually contain many inaccuracies, which need to be removed before further processing. Modification can be done by smoothing, which means applying an approximation function on the data set. As a result, it will separate the noise from points of the data.

There are many smoothing algorithms [7, 8, 9]. When result of smoothing can be expressed as linear transformation to original values, the process is called linear smoothing.

This operation involves applying a matrix which represents the transformation and it is known as convolution, therefore this matrix is a convolution matrix (kernel).

One of the simplest smoothing algorithms is a kernel smoother. There is defined a smoothing window around round every point of the data set. Then there is estimated kernel from values within the smoothing window computed by a particular function.

Gaussian blur

A common example of kernel smoother is a Gaussian blur (smoothing). As the expression says, this smoothing blurs the image using a Gaussian function. This function is used for computing transformation applied to pixels and is defined:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/2\sigma^2} \quad (2.1)$$

where x (μ) is the distance in the horizontal (vertical) axis, σ is the standard deviation of the Gaussian distribution [10].

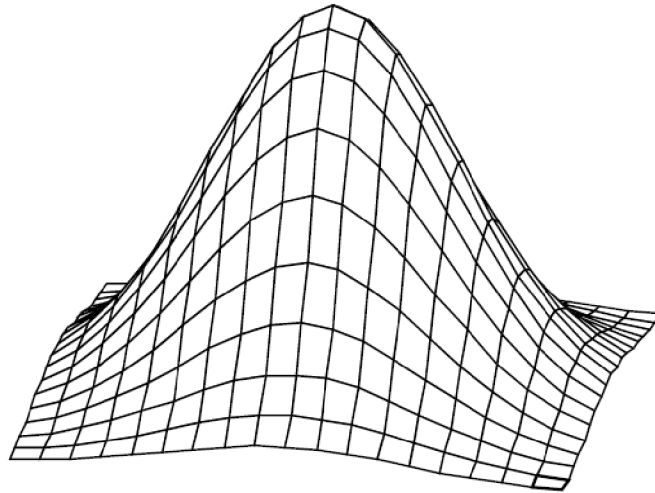


Figure 2.7: Gaussian function in 3D

From Figure 2.8 is it obvious, that the rate of blur is related to the size of σ . The bigger the σ value is, the blurrier the result will be.

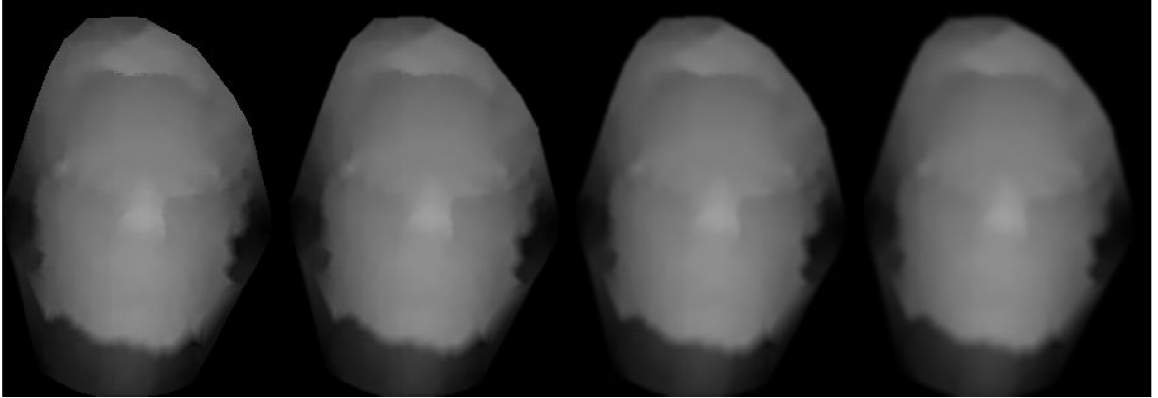


Figure 2.8: Gaussian blur, from the left: without Gaussian blur, with $\sigma = 3$, with $\sigma = 7$ and with $\sigma = 11$.

Laplacian smoothing

Another smoothing algorithm is Laplacian smoothing. Every vertex is moved to new location due to local information. Definition for the new position n_i of vertex:

$$n_i = \frac{1}{d_i} \sum_{t_j \in \mathcal{T}(v^*)} n(t_j) \quad (2.2)$$

where d_i is number of neighbours (adjacent vertices) $d_i = |v_i^*|$, t_j is an adjacent vertex of triangle $t_j \in \mathcal{T}_j^*$ and $\mathcal{T}(v^*)$ is set of triangles in the neighbourhood v^* [11].

Chapter 3

Kinect

This chapter is based on the official Microsoft Developer Network site including Kinect [12]. Kinect is a controller device originally designed for Microsoft's video game console Xbox 360, however, nowadays it's available for PC too. It is an alternative to gamepads, which offers far more possibilities of control and interactivity. This device can track full-body, it responds to user's movement, of course in its range. Kinect also responds to voice, which enables control via voice commands.



Figure 3.1: Kinect device

History of Kinect

The main design of Kinect was based on reference device developed by company PrimeSense. They found a significantly cheaper way to build a device to tracking motion and sensing distance from sensor. Microsoft started the Kinect project in 2007.

After years of solving many issues, Kinect was finally introduced in 2010 and official launch date was set on November 4, 2010. In the beginning, Kinect was exclusively designed for video game console Xbox 360, so there Kinect couldn't have been used on any other platform.

However, company PrimeSense released their open source software framework OpenNI in December 2010. Later, in 2011, Microsoft announced and later released Kinect software development kit for Windows.

Setting up a Kinect

1. Place the Kinect on a stable surface with low probability of falling down or getting damaged. It is reasonable to avoid putting the Kinect near devices which vibrate or make noise and placing it in direct sunlight. Keep it in environment with temperature between 5 and 35 °C and do not manually tilt the Kinect sensor, angle of its camera is controlled by software.
2. Install the proper software. If you want to develop applications for Kinect, you can download and install the official software *Windows SDK*, additionally *Developer Toolkit* or unofficial *libfreenect* from OpenKinect project. Software will be described in the second section.
3. Plug in your Kinect sensor. Connect its power supply to a power socket. Then connect Kinect to your PC's USB port. Wait for PC to identify the device.

3.1 Hardware

Kinect consists of these main components: RGB camera, 3D depth sensor (IR emitter and IR depth sensor), microphone array and motorized tilt.

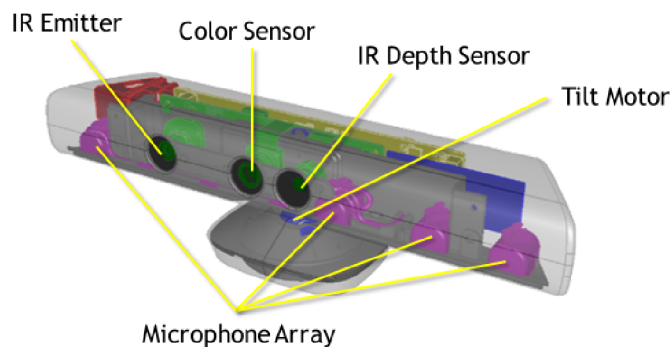


Figure 3.2: Main components of Kinect device

- **RGB camera** is a color sensor, which is used to capture color of models in RGB color space. The color is stored in a 1280x960 resolution.
- **3D depth sensor** consists of IR emitter and IR depth sensor. IR emitter projects an infra red light beams. These beams consist of irregular patterns of infra red dots. Whereas IR emitter emits IR beams, the depth sensor reads the reflected beams and afterwards reconstructs accepted information into a depth image. Depth image contains information of distance between sensor and an object. This enables the device to capture 3D models.
- **Microphone array** is an array of four microphones which captures the sound. These four microphones can record audio, find the direction and location of sound source. Besides it can more or less reduce the noise coming from the surrounding. Consequently it extends the distance that can the device capture sound from.

- **Motorized tilt** is automatically moves up and down. It is controlled by an application software, so it is advised not to tilt it manually. The sensor tilts down to find the floor and afterwards up to find the players.

3.2 Software

Kinect for Windows SDK

This official software includes API and tools for developing Kinect applications for Microsoft Windows in C++, C# and Visual Basic. In addition to SDK, **the Developer Toolkit** provides more resources, full source code examples and a tool Kinect Studio, which is for recording and playing back depth and color data [13].

Libfreenect

It's an open source driver for Kinect for PC, supporting operating systems Windows, Linux, Mac OS X. This driver is developed by open community of people, who are enthusiastic about creating a suite of applications for Kinect hardware. It includes wrappers for Python, C++, C#, Java, javascript, Lisp and more [14].

OpenNI

OpenNI is an open source framework used for the development of 3D sensing middleware libraries and applications. It was developed by company PrimeSense in November 2010. Today, OpenNI community provides tools, resources, tutorials and support for those, who want discover, develop and distribute all the possibilities it offers [15].

Chapter 4

Curvatures and surface

Because, the most of methods for face recognition and facial features detections are directly related to curvatures of particular object surfaces, some basics of curvatures in geometry are necessary.

The curvature measures how fast a curve is changing direction at a given point. Which means that sharply bended curve has large curvature, while straight line has 0 curvature. This is related to direction of tangent vectors of curve. The bigger the curvature is, the more the direction of tangent vector changes.

Therefore, curvature can be formally defined:

$$\kappa = \left\| \frac{d\vec{T}}{ds} \right\| \quad (4.1)$$

where \vec{T} is the unit tangent and s is the arc length.

This information can be used to determine curvature from osculating circle of curve.

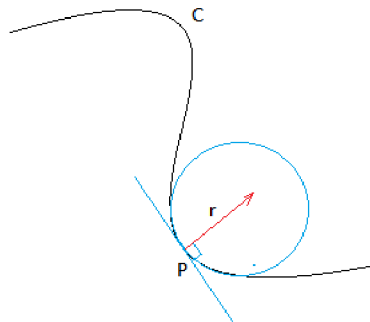


Figure 4.1: Osculating circle with radius r

The derivation of the formula for circle proves that curvature equals inverted value of circle's radius [16].

$$\kappa = \frac{1}{r} \quad (4.2)$$

Curvature showing amount of curve's bending at a given point in the direction of its normal vector is known as normal curvature.

Unlike 2D curve, 3D surface cannot be described by only one curvature. Two of these normal curvatures, the maximum and minimum, are called principal curvatures. First one determines the rate of maximal bending of the surface and the tangent direction, whilst the second characterizes the rate of minimal bending. According to Euler's formula, the rate of surface bending along any tangent direction at one point is determined by these two curvatures.

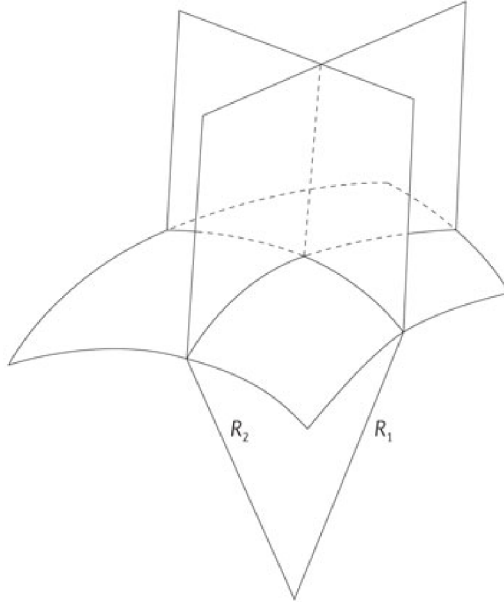


Figure 4.2: A curved surface in 3D space, where R_1 and R_2 are planes of principal curvatures [17]

4.1 Mean and Gaussian curvature

All formulas and picture are from [16]. Mean curvature is defined as the arithmetic mean of principal curvatures:

$$H = \frac{1}{2}(\kappa_1 + \kappa_2) \quad (4.3)$$

where κ_1 and κ_2 are principal curvatures. Mean curvature is an extrinsic measure, which means that it locally describes the curvature depending on its embedded surface.

Gaussian curvature is defined as the mean value of principal curvatures:

$$K = \kappa_1 \cdot \kappa_2 \quad (4.4)$$

where κ_1 and κ_2 are principal curvatures. Whereas mean curvature depends on particular embedding, the value of Gaussian curvature depends only on distances that are measured on the surface. Therefore it is an intrinsic measure.

4.2 Surface types

The values of Gaussian (K) and mean (H) curvatures can be used to classify the type of the specific surface. Classification is determined by signs of these values, nevertheless there are special cases, when these measures can have zero value. There are different situations which can occur depending on the curvatures' values:

1. $K > 0$. This is the case when the principal curvatures have the same sign. The surface is bending away from its tangent plane in all tangent directions at one point p . This point p is called an elliptic point of the surface. There are two possibilities depending on the sign of the mean curvature:
 - (a) $H < 0$. This surface looks like a peak.
 - (b) $H > 0$. Surface has form of a pit.

(a) $H < 0$. This surface looks like a peak.

(b) $H > 0$. Surface has form of a pit.

The type with 0 mean value doesn't exist in this case.

2. $K = 0$. In this case, if the only one principal curvature is zero, the point p is called a parabolic point. Otherwise, when both principal curvatures have zero value, the point p is a planar point to the surface.
 - (a) $H < 0$. This surface is classified as a ridge.
 - (b) $H = 0$. Both principal curvatures are zero, this surface is flat.
 - (c) $H > 0$. This type of surface is called a valley.

(a) $H < 0$. This surface is classified as a ridge.

(b) $H = 0$. Both principal curvatures are zero, this surface is flat.

(c) $H > 0$. This type of surface is called a valley.

3. $K < 0$. The signs of the principal curvatures are opposite at point p . This point is known as a hyperbolic point of the surface.
 - (a) $H < 0$. This kind of surface is a saddle ridge.
 - (b) $H = 0$. In this case, the surface is minimal.
 - (c) $H > 0$. This surface is called a saddle valley.

(a) $H < 0$. This kind of surface is a saddle ridge.

(b) $H = 0$. In this case, the surface is minimal.

(c) $H > 0$. This surface is called a saddle valley.









	$K < 0$: hyperbolic	$K = 0$: parabolic/planar	$K > 0$: elliptic
$H < 0$			
$H = 0$			not possible
$H > 0$			

Figure 4.3: Types of surfaces [18]

Chapter 5

Design of application

The subject of this thesis is the detection of these facial features: nose, eyes and mouth. These features will be detected on a set of 50 models generated by a Kinect acquisition application.

Before detecting any of the features, the model data must be loaded from the input file. Models are represented and stored in Wavefront OBJ (.obj) files. Afterwards, the loaded data must be processed, so it can be used for detection.

There exists many methods for detection of facial features, so in one of the following chapters will be described the chosen method.

After performing the detection itself, the results will be presented to the user.

5.1 Data processing

The points in mesh are spread in every direction in different distances from each other, so it wouldn't be easy to estimate their curvatures. And even then, the results could be rather inaccurate. In order to prevent that, the mesh is transformed into a depth map. This depth map is basically a matrix, in which the distance between every pair of points in x-axis and y-axis is equal. This allows to determine curvatures at a given point by estimating it's value in point's horizontal and vertical direction. 3D model consists from vertices and faces, which connects them. Vertices are taken from 3D model and put into depth map. And remaining empty point in depth map are filled by creating points using interpolation of neighbouring points (explained in Section 2.3).

Of course, acquired data aren't in perfect quality and contain noise. That could be solved by applying the Gaussian blur on the set of points. Or a different approach could be used - take more than only two neighbours from vertical and horizontal direction and estimate curvature from them. Because the library OpenCV already includes an implemented Gaussian blur, this is the applied approach how to eliminate incorrect results in this part.

5.2 Specification of surface types

The following method is used to determine curvatures: the principal curvature is computed as the perpendicular distance between the given point and line crossing his two adjacent points. As a result are given two curvatures, minimal and maximal. Gaussian and mean curvatures are gained by applying the minimal and maximal curvatures to their formulas.

5.3 Detection of specific features

The method of detecting facial features is based on using the surface type at each point. This idea is based on [19, 20]. Every facial feature has a specific shape. In relation to its shape it is obvious, which surface type needs to be look for in order to find the demanded facial feature. Of course, due to models imperfection, some of the types of surfaces might be interpreted incorrectly. To prevent this problem, there can be set regions, which the searched feature is probably in. For example, it can be assumed that nose is in the middle of the face or mouth is in the lower third of the face.

Another possibility is to set limit, after which the looking for the specific value of the feature will be terminated, e.g. if the face isn't upside-down, the mouth will always be under the nose.

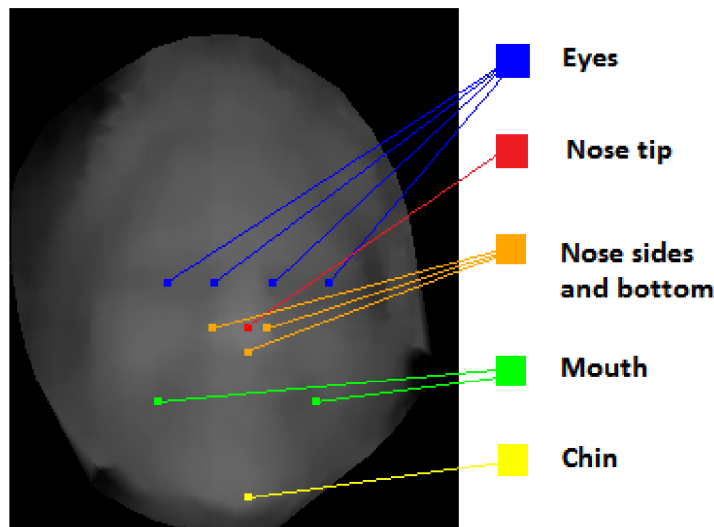


Figure 5.1: Face depthmap with detected facial features, left and right sides of face are considered from the user's perspective

5.3.1 Nose detection

The detection of nose consists of detecting nose tip, nose sides and the bottom of nose. The nose tip is considered as the most important element, because it helps to detect some other features. The detection of nose tip and it's area is the easiest, because nose is often the most salient and most visible part of the human face. Therefore, there are many ways and methods to locate this feature.

Nose tip

The first step to get y-coord of nose tip is to take the highest z-value of every row (y-axis) and therefore create a profile curve. This curve shows the highest and the deepest point on face. After that, a median value is taken from every of these rows, which creates a median curve. This will show the extreme points compared to other points of the same row. And finally a difference between these two curves, profile and median. Result is shown in Figure 5.2. Nose is far more high than wide, so the nose tip will be the most significant

point in this curve. However, face can be turned to one side, face model can be a bad quality or the person can have smaller nose and this can result in an incorrect y-coord. To prevent any of these cases, there is set a region of probable nose tip occurrence.

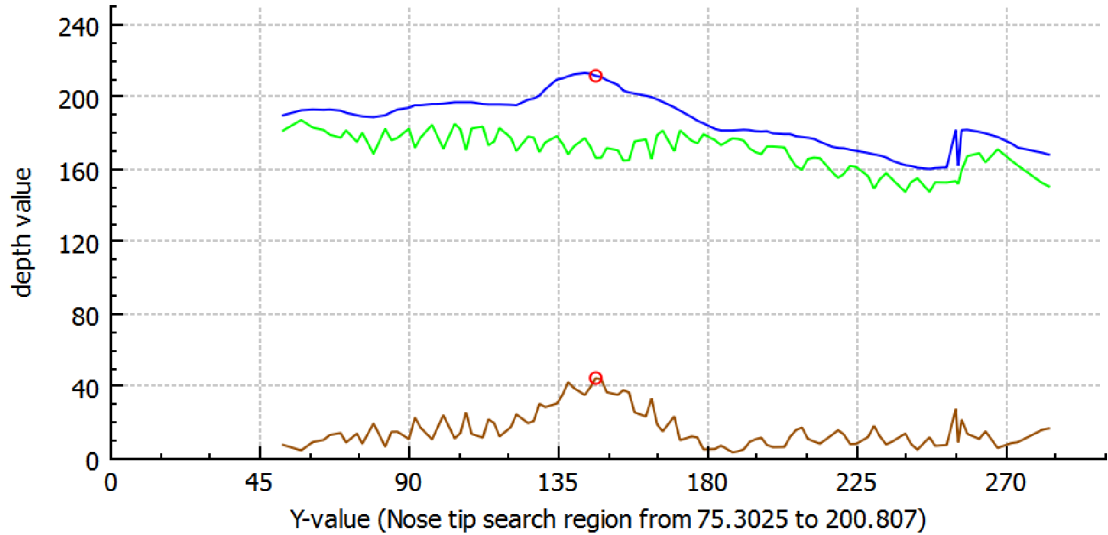


Figure 5.2: Graph of getting nose tip y-coord: profile curve is blue, median green and difference brown

The detection of x-coord of nose sides and nose bottom as well as all x-coordinates is based on a different approach, which is related to surface types. According to the shape of nose, the searched type of surface is peak. X-coord of nose tip is found by searching the biggest density of peaks in the row with already detected y-coord.

Nose sides and nose bottom

These parts of nose are located through the position of nose tip, if successfully detected. In this design, the nose sides are meant as the nose area boundary near the nose tip. Therefore, as the y-coordinate of nose sides is directly used the y-coordinate of the nose tip. Both nose sides have this coordinate the same, because in some cases it could be confused with the bottom of nose or it could be difficult to explicitly see it in the result. The second coordinate is determined by looking for first more significant presence of the valley surface type. Because of searching for both sides, the valley is being looked for in both directions, left and right.

The nose bottom is the ending part of nose, it's vertically on the same line with the nose tip. Its x-coordinate corresponds to nose tip's. Again, this approach was chosen due to its easy readability. So unlike the nose sides, in the nose bottom case it's needed to find y-coord. This is done by a vertical search for the last nose peak location. This search starts under the nose and continues downwards.

5.3.2 Eyes detection

Eyes can be located thanks to its' special profile. Because of the eye balls, the sensor is not able to capture the most of the eye pits. Despite this fact, it is enough to detect pit surface

type in the area of eyes. This area is the most significant area containing pits on the whole face, as shown in Figure 5.3.

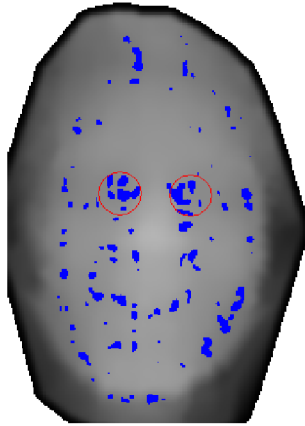


Figure 5.3: Depth map with highlighted eyes areas containing pits

Eyes are detected by four points, every one of them has the same y-coordinate and they describe: left corner of left eye, right corner of left eye, left corner of right eye and right corner of right eye (from viewer's perspective). The y-coord of the eyes is found by counting pits in each row of depth map. The index of row with the biggest pit count is set as the y-coordinate.

In this case, there are detected pairs of points, where one point is the beginning of eye and the second its end. In the depth map, these two points are corners of the cluster of pits. So search begins in the left part of face and continues horizontally till the end on the right. In the default scenario, face is looking straightforward. Consequently, there has been set two borders: left in the first quarter and right in the third quarter, because eyes should be somewhere in this area. The process of left corner of left eye consists of looking for the first more significant occurrence of pits. Right corner is assumed to be in the end of cluster of pits, so it searches for the first point, which is not a pit. This method is also used for the right eye, but in the reversal way. Search starts on the right corner and continues to find the left corner. However, as seen in Figure 5.3, this cluster has many points, which are not considered as pits. To solve this, there has been a preset minimum length of eye to avoid unrealistically short eye due to precocious end of cluster.

5.3.3 Chin detection

The chin detection is a subsidiary part of this application. It is used to detect mouth. Chin is the bottom border of region, where mouth is looked for.

The y-coordinate is index with highest density of peaks in the first sixth of face. And the x-coordinate is the same as nose tip's.

5.3.4 Mouth detection

Mouth detection is the most difficult part, because compared to other features, mouth is almost flat. Additionally, the shape of mouth changes the most of all features with the different facial expressions. When has the face model captured yawning, mouth is the

largest area with highest pit density. However with the sticking tongue out, it is one of the highest regions on the face and contains peaks. These special cases are discussed in the Section 5.4. In the text below, it is expected to have a face without any special facial expression.

Mouth region is defined by nose bottom and chin vertically, with the first quarter and third quarter horizontally. Surface types are not very significant in this area and their density differs with every person. The y-coordinate is found the same way how nose tip's (in Subsection 5.3.1) with different region for looking for the maximum difference between profile and median curve. This method has been chosen, because it does not need surface types.

The second coordinate detection however uses surface types. Because of mouth's insignificant shape, the searched surface type can differ in this case. When the search ends with an unsatisfying result, it continues with different surface type. It starts with valleys, in case of a failure it repeats the process with peaks instead of valleys. If even this did not end with a satisfying result, pits are used as the searched surface type. The order of these three types has been set on the results of experimenting with probability. The detection itself starts in the x-coordinate of nose tip and continues left or right direction, depending on the mouth corner, and looks for the first appearance of the surface type in already known y-index.

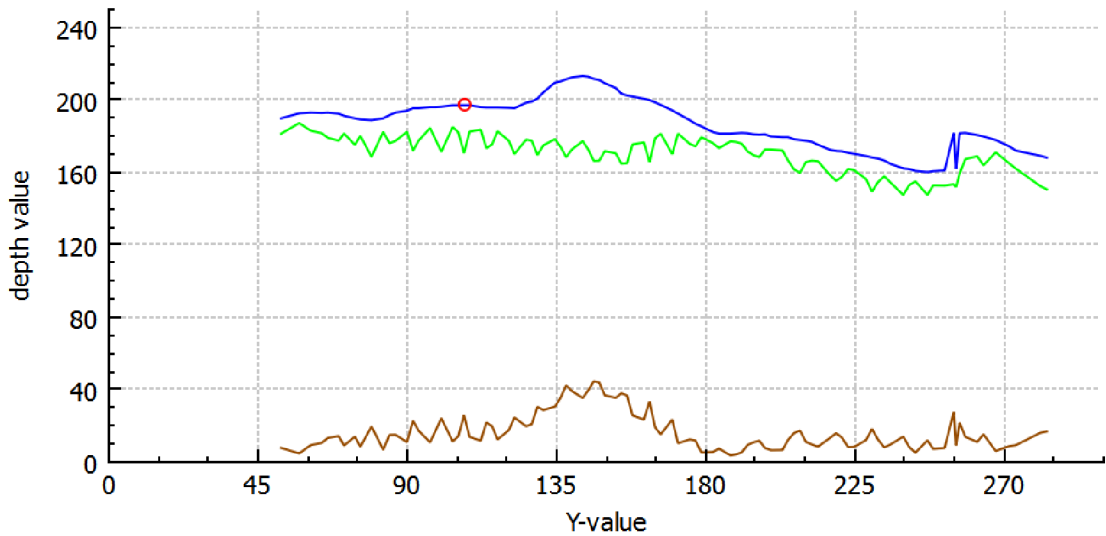


Figure 5.4: Graph of getting mouth y-coord

5.4 Special cases

In the previous sections it is assumed that 3D face model is looking directly into camera, has casual facial expression and anything isn't covering the face. However, a lot of models are not as good as in this case. The subject might be distracted, wearing glasses, yawning, etc. All of this can have impact on model's quality and lower the chance of successful detection. In the acquired set of 3D models, the facial features detection is influenced by several factors: face rotation, facial expression and various accessories on face. Some factors

change the shape of particular features, which can lead to unsuccessful detection.

This application has been designed to handle several of these possibilities: face rotation to the left/right and a few facial expressions.

5.4.1 Turned face

When model isn't looking directly into camera, there are two other cases handled in this application: turn to left and right. Looking upwards and downwards is not considered, because it isn't as influencing as looking to left or right - it doesn't require rotation of neck.

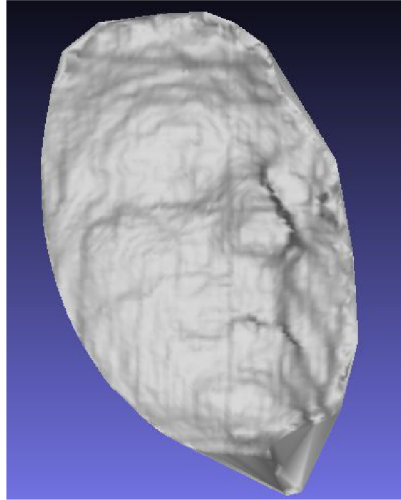


Figure 5.5: Model of face turned to the right (smoothed for a better visibility)

As seen in Figure 5.5, face rotation to right changes the layout of features on face:

- Nose is out of its original middle position.
- Mouth is also in slightly different position.
- Right eye is less visible and therefore more difficult to detect.
- Basically whole main axis nose-mouth-chin is moved to right.

The measure of these changes is related to how much the face is rotated. The solution to this situation is to change the regions of probable location for each feature, e.g. nose tip won't be expected in the middle of face, but the space between middle and right border of face.

Previous items applies also to left rotation, of course with orientation to left.

5.4.2 Special features

Glasses

First special case occurs when model is wearing glasses. Glasses slightly decrease pit count in the eyes area and increase the count of peaks. So this case can be detected by counting peaks near eyes and check if this value exceeds a preset limit.

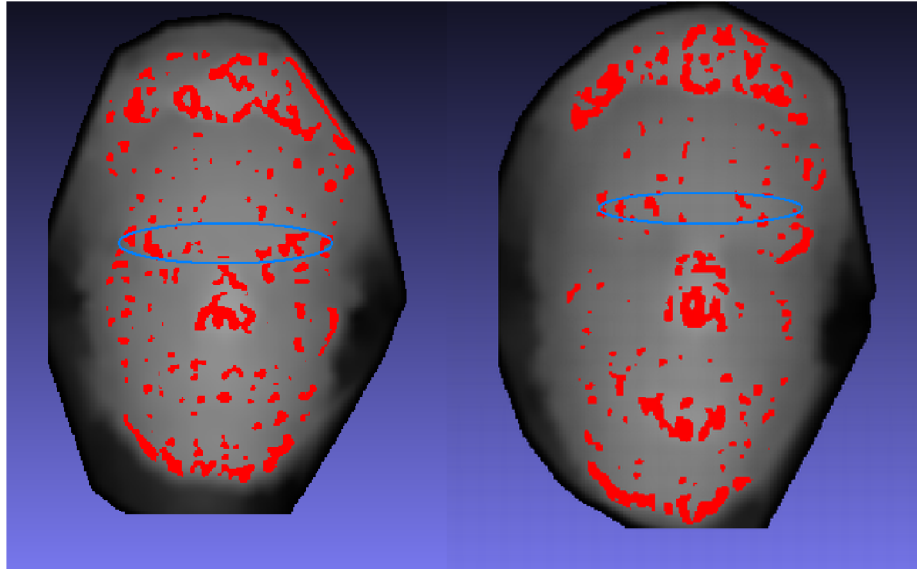


Figure 5.6: Compared pit areas above eyes with glasses (left) and without glasses (right)

Smile

If Kinect has captured smiling person, it results with pits in mouth corners comparable to pits detected near eyes. Eyes area should contain the most pits. To distinguish smile from eyes, the location with highest pit density is taken. If is this area located under nose tip, it is considered smile and if above the nose, area is treated as eyes region. In case of smiling face, application looks for region with second highest pit amount.

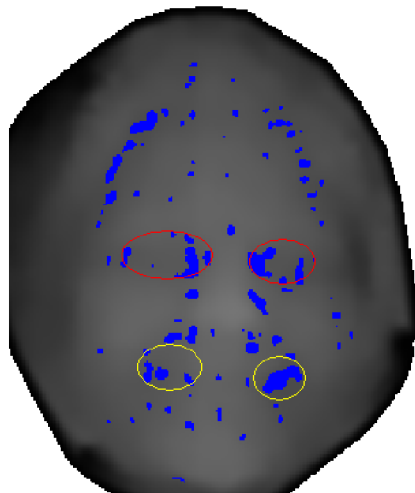


Figure 5.7: Depth map with highlighted eyes and smile areas containing pits

To detect smile, it must be wide enough. The corners of smiling mouth must have higher density of pits than eyes. Otherwise, the smile won't be detected.

Opened mouth

In this case, face has an opened mouth. It leads to pit area like with smile, but it is one bigger region.

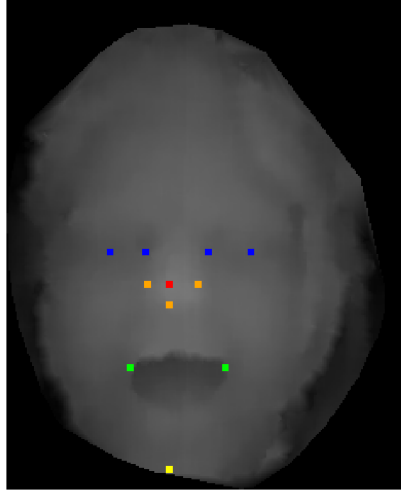


Figure 5.8: Face depth map with widely opened mouth

To detect an opened mouth, detection could simply focus on pits. However, it's not that easy. Vertices, which were in OBJ file too deep, weren't saved into depth map. Because of this, only pit areas, which are significant, will be right under the lips, which is not enough. This method had to be replaced with another using profile curve from nose tip detection (Section 5.3.1). Similar method is used, the region, which the detection will be applied for, is from the face bottom to the bottom of nose. In this area will be located lowest point, with highest difference with the Z-value of nose tip. If is this difference greater than preset limit, mouth on this face model is opened.

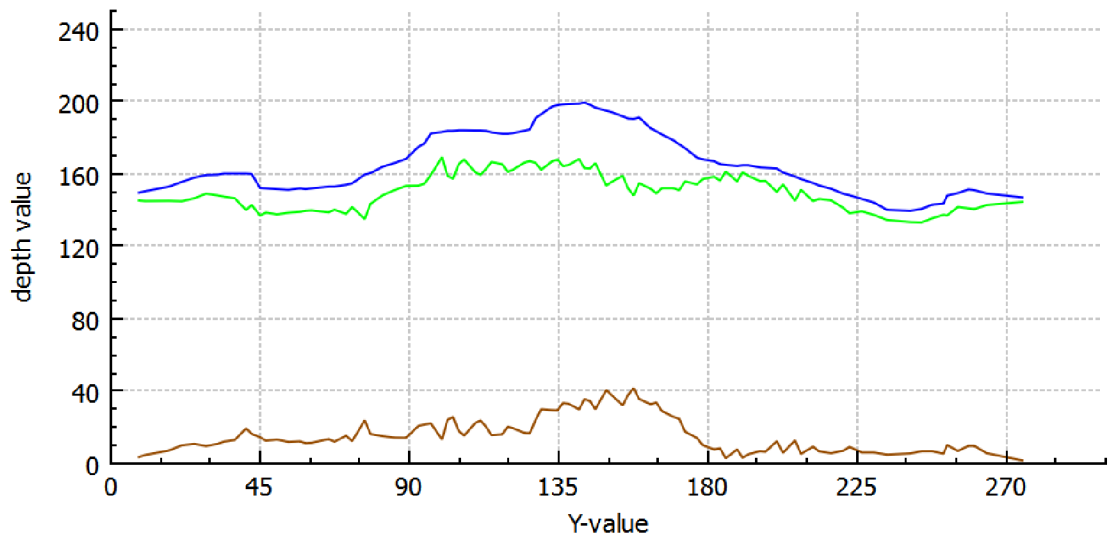


Figure 5.9: Graph of profile curve with opened mouth

Tongue

The last special feature is the tongue. If person is sticking the tongue out, it creates another high feature on face.

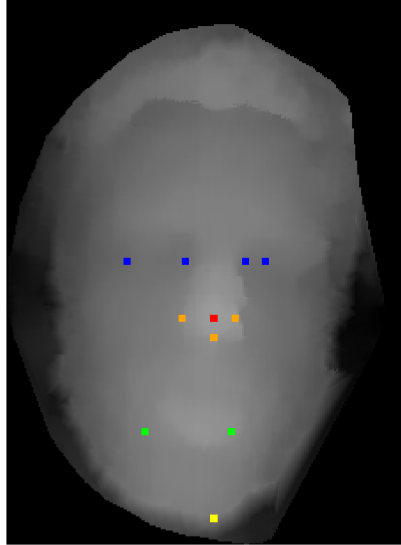


Figure 5.10: Face depth map with tongue sticking out

In this case, the mouth can be located by looking for peaks. Also, it can be located by similar method as in previous paragraph. In this case, application won't look for lowest, but highest point in area approximately between chin and nose bottom. If this point is only slightly lower or even higher than nose tip, face captured on this model is sticking its tongue out.

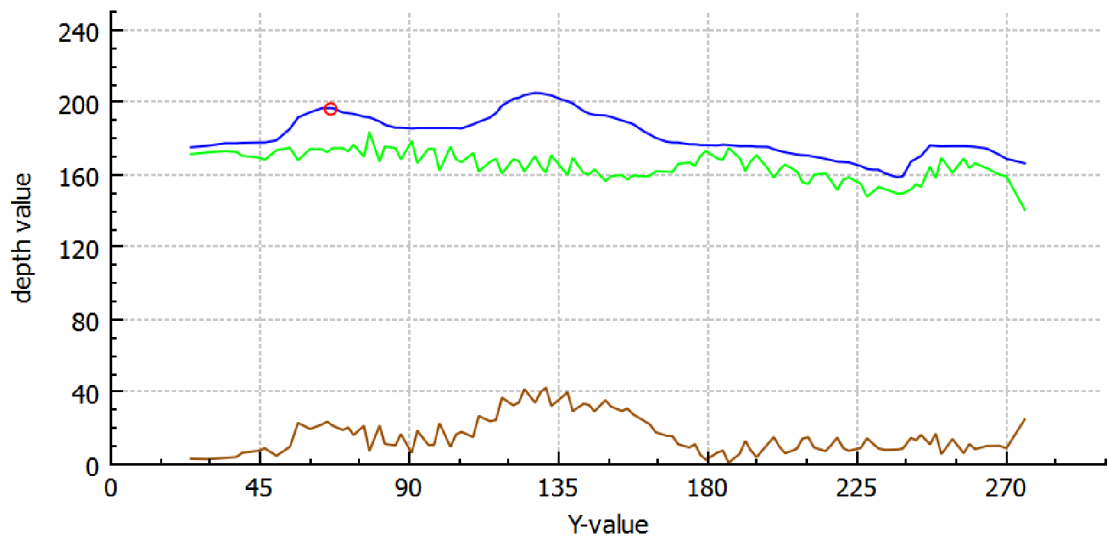


Figure 5.11: Graph of profile curve with tongue

Chapter 6

Implementation

This chapter describes facial features detection application implementation details, which are based on design from previous chapter. The application is creating for Windows and Linux OS. As programming language was chosen C++ and Qt framework, mainly because of OpenCV's support and author's experience with this language. In the sections below will be described used technology and classes.

6.1 Used technology

OpenCV

This open source library was used because of many reasons. This library focuses on image processing, which is necessary in creating 2D depth map from original 3D model and vice versa. Also, it includes basic image treatment functions (blurring, brightness, etc.). Last but not least, this library is cross-platform, runs on Windows, Mac OS X and Linux.

Libfreenect

Libfreenect is used to acquire data from Kinect, more information in Section [3.2](#).

QCustomPlot

Graphs shown in this thesis are made by QCustomplot, Qt C++ widget focusing on making 2D plots, graphs and charts.

6.2 Facial features detecton implementation

6.2.1 Depth map class

Depth map and every function related to it is represented by class `Depthmap`. This class consists of two main members: `DepthMatrix` and `CurvatureMatrix`, both are 2D matrices (type `cv:Mat`). `DepthMatrix` stores 64-bit floating-point Z-values of depth map points. And `CurvatureMatrix` is used to store surface type of every point in the depth map, which is represented by 32-bit signed integer value. The process of creating depth map from 3D model is implemented in method `createMap`.

Filling the remaining empty spots proceeds in method `fillDepthmap`, where is implemented a parallel algorithm for rasterization [\[21\]](#). This algorithm uses edge function, which

determines whether the point is to the left or to the right from the edge or exactly on the line of the triangle. Edge function is defined [21]:

$$E(x, y) = (x - X)\delta Y - (y - Y)\delta X \quad (6.1)$$

where x, y are point coordinates, X, Y are vector coordinates and $\delta X, \delta Y$ are increment values for each iteration. This algorithm traverses the triangle from the bottom to top, from left to right and evaluates the depth values for every point inside the triangle by linear interpolation of triangle vertices. The distances from each triangle vertex serve as weight:

$$z = z_A w_A + z_B w_B + z_C w_C \quad (6.2)$$

where z is depth of points inside triangle, z_A, z_B, z_C are depths of triangle vertices and w_A, w_B, w_C are weights (distances between point and triangle vertices).

This class also includes the process of determining surface types in method `curvaturesType`. Firstly, there are estimated principal curvatures for every point. Then are computed Gauss and mean curvatures and finally from them determined surface types.

6.2.2 Detection class

Detecting facial features is represented by class `Recognition`, which includes methods for detection of particular features. List of members storing the coordinated of features:

- Nose: `noseY, noseBotY, noseX, noseRightX, noseLeftX`.
- Eyes: `eyesY, eyesLeftX, eyesLeftX2, eyesRightX, eyesRightX2`.
- Mouth: `mouthY, mouthLeftX, mouthRightX`.
- Chin: `chinY, chinX`.

This class includes instance of class `Depthmap` `_depthmap` and triggers the creation of depth map in member function `depthMap`. The process of detection starts will method `detectFeatures`, which sequentially detects all features by calling these methods: `findNoseTip, findEyes, findChinTip` and `findMouth`.

6.2.3 Special cases

Special features are located before and during the regular features detection. The class dedicated to special features is named `SpecialFeatures`. This class basically consists of member functions `findSpecialFeatures` and `setSpecialFeatures`. First one determines, whether the face contains opened mouth, tongue or neither of these. The second initiates the special features values. Other features, face direction, smile and glasses, are found during detecting the regular features and set in this class later. The members used to store bool value `true` or `false` are: `_glassesEyes, _smileMouth, _openedMouth, _tongueMouth`, and member with one of these string values `Normal, Left` or `Right` is named `_directionFace`.

6.3 User interface

The user can access and control the application through the graphical user interface (GUI).

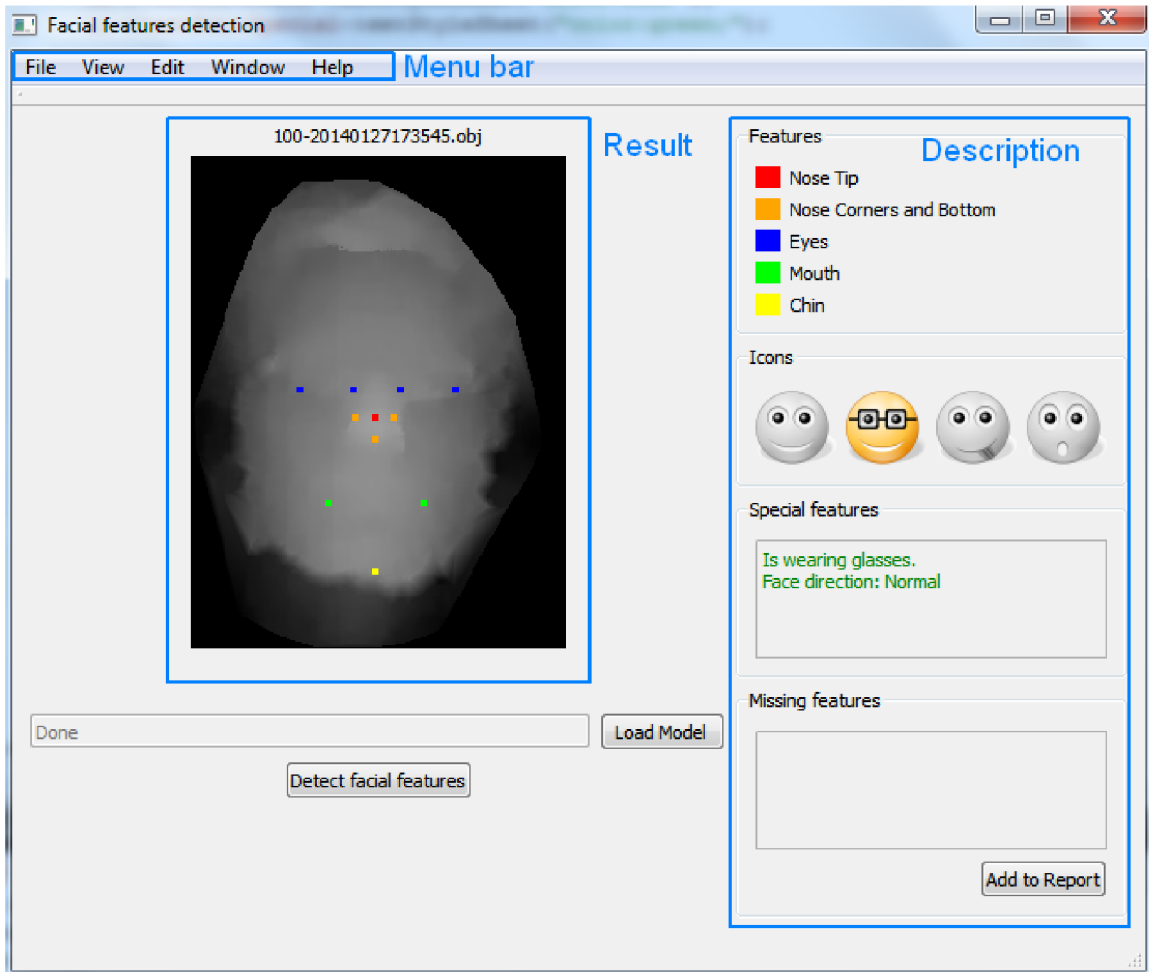


Figure 6.1: Graphical user interface of application

The GUI is divided into three part according to their purpose:

1. The first is menu bar, which offers manipulation with input and output files (load, save, exit), showing and hiding specific elements of GUI and other actions.
2. The second part is the part showing the name, depth map image of loaded 3D model and detected features on this image.
3. The last are boxes, which describe results of the detection. Results are explained in text and graphically by icons. The icons as well as special and missing features boxes can be optionally hidden. Missing features can be added to report, which could be later used in the diagnosis of results.

There is also implemented testing of results. Testing buttons are in default hidden, so user can access them through menu bar. This will enable manual setting position for each facial feature. Afterwards, the evaluated results will be written to output.

Chapter 7

Experiments

Designed and afterwards implemented application was tested on the set of 50 3D models acquired with Kinect. The 3D models were obtained by capturing six different persons with various facial expressions.

The experimenting consists from two parts. Calculating the percentage of successfully detected facial feature for whole set. Second part is found out, whether the special features were recognized properly.

The testing set was consisting from detection results and manually set features in their correct position. This was done through implemented function for setting point directly on image by left-clicking. Every click was for each feature in a specific order.

7.1 Testing detected features

The results were compared to manually set facial features on model depth maps. Comparison consisted from calculating the distance between points detected by application and manually set points into correct positions:

$$\begin{aligned}\Delta x &= |x(\textit{detected}) - x(\textit{correct})| \\ \Delta y &= |y(\textit{detected}) - y(\textit{correct})| \\ \Delta &= \sqrt{\Delta x + \Delta y}\end{aligned}\tag{7.1}$$

Based on difference, the results were divided into three groups according to their accuracy. Every group has a difference limit, which every result complies with. Difference values were preset in order to divide results into three groups, where each of them has enough samples to create statistics. The three groups depending the results quality are [22]:

- Good - the accuracy of results belonging to this group is satisfying. Detected facial features are successfully detected, approximately near their correct position.
- Bad - these results are less satisfying, but still for the most part are correct. Most of samples have, despite the classification name, solid quality.
- Ugly - results in this group are, simply put, failed attempts for detection. The circumstances for successful detection were very low in these 3D models.

Good results

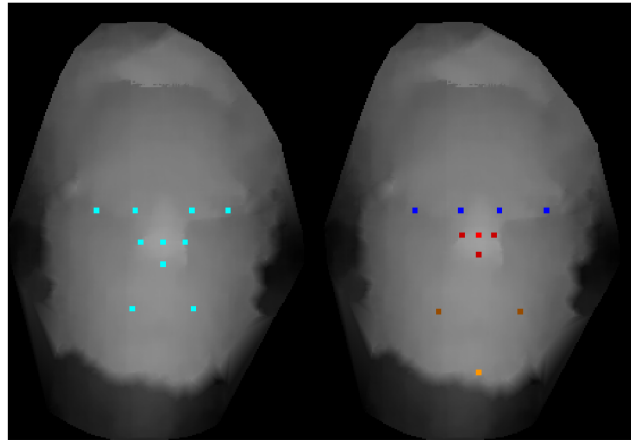


Figure 7.1: Example of manually set result (left) and good result (right)

In the results belonging to this group was the overall difference below 75 points. Samples belonging in this group are considered with a good quality. The inaccuracy in some cases may not be even visible or it might just slightly differ from correct positions. This groups consists of 20 results with the highest detection success.

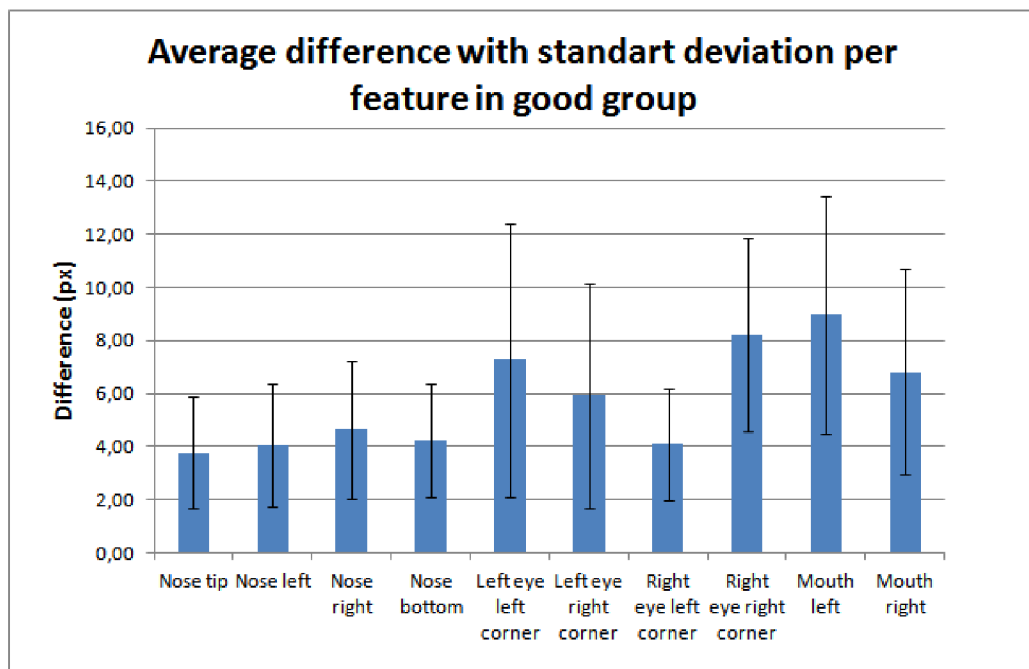


Figure 7.2: Graph showing average difference for each feature in good results

As seen in Graph 7.2, the most problematic feature to recognize, was the left side of mouth. Furthermore, mouth was the feature with the biggest average difference between the manually set points and detected. Left and right mouth side together have the highest average difference than other features. It was mainly because the mouth has the biggest

area of probable location and it is also the most divergent feature. On the other hand, the nose tip was most successfully detected feature.

The highest standard deviation is in the case of left eye detection, however it's still low enough, because all standard deviation values scale from 2 to 5 pixels.

Bad results

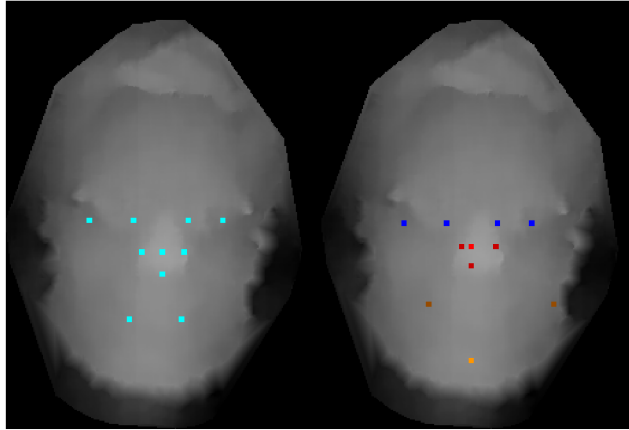


Figure 7.3: Example of manually set result (left) and bad result (right)

The overall difference limit was under 120 pixels. 20 results were considered as bad. In majority of samples, one detection has partially failed. One of features wasn't found, was absolutely out of it's correct position or most of detected features were too distant from their manually set location.

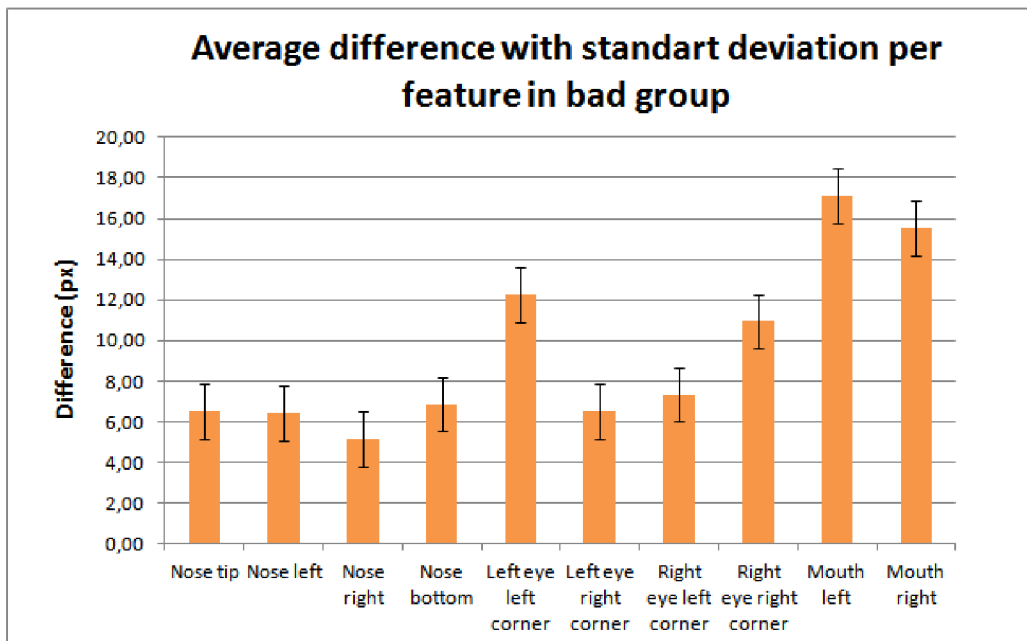


Figure 7.4: Graph showing the average difference for each feature in bad results

Similarly like in previous group, mouth ended up with the biggest average difference and the nose with the lowest. However, in contrast with good group, the standard deviation is highest in the mouth case. The inaccuracy of mouth was drastically higher, average difference of left side of mouth is 52% higher more than in the group with good results.

This group contains most of special features and cases, where mouth is difficult to be recognized accurately.

Ugly results

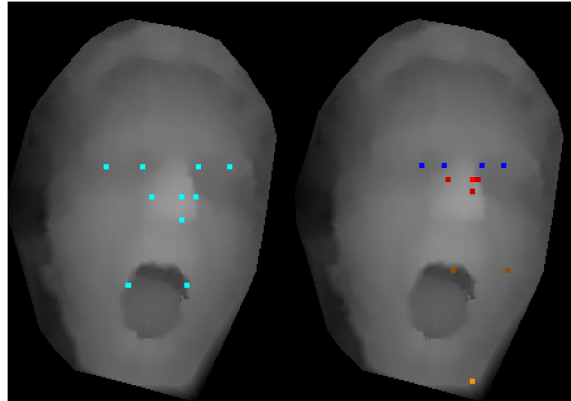


Figure 7.5: Example of manually set result (left) and ugly result (right)

Any result with difference higher than 120, was classified as ugly. This group consists of 10 samples, mostly with missing features or with more features too distant from correct position.

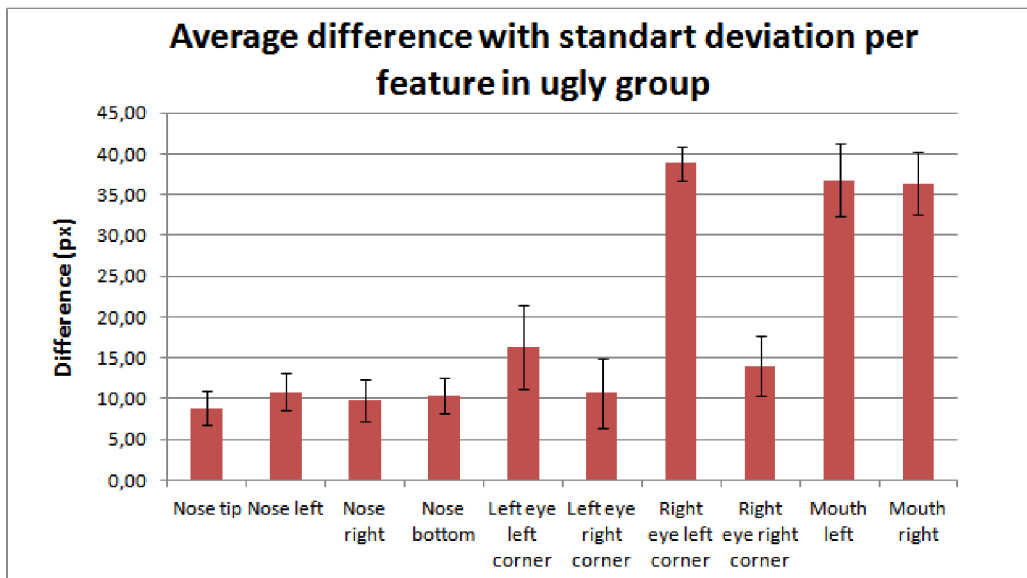


Figure 7.6: Graph showing the percentage of total difference for each feature in ugly results

There could be a few reasons: because of their low quality, too complicated combination of facial expressions or excessively rotated face. Some of results in this group might have

too low quality, therefore suggestion is to discard them from the set of 3D models. Or in the worst case, there must be used another approach to detect features on these models.

According to Graph 7.6, in the most results the worst detected feature there was one of the mouth sides. However, in some cases, right eye wasn't detected at all, what resulted into a huge difference. Therefore, the least successful in this group was the detected of right eye.

On the contrary, nose points had the lowest differences. In average they didn't exceed 9 pixels.

Majority of model results classified as ugly, were facing to the right side. Because of this, in these locations wasn't found enough pits to successfully detect eyes. This explains the undetected right eye in several depth maps.

Overall results

In this paragraph, results will be summarized overall for all groups. Nose remained best results in all groups. Nose as a whole is the most successfully detected feature, specifically nose tip and nose right side had the lowest average difference, both under 4 pixels. Regarding the eyes, right corners of both eyes were detected more accurately than the left corners. Mouth ended up worst of all features, it's left side to be specific. It reached almost 18 pixels average difference value. Standard deviations of left and right mouth corners were almost as high as their mean value. However, the detection failed the most times in the case of left corner of right eye, where it had not been detected at all.

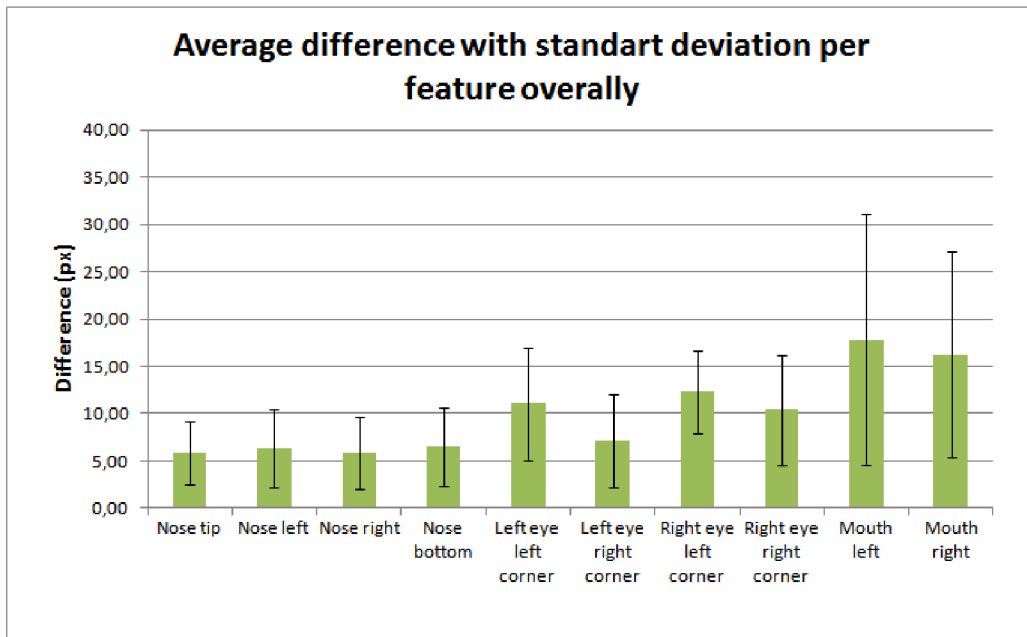


Figure 7.7: Graph showing the percentage of total difference for each feature overall

As undetected features considered, there were three cases left part of right eye wasn't detected. Additionally, there was one sample, in which mouth detection failed. All of there four results are belonging to group with ugly quality.

These experiments showed, that considering face looking directly onto camera, mouth is almost in every case the most difficult feature to be successfully and accurately located.

The second most problematic issue, was determining position of right eye, when face was turned to the right. None of samples captured person looking to the left, however it is suspected, that it would have similar result - difficult detection of left eye.

7.2 Testing special features

This section will summarize testing of the special features recognition described in Subsection 6.2.3. All implemented special features were tested: face direction, smiling, opened mouth, glasses and tongue. There were total 50 3D models to be processed, 6 of them had different face direction than normal - all of them had right direction. According to Table 7.1, this recognition had 100% success. Right face direction was present only in bad and ugly groups, all of good quality results faces were looking directly into camera.

Table 7.1: Table with results of testing special features detection

Special feature	Test results status			
	True positive	True negative	False positive	False negative
Face direction	12%	88%	0%	0%
Glasses	14%	82%	0%	4%
Smile	8%	84%	4%	4%
Opened	8%	84%	0%	8%
Tongue	8%	84%	6%	2%

There were 9 models with glasses, 2 of them wasn't successfully detected, resulting into 4% undetected glasses. Every failed glass recognition resulted into classifying detection as a bad quality. The ugly group, however, doesn't contain any face model with glasses.

Smiling face had been captured on 6 3D models, where four of them were correctly detected, two weren't detected at all and other two were incorrectly detected on models, where they shouldn't had been. Both occurrences of incorrectly determined smile were in samples with ugly quality.

Opened mouth had the highest rate of all false negative results. Nonetheless, there was none wrong detection of opened mouth on model where it isn't present. Mostly, if the mouth wasn't correctly treated as opened, it led into result's bad quality.

And finally, 5 of 50 models had face with sticking tongue out. Four of them were correctly located. Three detections showed tongue on wrong models. Although, in the good quality group, the tongue was mostly successfully found.

As the experiments show, incorrectly detected special feature on sample, where it is not present, has bigger impact on result's quality than missing special feature. Special features occurred in the experiments overall 39 times. Successfully detected were 25, which is 64%, the number of false negative results is 9 (23%) and false positive results were in 5 cases, presenting 13% overall.

The results of experiments have confirmed, that mouth is the most difficult feature to detect. All special cases related to mouth had the lowest success rate. The mouth has the largest area, where it might be located and every facial expression changes the presence of surface types and depth values near mouth far more radically than other features.

Chapter 8

Conclusion

The main goal of this bachelor thesis was to implement an algorithm for facial features detection from 3D models obtained with Kinect sensor. The result of design and implementation of this algorithm is an application, which detects main facial features: nose, eyes, mouth and additionally chin.

The quality of database acquired by Kinect wasn't highest and many models included various face expressions, so the results were expected to be rather inaccurate. To improve the success rate, especially of the mouth detection, there were besides regular features additionally implemented some more specific special cases, which have occurred in obtained samples. This part extends the original thesis assignment in order to improve the detection accuracy. These special features were face direction, several of facial expressions (smile, yawn or sticking the tongue out) and finding whether is the captured person wearing glasses.

On both feature types were used similar methods to prove that they can be used for this matter. Applied method was using surface types combined with expected location of these types according to natural shape of these features.

The results were for experimental purpose quite satisfying. But in real world, of course, the results using detection need to be as accurate as possible, so there is room for improvement. By further experimenting there could be set better threshold values, implemented more complex versions of algorithms and structures. Also, there can be added more tools to edit or manipulate with depth maps or detection results.

Bibliography

- [1] A. Jain, R. Bolle, and S. Pankanti, “Biometrics: Introduction to biometrics (chapter 1).” http://link.springer.com/content/pdf/10.1007%2F0-306-47044-6_1.pdf, 1996. ISBN 978-0-306-47044-8 [online].
- [2] P. Vincent Tao, “Article: 3d data acquisition and object reconstruction for aec/cad.” [online] <http://www.directionsmag.com/articles/3d-data-acquisition-and-object-reconstruction-for-aeccad/123668>. Accessed: 2013-10-22.
- [3] Štěpán Mráček, J. Váňa, R. Dvořák, M. Dražanský, and S. Yanushkevich, *3D and Thermo-Face Fusion, New Trends and Developments in Biometrics, Dr. Jucheng Yang(Ed.)*. InTech, 2012. ISBN 978-953-51-0859-7.
- [4] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy, “Polygon mesh processing.” [online] <http://books.google.cz/books?hl=sk&lr=&id=8zX-2VRqBAkC>, 2010. ISBN 978-1-56881-426-1.
- [5] “Meshlab.” <http://sourceforge.net/apps/mediawiki/meshlab/index.php> [online], 2012. Accessed: 2013-10-22.
- [6] G. Leparmentier, “Manipulating colors in .net - part 1.” [online] <http://www.codeproject.com/Articles/19045/Manipulating-colors-in-NET-Part>, 2007. Accessed: 2014-03-22.
- [7] Q. Fang and D. Boas, “Tetrahedral mesh generation from volumetric binary and grayscale images.” [online] <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5193259>, 2009. ISBN 978-1-4244-3932-4.
- [8] D. Vartziotis and J. Wipper, “The geometric element transformation method for mixed mesh smoothing.” [online] <http://link.springer.com/article/10.1007/s00366-009-0125-6>, 2009. ISSN 1435-5663.
- [9] D. Vartziotis, T. Athanasiadis, I. Goudas, and J. Wipper, “Mesh smoothing using the geometric element transformation method.” [online] <http://www.sciencedirect.com/science/article/pii/S0045782508000996>, 2008. ISSN 3760-3767.
- [10] M. Nixon and A. S. Aguado, “Feature extraction & image processing.” [online] <http://www.itu.dk/courses/SIGB/F2011/untitled%20folder/Reading/>

- [RelatedBooks/FeatureExtraction&Imageprocessing.pdf](#), 2008.
ISBN 978-0-12372-538-7.
- [11] H. Badri, M. E. Hassouni, and D. Aboutajdine, *Kernel-Based Laplacian Smoothing Method for 3D Mesh Denoising*. Springer Berlin Heidelberg, 2012.
ISBN 978-3-642-31253-3.
- [12] Microsoft, “Kinect for windows sensor.” [online]
<http://msdn.microsoft.com/en-us/library/hh855355.aspx>. Accessed:
2014-03-26.
- [13] Microsoft, “Kinect sdk.” [online]
<http://msdn.microsoft.com/en-us/library/hh855347.aspx>. Accessed:
2014-03-26.
- [14] OpenKinect, “Openkinect project - libfreenect.” [online] <http://openkinect.org/>.
Accessed: 2014-03-26.
- [15] OpenNI, “Openni.” [online] <http://www.openni.org/about/>. Accessed: 2014-04-10.
- [16] Y.-B. Jia, “Curvature, gaussian curvature.” [online]
<http://www.cs.iastate.edu/~cs577/handouts.html>, 2013. Accessed: 2014-03-22.
- [17] J. Zimmerberg and M. M. Kozlov, “How proteins produce cellular membrane curvatures.” [online]
http://www.nature.com/nrm/journal/v7/n1/box/nrm1784_BX1.html, 2005.
ISSN 1471-0072.
- [18] F. Crosilla, D. Visintini, and F. Sepic, “Reliable automatic classification and segmentation of laser point clouds by statistical analysis of surface curvature values.” [online]
<http://link.springer.com/content/pdf/10.1007%2Fs12518-009-0002-4.pdf>,
2009. ISSN 1866-9298 [print], ISSN 1866-928X [online].
- [19] M. P. Segundo, C. Queirolo, O. R. P. Bellon, and L. Silva, *Automatic 3D facial segmentation and landmark detection (pages 431-436)*. 14th International Conference on Image Analysis and Processing, 2007. ISBN 978-0-7695-2877-9 [print].
- [20] K. Chang, W. Bowyer, and P. Flynn, *Multiple Nose Region Matching for 3D Face Recognition under Varying Facial Expression*. IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume:28, Issue: 10), 2006. ISSN 0162-8828.
- [21] J. Pineda, “A parallel algorithm for polygon rasterization.” [online]
http://dl.acm.org/ft_gateway.cfm?id=378457&ftid=73749&dwn=1&CFID=339058548&CFTOKEN=91557720, 1988. ISBN 0-89791-275-6.
- [22] P. J. Phillips, J. R. Beveridge, B. A. Draper, G. Givens, A. J. OToole, D. S. Bolme, J. Dunlop, Y. M. Lui, H. Sahibzada, and S. Weimer, “An introduction to the good, the bad, & the ugly face recognition challenge problem.” [online]
<http://www.nist.gov/itl/iad/ig/upload/05771424.pdf>, 2011.
ISBN 978-1-4244-9140-7.

Appendix A

CD contents

Attached CD contains:

- application source code and libraries
- Doxygen documentation
- manual with basic instructions
- database containing 50 3D models acquired with Kinect
- set of manually set features on 50 depth maps used for testing
- L^AT_EX and PDF versions of this document