Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student:	Andrey Stanislavovich M r a m o r n o v
Study programme:	Open Informatics
Specialisation:	Computer Vision and Image Processing
Title of Diploma Thesis:	Muscle Tension Analysis Based Facial Motion Capture

Guidelines:

Realize a literature search concerning approaches where a skin surface is visually analyzed to get information about corresponding muscle tension. Focus on cases of a human face. Based on results of this search, try to design application which uses one of these approaches to derive facial motions from muscle tension analysis. Implement the application using programming environment based on C++ alternatively combined with python. You can use animation tool developed for modeller Blender during previous project.
Propose a set of testing methods and comment your final results.

Bibliography/Sources:

[1] Byoungwon Choe and Hanook Lee and Hyeong-Seok Ko: Performance-Driven Muscle Based Facial Animation, The Journal of Vizualization and Computer Animation, 12(2): 67-79, John Wiley & Sons, Ltd., 2001.
[2] Byoungwon Choe and Hyeong-Seok Ko: Analysis and synthesis of facial expressions with hand-generated muscle actuation basis. In ACM SIGGRAPH 2005 Courses (SIGGRAPH '05), John Fujii (Ed.). ACM, New York, NY, USA, , Article 15.

Diploma Thesis Supervisor: Ing. Roman Berka, Ph.D.

Valid until: the end of the summer semester of academic year 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, February 16, 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Andrey Stanislavovich M r a m o r n o v
Studijní program:	Otevřená informatika (magisterský)
Obor:	Počítačové vidění a digitální obraz
Název tématu:	Snímání pohybu obličeje založené na analýze napětí svalů

Pokyny pro vypracování:

Proveďte rešerši přístupů, kde je na základě vizuální analýzy povrchu kůže zkoumáno svalové napětí. Zaměřte se na studium povrchu lidské tváře.
Na základě získaných informací navrhněte aplikaci, která s využitím některé z nastudovaných metod bude odvozovat pohyby částí obličeje pro účely záznamu a mapování na model tváře.
Implementujte svůj návrh v podobě aplikace v prostředí jazyka C++, alternativně python.
V průběhu implementace je možné využít animační nástroj vyvinutý pro modelář Blender v rámci předchozího projektu.
Navrhněte rovněž sadu testovacích metod a na nákladě testů pak vhodně okomentujte dosažené výsledky práce.

Seznam odborné literatury:

[1] Byoungwon Choe and Hanook Lee and Hyeong-Seok Ko: Performance-Driven Muscle Based Facial Animation, The Journal of Vizualization and Computer Animation, 12(2): 67-79, John Wiley & Sons, Ltd., 2001.
[2] Byoungwon Choe and Hyeong-Seok Ko: Analysis and synthesis of facial expressions with hand-generated muscle actuation basis. In ACM SIGGRAPH 2005 Courses (SIGGRAPH '05), John Fujii (Ed.). ACM, New York, NY, USA, , Article 15.

Vedoucí diplomové práce: Ing. Roman Berka, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
vedoucí katedry	**děkan**

V Praze dne 16. 2. 2015

master's thesis

Muscle Tension Analysis Based Facial Motion Capture

Andrey Mramornov

May 2015

Supervisor: Roman Berka

Czech Technical University in Prague
Faculty of Electrical Engineering, Katedra kybernetiky

Acknowledgement

I would like to thank my supervisor, Roman Berka, for support in writing this thesis. I am grateful for my family and friends for supporting me during my studies.

Declaration

Abstract

The paper presents description of muscle model for animation used to do motion capture worked out as a master thesis project. The model allows one to animate virtual character by small number of parameters, and capture procedure makes possible animation based on performance. Thesis starts with brief introduction, followed by historical overview and background information. The theoretical description of model and performance tracking procedure comprise core of the paper, consequently supported by implementation details and experiments. The paper ends with conclusion highlighting possible future direction of improvements for the model.

Keywords

muscle animation model; facial animation; motion capture; single camera tracking

Contents

Abbreviations

FACS	Facial Action Coding System
GPL	General Public License, a popular free software license
LK	Lukas-Kanade optical flow method
PC	Personal Computer
PNP	Perspective-n-Point, a technique in computer vision which estimates a camera pose from given n 3D points and their projections in an image
RAM	Random Access Memory
RGBD	in context of camera - camera which is capable of measuring distance per pixel as well as capturing color data
UI	User Interface

1 Introduction

This work is a master thesis written by Mramornov Andrey during study on master program Open Informatics in Faculty of Electrical Engineering of Czech Technical University in Prague. It deals with muscle representation of the facial animation. It is a good way for solving many related tasks, for example animating virtual character by hand or based on performance, face expression analysis. It is considered good for these tasks because it is a natural representation which has understandable effect on face. The work itself is focused on building muscle model of face and finding its parameters from actor performance recorded on a single camera without any depth information. Such setup is different from many of state of the art methods which use either multiple cameras [1] or cameras with depth information [2], [3]. It is generally cheaper and easier to use single camera setup and sometimes 2D video is the only available data, for example in forensics or on mobile devices.

The paper is structured as follows. This chapter gives overview of previous work followed by definition of thesis goals. Second chapter gives background information for the reader to understand further description better. Subsequent chapters describe theoretical muscle model, derivation of tracking procedure, implementation details of proposed model, tracking method, experiment procedure and discuss their results. Last chapter summarizes the study, provides conclusions and lists possible future improvements.

1.1 State of the art

Computer facial models have a long history, first parametric model of face was presented by Parke in 1974 [4]. In 1981 Platt and Badler in their work created first muscle based facial animation, they represented muscles and skin as different layers and used springs to simulate muscles [5]. In 1987 Waters presented muscle model for facial animation which, in contrast to model of Platt and Badler, did not distinguished muscle and skin layers [6]. Wilhelms in 1994 [7] proposed an ellipsoid muscle model which she and Gelder in 1997 generalized into deformable cylinder model [8]. Scheepers et al. also took Wilhelms ellipsoid model as basis for their multi-belly muscle model which represent each muscle as a number of ellipsoids instead of three as in Wilhelms model [9]. In 1998 Nedel and Thalmann modeled facial animation as set of particles connected by springs [10].

Facial motion capture was developed in parallel with animation models. Williams in 1990 presented first facial motion tracking approach [11]. In 1996 Essa et al. used muscle model with physically-based face model to capture performance [12]. Blanz and Vetter in 1999 represented faces and facial animations by blendshapes [13]. They then used probabilistic approach to find optimal model parameters for a given image. Chai et al. in 2003 proposed to use 2d video motion data to extract 3d landmark positions and retarget them to virtual avatar [14].

At present, majority of the methods use blendshapes as a model for facial animation along with 3D information acquired from different sources: 3D reconstruction from multi-camera array, depth cameras, 3d-scanners. Blendshapes usually represent FACS (Facial Action Coding System) action units.

In a study of Weise RGBD data from Microsoft Kinect camera with blendshape model and a probabilistic animation model is used to produce realistic animation [2]. Such model, however requires training of user-specific blendshapes expression model which limits use of such approach. Bouaziz improved such model by combining tracking with user-specific expression learning which removes training or calibration steps [3].

Approach presented by Bradley uses array of calibrated cameras with uniform lightning to reconstruct noise-free dense 3D model [1]. This model with texture information is then tracked to further enhance resulting model and texture. Their approach produces a very detailed models preserving even small texture details, however requires a complex setup and does not present any way to transfer animation to virtual character.

In paper "Facewarehouse" authors propose to train 3D shape regressor for each user and use it to infer 3D data from 2D video, according to authors experiments this approach outperforms RGBD-based technique from Weise [15]. Weng et al. reduced computational costs of that algorithm by directly regressing head poses and expression coefficients which allowed them to produce real-time performance based facial animation on mobile device [16]. However, both approaches still require training step for each user.

Modification of such regression approach called Displaced Dynamic Expression was proposed by Cao et al. [17]. Its improvement was removal of training step while keeping quality. According to authors experiments their solution, which only uses 2D video information, achieve same level of robustness, accuracy and efficiency in face tracking as technique using RGBD information Weise.

To sum up a variety of animation models were proposed, but for motion capture most authors use blendshapes. In capture new methods were proposed after widespread of RGBD cameras, large focus is on building fast and robust methods to be used in limited and noisy environment for wide use on consumer devices.

1.2 Goals

As it was mentioned above, most of state of the art methods use blendshapes to model facial animation. Blendshapes serve well for this purpose, however they have a number of drawbacks.

First of all blendshapes set is fixed and in most methods can not be changed without retraining. Change to the set can be required, for example, if current set is not able to produce some desired animation. This problem is not that severe if blendshapes represent FACS action units because they are known to describe almost all possible face expressions, but can still be an issue.

Second, blendshapes should be made either by hand which is tedious, or by learning from some input set which can be erroneous or require special setups.

Finally blendshapes do not directly correspond to physiological processes and thus their effect on face may be not that understandable as in muscle based approaches.

To cope with these problems a muscle based parametric model which has a physiological basis will be proposed, along with technique to derive model parameters from user performance. To be more precise goals are:

- Propose muscle model for facial animation which is capable of producing different facial emotions.
- Such model should be easily retargetable from one virtual character to another.
- Design and implement approach for tracking of model parameters from user performance captured on a single RGB camera.

- Tracking technique should be as robust as possible to user and camera position changes.
- Tracking should be made as fast as possible, desirable be able to work real-time.
- It should require small amount of hand input.
- Implement visualization and editing tool for tracked data so it can be used in render/game.

To sum up, to solve drawbacks of blendshape animation model, in this work we will propose a muscle model for facial animation as well as approach to track such model parameters based on performance.

2 Background

This section gives reader required background knowledge for understanding this work.

2.1 Anatomy of the head

Human head has a lot of different parts in it, we will focus only on those parts which deal with facial animation, that is skull, skin and muscles.

2.1.1 Skull

Human skull (figure 1) is a bone structure which forms shape of the head and serves as rigid basis for skin and muscles. It consist of three bone areas: skull proper (*neurocranium*), facial area (*viscerocranium*) and associated bones (6 ear bones and the *hyoid*). Skull also includes a single joint: the mandible, which can open and slightly move to left and right. Mandible opening is important in animation since it moves large part of face.

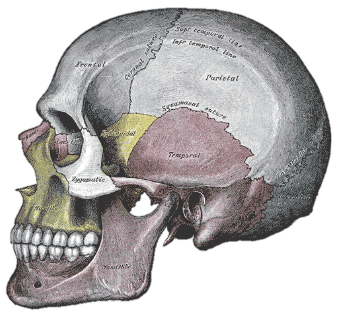


Figure 1 Skull

In this work we will not represent skull as a part of the model, but we will use the fact that it is rigid to fix muscles positions.

2.1.2 Skin

Skin (figure 2) covers body and protects deeper layers of human body from foreign organisms, injuries and drying. It also serves as tactile sensor. Skin consist of two parts: epidermis and dermis, hypodermis lying below them is usually not seen as part of the skin.

Epidermis - the first layer of skin is important for us since it is the visible part and determines appearance of human. It consist of several layers, where bottom (closest to the body) one *stratum basale* produces cells and push them to the upper levels. These cells reach upper levels dead, they renew every two weeks making soft, glowing appearance [18]. Skin has following properties important for animation:

- Non-linearity on strain - skin response to low strains is linear, on average strains it loses linearity and on high strains again becomes linear [19].

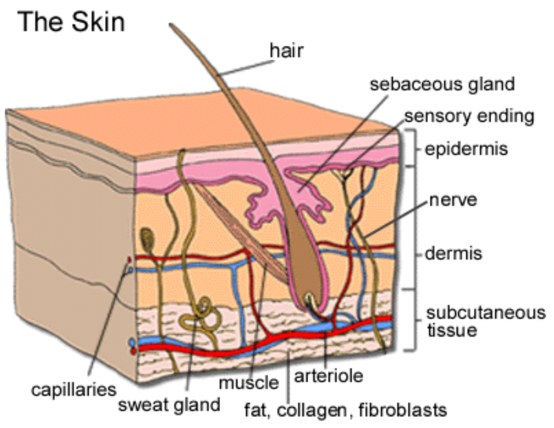


Figure 2 Skin model

- Plasticity which means that skin shows combination of time-dependent fluid and solid object properties. For example it recovers with time after strain.
- Quasi-Incompressible - some parts of skin are incompressible which means that they do not change volume after deformations. This property results, for example, in wrinkles.

2.1.3 Muscles

Muscles serve for moving different parts of body, they connect to skin, bones, ligaments and cartilages. Connection can be direct or via tendon. Muscles fill almost all space between bones and skin and weight about half of the whole body of an adult. Muscle fibers are usually with one side connected to bone (that is origin of muscle) and with other to skin (insertion of muscle). When contracting, muscles shorten and since origin is connected to bone which is rigid and thus can not move, insertion is pulled towards origin.

Muscles are composed of fascicles which are in turn composed of muscle fibers. All fibers in fascicle are parallel to each other, different types of muscles are formed by different alignment of fascicles: There are different types of muscles (figure 3) [20]:

- Parallel or fusiform muscles have fibers in parallel directions.
- Convergent muscles converge on the insertion point to maximize force of muscle contraction.
- Pennate muscles have many fibers per unit area.
- Circular muscles fibers surround opening to act as a sphincter (close opening).

Face has 25 muscles (figure 4), we will list muscles important for animation from each group:

- Scalp - the only important muscle for animation is *frontalis* which moves eyebrows up and creates wrinkles on forehead.
- Mouth - almost all muscles from that group significantly affects animation. Notable one is orbicularis oris which is a circular muscle and serve for closing of the mouth.
- Eye - muscles of that group closes eyelids and move eyeball which is an important part of facial animation, however we will not use represent their action using muscles, but simulate through other means.
- Neck - the only muscle of that group slightly moves corners of the mouth, we will use muscles from mouth to simulate that movement.

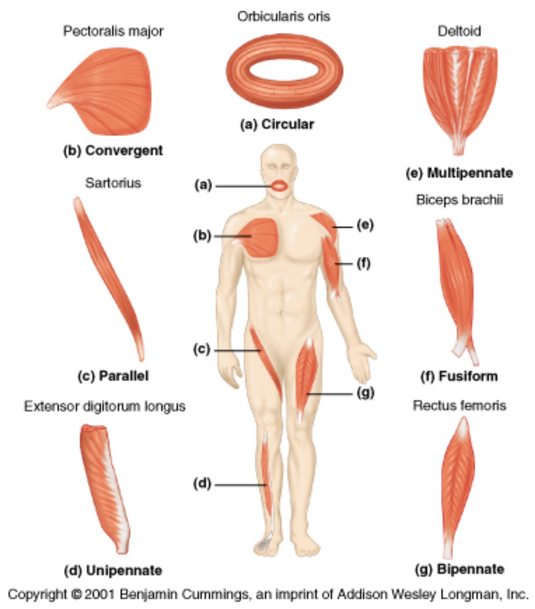


Figure 3 Muscle types [20]

- Nose - this group has a small effect on nostrils, we will not use it our model.
- Muscles of mastication - main effect of these muscles is opening of mandible.

2.2 Optical flow

In this work we use optical flow to track facial landmarks between frames. In this section we briefly overview optical flow estimation task and selected method that will be used in implementation.

Optical flow is the pattern of apparent motion of objects in visual scene caused by relative motion between observer and scene [22]. Given an ordered sequence of images one can estimate displacements of image objects between pairs of frames. Since for general case objects present in scene are unknown, general optical flow methods try to calculate motion vector at each voxel position. Formally, a voxel at location (x, y, t) with intensity $I(x, y, t)$ modified by $\Delta x, \Delta y, \Delta t$ between two image frames, following constraint should hold:

$$I(x, y, y) = I(x + \Delta x, y + \Delta y, z + \Delta z)$$

Assuming that movement was small, using Taylor expansion we can gen:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t$$

From these equations it follows that:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0$$

Dividing by Δt we can get:

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0$$

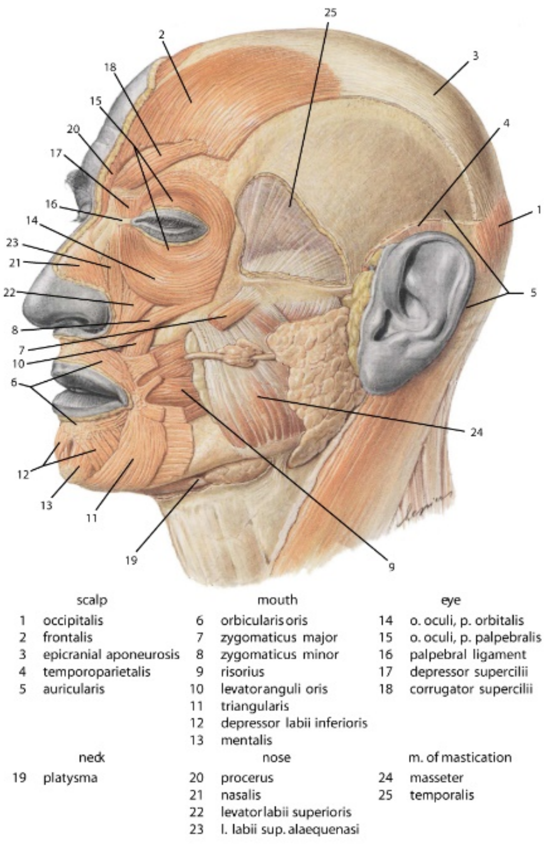


Figure 4 Face muscles [21]

Where V_x, V_y are optical flow velocity components. Replacing $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, $\frac{\partial I}{\partial t}$ by I_x, I_y, I_t we get:

$$I_x V_x + I_y V_y = -I_t$$

Which is a single equation with two unknowns. Different optical flow methods then add constraints to be able to solve such equation [23] [24].

2.2.1 Lukas-Kanade method

In this work we will use Lukas-Kanade method. It assumes that the displacement of the image contents between two frames is small and almost constant within some neighborhood of the pixel under consideration. Thus the optical flow equation can be assumed to hold for all pixels within a window centered at p. This adds following constraints:

$$\begin{aligned}
I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1)\\
I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2)\\
&\vdots\\
I_x(q_n)V_x + I_y(q_n)V_y &= -I_t(q_n)
\end{aligned}$$

where $q_1, q_2, \ldots, q_n$ are the pixels inside the window, and $I_x(q_i), I_y(q_i), I_t(q_i)$ are the partial derivatives of the image I with respect to position x, y and time t, evaluated at the point q_i and at the current time. These constraints can be rewritten in matrix form as

$$Av = b$$

where $A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}$, $\quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}$, $\quad$ and $\quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$ Now we have more constraints than unknowns (given that window contains more than 2 points) and can be over determined. To end up with some solution, Lucas-Kanade method uses least squares principle obtaining solution:

$$v = (A^T A)^{-1} A^T b$$

Since pixel closer to the current point may be more correlated then one on border of the window, weight version of least squares can be used:

$$v = (A^T W A)^{-1} A^T W b$$

Where W is a diagonal matrix of weights [25].

3 Theoretical part

3.1 Muscle model

Proposed muscle model which is used as the basis for animation and performance analysis consist of three different types of muscles:

- Linear muscle
- Parallel muscle
- Sphincter muscle

All muscle types are animated same way - by multiplying disposition of vertex by contraction factor ranging from 0 to 1:

$$P_{new} = P + contraction * (P' - P)$$

Where P_new is new position of muscle, P is original vertex position, P' is position of vertex P deformed by muscle, *contraction* is contraction of the muscle.

If more then one muscle affects some point P their effects sum:

$$P_{new} = P + \sum_{i \in M} contraction_i * (P'_i - P)$$

where M is mucsle set, $contraction_i$ is contraction of muscle i, P'_i is position of vertex P deformed by muscle i.

Each type of muscle is described in detail in following subsections.

3.1.1 Linear muscle

Linear muscle model was first described by Waters [6] and is defined by following equations:

$$A = cos(\frac{\mu}{\pi} * \frac{\pi}{2}) \tag{1}$$

$$R = \begin{cases} cos(\frac{D-R_s}{R_f-R_s} * \frac{\pi}{2}) & ifD > R_s \\ cos(\frac{1-D}{R_s} * \frac{\pi}{2}) & ifD \leq R_s \end{cases} \tag{2}$$

$$P' = P + K * A * R * (\overrightarrow{PV_1})_{norm} \tag{3}$$

Where (see figure 5):

- V_1 is muscle origin
- V_2 is muscle insertion
- P is original vertex position
- P' is deformed vertex position
- R_s is start of effect distance
- R_f is end of effect distance
- μ is angle between $\overrightarrow{PV_1}$ and $\overrightarrow{V_1V_2}$
- K is elasticity coefficient

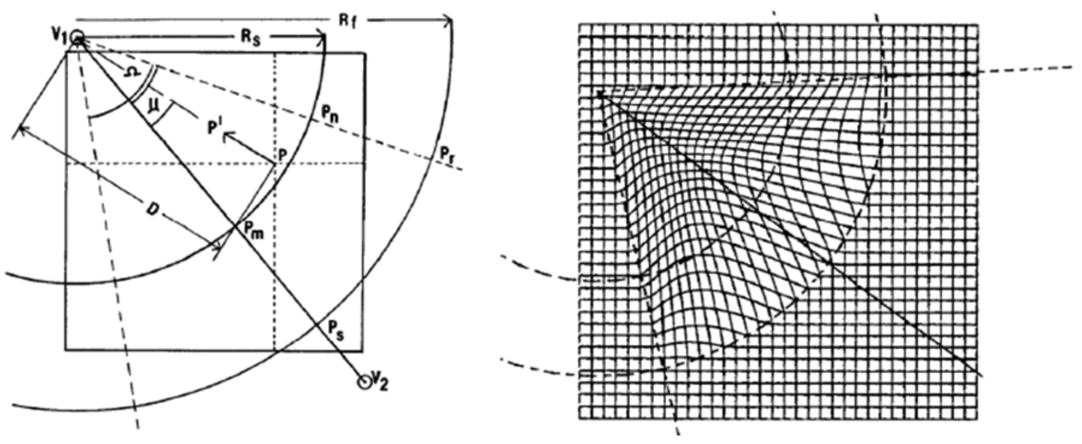


Figure 5 Linear muscle model

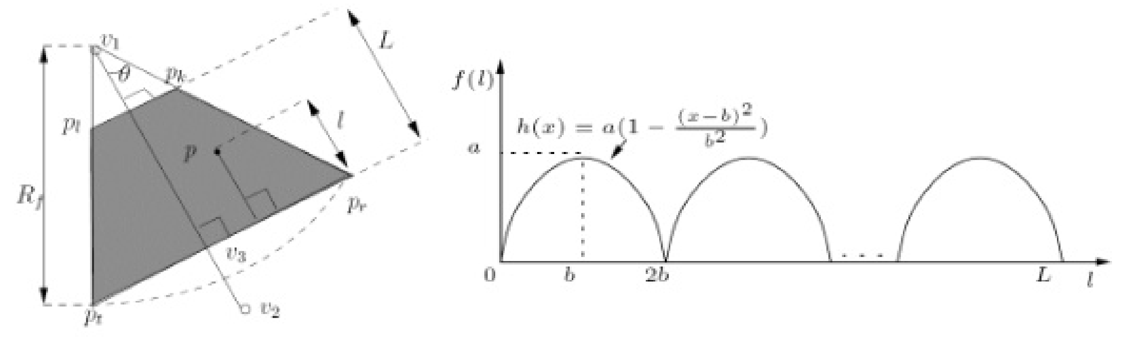


Figure 6 Wrinkles

- $D = |\overrightarrow{PV_1}|$

However such model does not produce wrinkles and thus does not look very realistic. An extension proposed by Bui [26] adds them to the model. In this extension wrinkles are created by raising vertices in effect area $p_l p_k p_r p_t$ (before applying muscle) to a defined height H (see figure 6):

$$P' = P + H * u(l) * \overrightarrow{N} \tag{4}$$

$$u(l) = l - \left\lfloor \frac{l}{b} \right\rfloor b \tag{5}$$

$$b = \frac{L}{2N_w} \tag{6}$$

$$L = \frac{3}{4} R_f cos(\theta) \tag{7}$$

Where N_w is desired number of wrinkles, N is vertex normal at point P. You can see effect of activating linear muscles with two wrinkles on figure 7.

3.1.2 Parallel muscle

Parallel muscle is used to simulate same physiological muscle which may be seen as set of thin linear muscles with very small μ. When contracting parallel muscle moves all

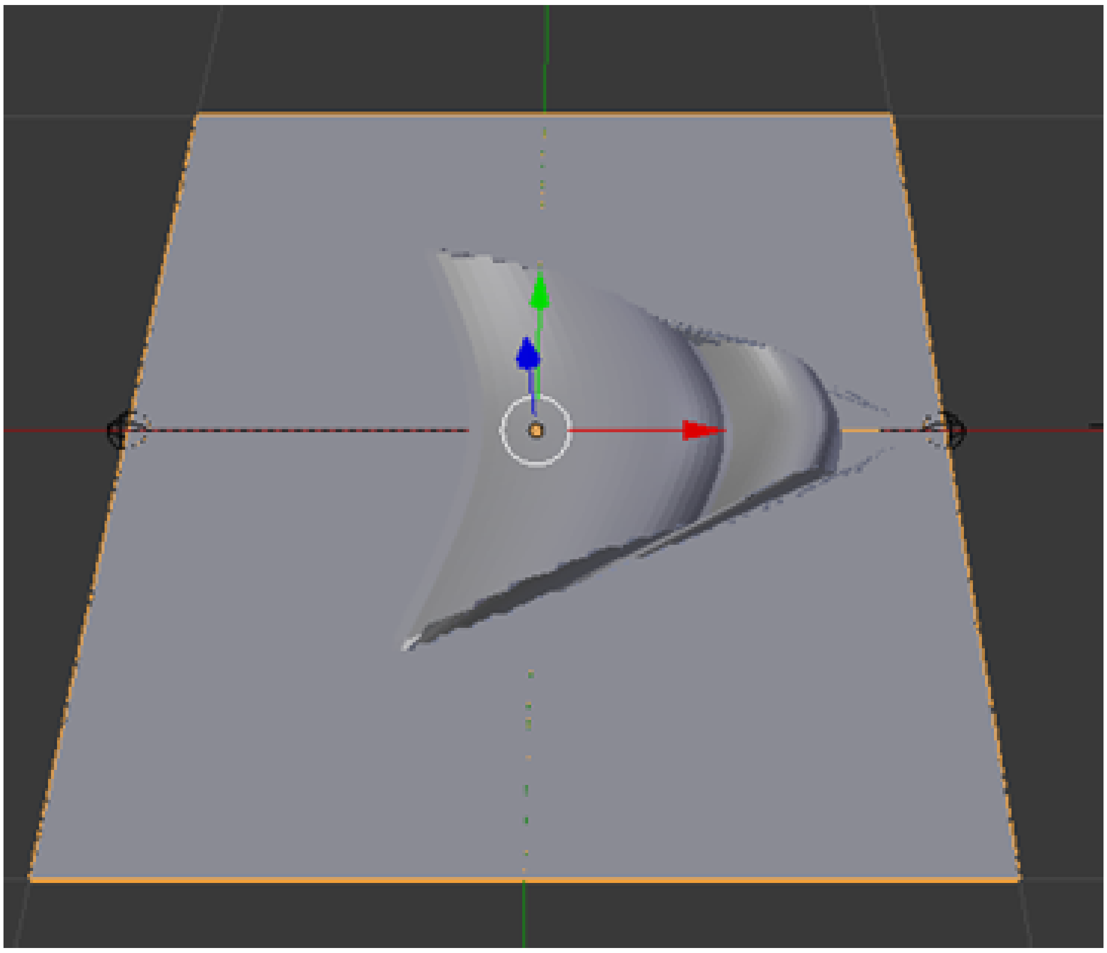


Figure 7 Wrinkles effect

vertices on same distance from line $\vec{oc}$ towards this line on the same amount (see figure 8).

In model parallel muscle is defined by three control points: origin o, insertion i and control point c. These points define muscle plain, however muscle is not absolutely thin so there is a *height* parameter (see figure 8). Each vertex P in zone of effect has following new coordinates P':

$$P' = P + K * R * (\vec{oi})_{norm} \tag{8}$$

$$R = \begin{cases} cos(\frac{D-R_s}{R_f-R_s} * \frac{\pi}{2}) & if D > R_s \\ cos(\frac{1-D}{R_s} * \frac{\pi}{2}) & if D \leq R_s \end{cases} \tag{9}$$

Where:

- R_s is start of effect distance
- R_f is end of effect distance
- K is elasticity coefficient
- $D = \vec{oP} \cdot \vec{oi}$ is distance of point from line $\vec{oc}$

3.1.3 Sphincter muscle

Even though most of the facial muscles can be described by linear and parallel muscles, Orbicularis Oris - muscle around mouth - works differently and thus requires special

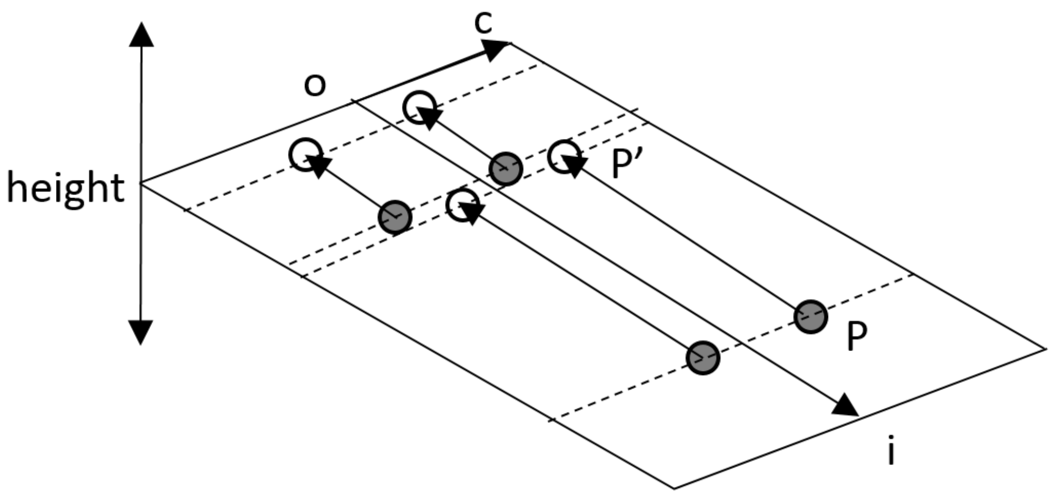


Figure 8 Parallel muscle

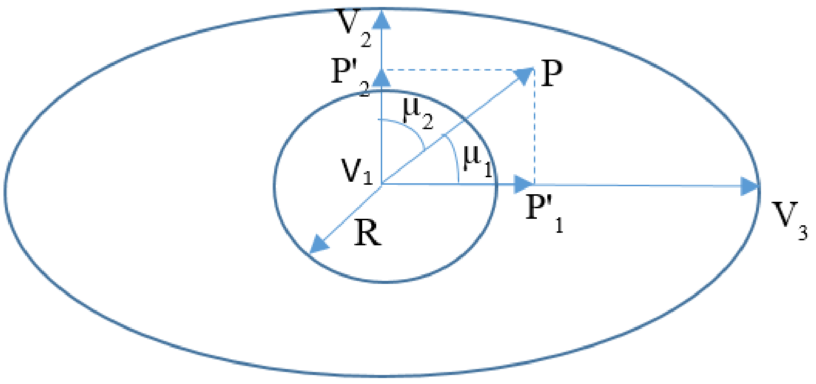


Figure 9 Sphincter muscle

type in model. With contraction this muscle rounds lips and move them closer to each other. To simulate that behavior *Sphincter* muscle type was created. It is defined by three control points, one in the center of mouth, one in the middle of upper lip and last one in the right corner of the mouth. Last 2 points should be orthogonal. These points define an ellipse which approximate mouth shape. Sphincter muscle moves vertices inside ellipse so that they form circle of defined radius raised to the given height H. New vertex position is given by (see figure 9):

$$P' = P + D_1 + D_2 + D_3 \tag{10}$$

$$D_1 = cos(\mu_1) * \overrightarrow{P_1'V_1}_{norm} * (1 - \frac{R}{|\overrightarrow{V1V3}|}) \tag{11}$$

$$D_2 = cos(\mu_2) * \overrightarrow{P_2'V_1}_{norm} * (1 - \frac{R}{|\overrightarrow{V1V2}|}) \tag{12}$$

$$D_3 = H * min(\frac{R}{max(|\overrightarrow{V1V3}|, |\overrightarrow{V1V2}|)}, (1 - \frac{D^2}{R_f * R_s})) * \overrightarrow{N} \tag{13}$$

Where N is a normal to the plane $V_1V_2V_3$ in direction out of the head.

3.2 Face model

Described muscle model is applicable to any polygonal face model, but during this work only Makehuman [27] face model is used. Makehuman model allows one to model a wide variety of human bodies and faces using exactly same mesh topology, which effectively means that same vertices on different meshes represent same body entities, for example vertex #5178 is always a nose tip. This makes it possible to retarget muscle set from one model to another using simple algorithm (see algorithm 1). This algorithm uses nearest mesh vertex as attachment point of the muscle. Since muscle can be lying off the vertex by small displacements, it is also used during retargeting. This may be a problem if size of the mesh or edges changes significantly, however that never happened during experiments.

```
input  : Muscle set $M$, source mesh $S$, target mesh $T$
output: Retargeted muscle set $M_2$

foreach muscle $m$ in $M$ do
    $m' \leftarrow$ copy of $m$;
    foreach control point $p$ of $m$ do
        $v \leftarrow$ closest vertex of $S$ to $p$;
        $v_2 \leftarrow$ position of vertex $v$ in $T$;
        $p' \leftarrow v_2 + p - v$;
    end
    $M_2 \leftarrow Union(M_2, \{m'\})$;
end
```

Algorithm 1: Retargeting algorithm

Makehuman model has also a skeleton which can be used to rig the body. We only use *jaw* bone from that skeleton since jaw opening is an important part of facial animation. Jaw opening was simulated in model by rotating *jaw* bone over X axis.

3.3 Tracking

Now, having muscle and face models defined, general model parameters tracking procedure can be introduced.

We define $I(p, x, M, P)$ as image of 3D point p affected by muscle set M with contraction values x and projected by camera with projection matrix P. Then having two images $I_{p,1} = I(p, x_1, M, P_1)$ and $I_{p,2} = I(p, x_2, M, P_2)$ for each point p from set of points D task of the tracking is to find such x_2 and P_2 that total error $E = \sum_{p \in D} ||I_{p,1} - I_{p,2}||$ is minimum and x_2, P_2 are feasible.

To ease that minimization task, estimation of P_2 and x_2 is separated. Assuming that camera is calibrated, P_2 can found by solving PNP (Persepctive-n-Point) task for set of points of face that is not affected by muscles at all (for example. nose tip, corners of eyes).

Having this estimated P_2, optimal x_2 can be found by bounded gradient-descent method. To use gradient-descent one can either approximate gradient numerically which would require more computations, or compute it analytically. Since muscle displacement functions are linear to contraction we can derive first-order derivative. First we define image function

$$P_2 \begin{bmatrix} p_x + x_2 \cdot C_X(M,p) \\ p_y + x_2 \cdot C_Y(M,p) \\ p_z + x_2 \cdot C_Z(M,p) \\ 1 \end{bmatrix} = \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix}$$

$$I(p, x_2, M, P_2) = \begin{bmatrix} \frac{A_x}{A_z} \\ \frac{A_y}{A_z} \end{bmatrix}$$

Where C_x, C_y and C_z are muscle displacements coefficients on each axis for muscle set M and point p. Values A_x, A_y and A_z are used to shorten further derivation. Now we can define error for single point

$$E_p = ||I_{p,1} - I(p, x_2, M, P_2)||$$
$$E_p = E_{p,x}^2 + E_{p,y}^2$$
$$E_{p,x}^2 = \left(I_{p,1,x} - \frac{A_x}{A_z}\right)^2$$
$$E_{p,y}^2 = \left(I_{p,1,y} - \frac{A_y}{A_z}\right)^2$$

And we can find derivative

$$P_2 \begin{bmatrix} C_x(M,p) \\ C_y(M,p) \\ C_z(M,p) \\ 0 \end{bmatrix} = \begin{bmatrix} C'_X(M,p) \\ C'_Y(M,p) \\ C'_Z(M,p) \end{bmatrix}$$

$$\frac{\partial}{\partial x} E_{p,x}^2 = 2\left(\frac{C'_Z(M,p) * A_x}{A_z^2} - \frac{C'_X(M,p)}{A_z}\right)\left(I_{p,1,x} - \frac{A_x}{A_z}\right)$$
$$\frac{\partial}{\partial x} E_{p,y}^2 = 2\left(\frac{C'_Z(M,p) * A_y}{A_z^2} - \frac{C'_Y(M,p)}{A_z}\right)\left(I_{p,1,y} - \frac{A_y}{A_z}\right)$$
$$\frac{\partial}{\partial x} E_p = \frac{\partial}{\partial x} E_{p,x}^2 + \frac{\partial}{\partial x} E_{p,y}^2$$
$$\frac{\partial}{\partial x} E = \sum_{p \in D} \frac{\partial}{\partial x} E_p$$

Such formulation could be used if face mesh would match closely real-world face of the user, otherwise absolute difference in pixels would be result of inaccurate 3D points, not muscle contraction values, and thus solution would be wrong. Constructing close face mesh is not always possible so some solution is required.

Using fact that input is a video sequence we propose to minimize difference in image positions between two consecutive frames instead of absolute image distances. Since such minimization can be imperfect, resulting error from previous iterations E_p^{prev}

should be respected. Formally, having two images

$$I_{p',1} = I(p', x_1, M, P_1)$$
$$I_{p',2} = I(p', x_2, M, P_2)$$

for each point p from set of points D where p' is unknown real world 3D point corresponding to the same face keypoint p, previous muscle state x_1 and projection matrix P_1 are known, task of the tracking is to find such x_2 and P_2 that total error

$$E = \sum_{p \in D} ||(I_{p',1} - I_{p',2} + E_p^{prev}) - (I(p, x_1, M, P_1) - I(p, x_2, M, P_2))||$$

is minimum and x_2, P_2 are feasible.

That modified task can be solved same way, error and derivative change slightly

$$E_p = ||(I_{p',1} - I_{p',2} + E_p^{prev}) - (I(p, x_1, M, P_1) - I(p, x_2, M, P_2))||$$
$$E_p = E_{p,x}^2 + E_{p,y}^2$$
$$E_{p,x}^2 = \left((I_{p',1,x} - I_{p',2,x} + E_{p,x}^{prev}) - (I(p, x_1, M, P_1)_x - \frac{A_x}{A_z})\right)^2$$
$$E_{p,y}^2 = \left((I_{p',1,y} - I_{p',2,y} + E_{p,y}^{prev}) - (I(p, x_1, M, P_1)_y - \frac{A_y}{A_z})\right)^2$$

$$\frac{\partial}{\partial x} E_{p,x}^2 = 2\left(\frac{C'_Z(M,p) * A_x}{A_z^2} - \frac{C'_X(M,p)}{A_z}\right)$$
$$\left((I_{p',1,x} - I_{p',2,x} + E_{p,x}^{prev}) - (I(p, x_1, M, P_1)_x - \frac{A_x}{A_z})\right)$$
$$\frac{\partial}{\partial x} E_{p,y}^2 = 2\left(\frac{C'_Z(M,p) * A_y}{A_z^2} - \frac{C'_Y(M,p)}{A_z}\right)$$
$$\left((I_{p',1,y} - I_{p',2,y} + E_{p,y}^{prev}) - (I(p, x_1, M, P_1)_y - \frac{A_y}{A_z})\right)$$

Here A_x, A_y, A_z, C'_X, C'_Y and C'_Z are same as in previous derivation.

As one can notice, x_1 is required to use such method, to get an initial value we assume that user starts interaction with neutral face which is when x_1 is zero in each element.

There is still one part missing in that solution, that is effect of jaw opening which has a big effect on face. To simplify solution, jaw opening is specified by single parameter - rotation of *jaw* bone in face model skeleton around X axis. This model parameter can be as well as projection matrix be estimated separately from muscle contractions by minimizing difference error of some static point. Formally, we define $J(p, xrot, b, P)$ as image of 3D point p modified by jaw rotation $xrot$ around point b and projected by matrix P. Then given two images $J_1 = (p', xrot_1, b, P_1)$ and $J_2(p', xrot_2, b, P_2)$ we need to find such $xrot_2$ that

$$E_{jaw} = ||(J_1 - J_2 + E_{jaw}^{prev}) - (J(p, xrot_1, b, P_1) - J(p, xrot_2, b, P_2))||$$

Here the same method is used to minimize difference between frames instead of absolute positions. Image function is defined as

$$P\begin{bmatrix} b+\begin{bmatrix}1 & 0 & 0\\ 0 & cos(\frac{xrot}{2}) & -sin(\frac{xrot}{2})\\ 0 & sin(\frac{xrot}{2}) & cos(\frac{xrot}{2})\end{bmatrix} * (p-b) \\ 1 \end{bmatrix} = \begin{bmatrix}A_x\\ A_y\\ A_z\end{bmatrix}$$

$$J(p, xrot, b, P) = \begin{bmatrix}\frac{A_x}{A_z}\\ \frac{A_y}{A_z}\end{bmatrix}$$

which can be derived to:

$$p-b = \begin{bmatrix}d_x\\ d_y\\ d_z\end{bmatrix}$$

$$P = \begin{bmatrix}p_1^T & p_2^T & p_3^T & p_4^T\end{bmatrix}$$

$$\frac{1}{2}\begin{bmatrix}p_2^T & p_3^T\end{bmatrix}\begin{bmatrix}-d_y * sin\left(\frac{x}{2}\right) - d_z * cos\left(\frac{x}{2}\right)\\ d_y * cos\left(\frac{x}{2}\right) - d_z * sin\left(\frac{x}{2}\right)\end{bmatrix} = \begin{bmatrix}A'_x\\ A'_y\\ A'_z\end{bmatrix}$$

$$E_{jaw} = E_{jaw,x}^2 + E_{jaw,y}^2$$

$$\frac{\partial}{\partial x}E_{jaw,x}^2 = 2\left(\frac{A'_z * A_x}{A_z^2} - \frac{A'_x}{A_z}\right)\left((J_{1,x} - J_{2,x} + E_{jaw,x}^{prev}) - (J(p, xrot_1, b, P_1)_x - \frac{A_x}{A_z})\right)$$

$$\frac{\partial}{\partial x}E_{jaw,y}^2 = 2\left(\frac{A'_z * A_y}{A_z^2} - \frac{A'_y}{A_z}\right)\left((J_{1,y} - J_{2,y} + E_{jaw,y}^{prev}) - (J(p, xrot_1, b, P_1)_y - \frac{A_y}{A_z})\right)$$

$$\frac{\partial}{\partial x}E_{jaw} = \frac{\partial}{\partial x}E_{jaw,x}^2 + \frac{\partial}{\partial x}E_{jaw,y}^2$$

Same way as for the muscle contraction values, we assume that jaw is initially closed ($xrot_1 = 0$).

These error functions and derivatives can be plugged into some minimization method. We use truncated Newton Conjugate-Gradient [28] which bounds variable by limiting step size so that variable would never exceed given bounds.

We can now use such minimization task to find optimal model parameters by performance. However performance in our setup is just video frames set, while error functions require images of points I_2, J_2. We use optical flow to estimate new positions of points I_1 and J_2.

3.4 Workflow

As all parts of tracking process were described, we can now present workflow of the application on figure 10. Given initial positions of keypoints in first frame, for each next frame we compute their new positions by optical flow, solve PNP task to get P_2, then minimize E_{jaw} and E given x_1, x_2, P_1, P_2 using bounded gradient descend. P_2 and x_2 are then used to display virtual avatar and replace x_1 and P_1 for the next frame.

Display of the avatar can be replaced by any other consumer of model parameters, for example one can classify model parameters to several expression classes and thus have sentiment analysis application, or model parameters can be used to detect visemes on face to assist speech-to-text.

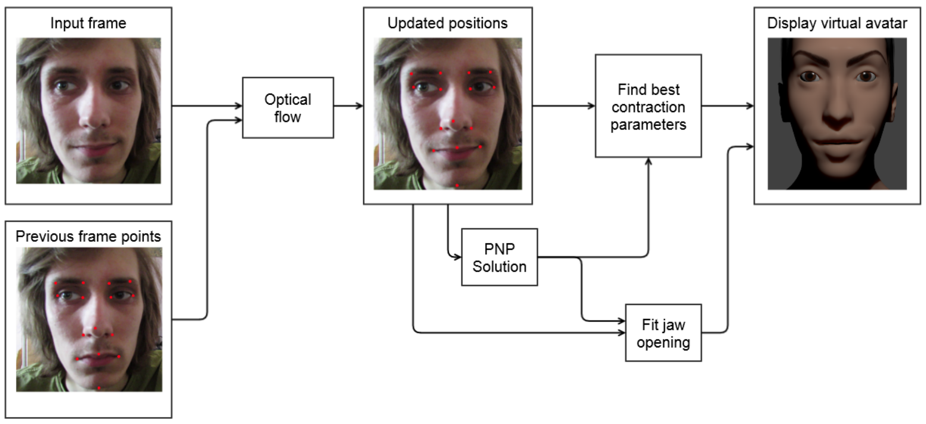


Figure 10 Workflow

4 Implementation

Proposed model was implemented in two parts:

- Visualization/Editing tool - provides user interface to edit, retarget and animate meshes using muscles sets.
- Model parameters tracker - handles all steps required to track model parameters following workflow described in 3.4.

Each of them is described in detail in following sections.

4.1 Visualization/Editing tool

We used Blender - an open source (GPL) 3D computer graphics software which can serve for 3D modeling, rendering and some other 3D related tasks, as a basis for Visualization/Editing tool because it already has integration with Makehuman project which allows simple importing of makehuman mesh. It also has powerful editing, rendering and animation tools which can be used instead of recreating them in custom application. Open source nature of Blender will also allow to share created tools with other users of Blender.

Blender has support for python addons, however they are limited in functions, for example it is not possible to create new type of mesh modifier which is a straightfoward way of representing muscle animation. Unfortunately there is no binary addons support in Blender, so adding some functionality requires modification of Blender source code.

We added to Blender a new object type called *Muscle* which purpose is to be placed in 3D scene to define one control point of some muscle and also store muscle parameters. We also added new modifier type for meshes called *Muscle modifier* which is used with muscle objects to deform mesh.

4.1.1 Muscle object

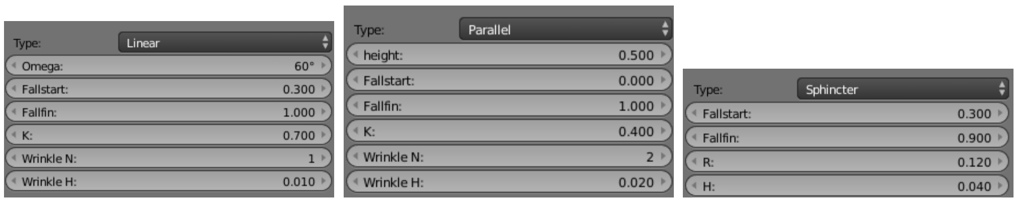
Muscle object is based on *ID* structure which represents basic properties of object in Blender. It contains parameters for all types muscles and type:

```
typedef struct Muscle {
        //blender data
        ID id;
        struct AnimData *adt;

        //type of the muscle
        short type;

        //padding required by blender
        char pad[2];

        //linear muscle parameters
        float omega;
```



Type: Linear
Omega: 60°
Fallstart: 0.300
Fallfin: 1.000
K: 0.700
Wrinkle N: 1
Wrinkle H: 0.010

Type: Parallel
height: 0.500
Fallstart: 0.000
Fallfin: 1.000
K: 0.400
Wrinkle N: 2
Wrinkle H: 0.020

Type: Sphincter
Fallstart: 0.300
Fallfin: 0.900
R: 0.120
H: 0.040

Linear Parallel Sphincter

Figure 11 Muscle parameters UI

```
        //height for parallel muscle
        float height;
        //common parameters for linear/parallel muscles
        float Fs;
        float Ff;
        float K;
        float wrinkle_h;
        int wrinkle_n;
        //parameters for sphincter muscle
        float R;
        float H;
} Muscle;
```

Which is displayed and can be edited in UI using some code written in python (see figure 11). Since each muscle object define only one control point, there should be a way to link several muscle objects to form one muscle. We used parenting for this, root node is always the origin of muscle and its child or children (depending on type of muscle) define control points. Such way also is good for editing muscle set since you can move whole muscle by moving only origin.

4.1.2 Muscle modifier

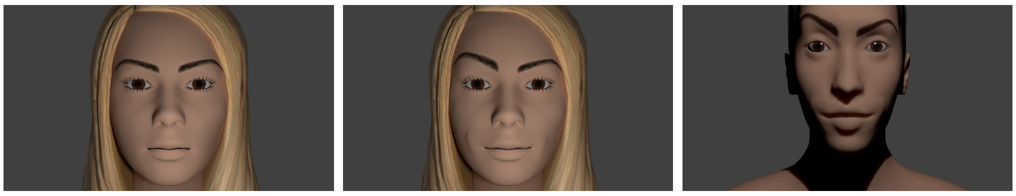
Muscle modifier in terms of Blender modifiers is non-desctructive deforming modifier which means that it only deforms vertices positions without adding or removing them. Modifier is parametrized by list of entries. Each entry contain:

- Reference to muscle origin object which give parameter of muscles as well as control points locations
- Optional vertex group used to limit or decrease effect of muscle
- Contraction value used to control contraction of the muscle

Which in code is defined as:

```
typedef struct MuscleModifierListData {
        struct MuscleModifierListData *next, *prev;
        char defgrp_name[64];      /* Name of vertex group to modify/weight. MAX_
        struct Object *muscle;     /* Associated muscle */
        float contraction;         /* Contraction of that muscle */
        char pad[4];               /* required padding */
} MuscleModifierListData;
```

Modifier works by for each vertex walking through all muscles and summing their displacement which are calculated as described in section 3.1. You can see effect of the modifier on two different meshes on figure 26.



Neutral | Effect of 3 muscles active: both *Cheek* with contraction 0.7 and *EyebrowForehead* fully contracted | Same contraction values on retargeted model

Figure 12 Example renders

Modifier can be animated by standard blender animation system thus having all of its wide possibilities, for example one can use simplify-curves plugin to automatically clean animation curves from tracker which sometimes are noisy.

4.1.3 Muscle set

We developed a muscle set which can represent different facial animations. It consist of 11 muscles (see figure 13), 5 types mirrored for both left and right sides of face, and one for the lips:

1. Cheek muscles - linear muscles, make smile expression
2. Lips1 muscles - linear muscles, make sad expression
3. Lips3 muscles - linear muscles, pull middle of the upper lip up
4. NoseForehead muscles - linear muscles, pull eyebrows close to nose
5. EyebrowForehead muscles - parallel muscles, pull eyebrows up
6. Lips muscle - sphincter muscle, closes mouth and push lips away from face

4.1.4 Tools

We also added several commands to Blender to work with muscle objects and transfer data between Blender and tracking application. All commands are bundled as one python addon called *Muscle tools* and are accessible through space menu.

Load muscle keyframes from file

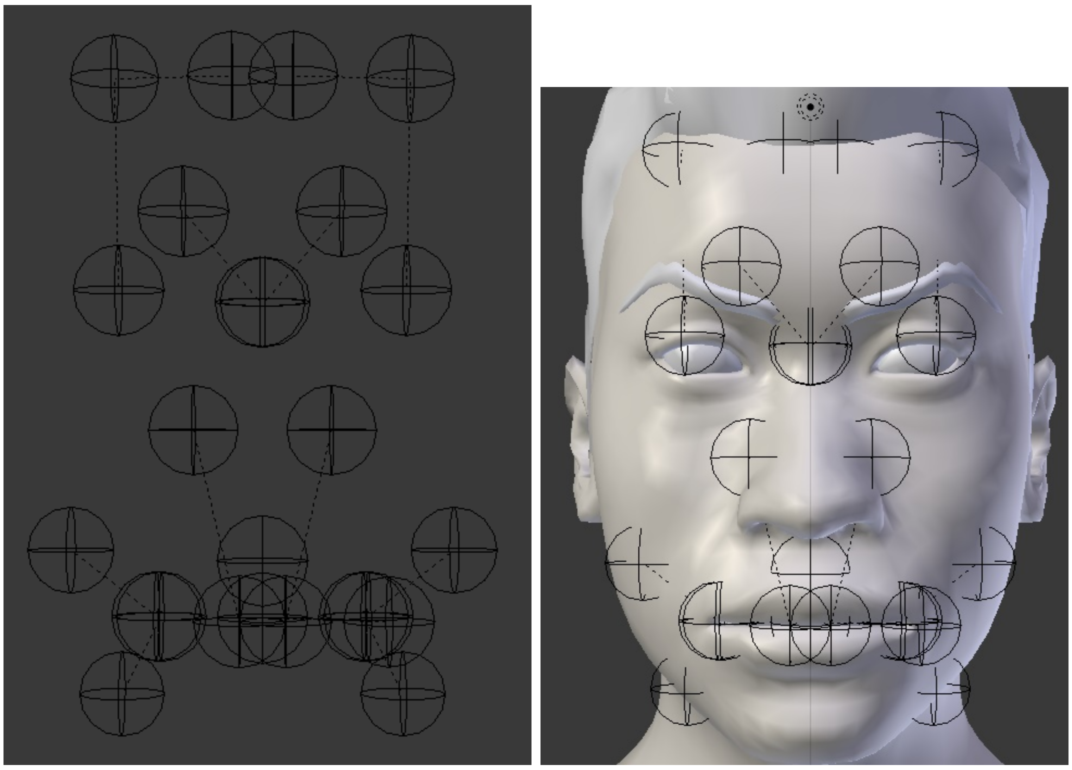
This operator loads keyframe data for animation of muscles and jaw from csv file produced by tracking application. Operator animates currently selected object using keyframes using loaded keyframes starting from currently selected frame.

Save muscles positions

This operator saves muscles positions in format suitable for retargeting. To do so, for each muscle object it finds closest mesh vertex and saves its id along with disposition.

Load muscles from file

This operator loads file produced by *Save muscles positions* operator and uses it to add muscle set based on currently selected mesh.



Muscles only   Muscles with mesh

Figure 13 Muscle set

Save muscles absolute positions

This operator saves muscle data to file similar to *Save muscles positions* operator, but it stores only absolute coordinates of muscle objects. This file is then used by tracking application.

Save keypoints

This operator saves to file positions of vertices that are in *keypoints* vertex group. This simplifies creation of keypoints file for tracking tool because keypoints can be chosen visually and saved all at once.

4.2 Tracker

Tracking tool is implemented in separate python file which can be used outside of Blender and has no dependency on it. It is written for python version 2.7 and uses numpy, opencv and scipy.optimize packages.

Input to tracker is following data:

- Muscle set data (may be produced by *Save muscles absolute positions* tool in Blender)
- Mesh keypoints 3D locations (may be produced by *Save keypoints* tool in Blender)
- Calibration matrix
- Jaw bone origin location
- Video file
- Positions of all keypoints in first frame

- Scale and window size options for optical flow

Tracker follows workflow decribed in section 3.4. It uses first frame and provided positions of static keypoints to evaluate initial projection matrix P by using opencv function *solvePNP* with CV_EPNP method. Initial x as well as $xrot$ for jaw are assumed to be zero (meaning that all muscles of face are relaxed and jaw is closed).

For each consecutive frame it then goes through several steps as depicted in Algorithm 2. As one can note, muscle error is minimized not globally, but point-by-point. This is done intentionally, because during experiments we found out that global minimization produces similar results but requires much more iterations to converge.

For optical flow we use Lucas-Kanade tracking algorithm implementation from opencv using function *calcOpticalFlowPyrLK*. Model parameters for each frame are then written to csv file for further processing. This file can be used as input for *Load muscle keyframes from file* Blender command to import animation to Blender scene.

```
input : previous contraction $x_1$, previous projection matrix $P_1$, previous jaw
        angle $xrot_1$, mesh keypoints $D$, calibration matrix $K$, previous frame $f_1$,
        current frame $f_2$, previous frame image locations $I_1$
output: new contraction $x_2$, new projection matrix $P_2$, new jaw angle $xrot_2$, new
        image locations $I_2$
$I_2 \leftarrow$ OpticalFlow ($f_1$, $f_2$, $I_1$);
$P_2 \leftarrow$ solvePNP ($D$, $K$, $I_2$);
$xrot_2 \leftarrow$ minimize $E_{jaw}$;
$x_2 \leftarrow x_1$;
foreach point p in D do
    $x_2 \leftarrow$ minimize $E_p$ starting from $x_2$
end
```

Algorithm 2: Tracking algorithm

5 Experiments

5.1 Evaluation

During following experiments we will evaluate quality of algorithm by comparing its output with ground truth values provided by user. Since there are many frames in each video and providing user input for each would require too much labor work, we limit evaluation to several keyframes. We then define error function as

$$E = \frac{\sum_{f \in keyframes} ||x_{f,eval} - x_{f,groundtruth}||}{|keyframes| * |x|} \tag{14}$$

where $keyframes$ is set of keyframes, $x_{f,eval}$ is muscle contraction vector concatenated with jaw rotation divided by $\pi/4$ (for normalization) produced by algorithm for frame f and $x_{f,groundtruth}$ is the same vector for the same frame ground truth values provided by user, $|x|$ is length of vector x.

5.2 Experiments setup

All experimental videos were recorded by Canon PowerShot SX40 HS camera with smallest zoom level without flashlight. Camera was calibrated using opencv calibration function *calibrateCamera*. All tests were conducted on PC with Intel i7-3630QM CPU, 8GB RAM, Windows 8.1, python v.2.7.9 with numpy v.1.9.2, scipy v.0.15.1, opencv-python v.2.4.11 acquired from [29].

For experiments we recorded five similar performances by two people (we further refer to them as setups, see figure 15):

1. Person was asked to produce expression which contract only one pair of muscles, one by one, returning to neutral position after each, with camera looking straight on face and not moving. Following order of expressions was asked (figure 14):
   - Smile - activate both Cheek muscles in our model
   - Sad - activate both Lips1 muscles in our model
   - Open jaw
   - Frown - activate both NoseForehead muscles in our model
   - Raise eyebrows - activate both EyebrowForehead muscles in our model
   - Push lips forward - activate Lips muscle in our model
2. The person was asked to do same expressions again, but this time time camera was looking from some angle.
3. In third setup same expressions were asked but camera looked completely from side.
4. Fourth setup again used same expressions, but this time camera was moved and rotated around the person.
5. Person was asked to produce random facial expressions involving several muscles.

Tests are named as “m%is%j” where “%i” is person number (1, 2) and “%j” is number of setup (from 1 to 5).

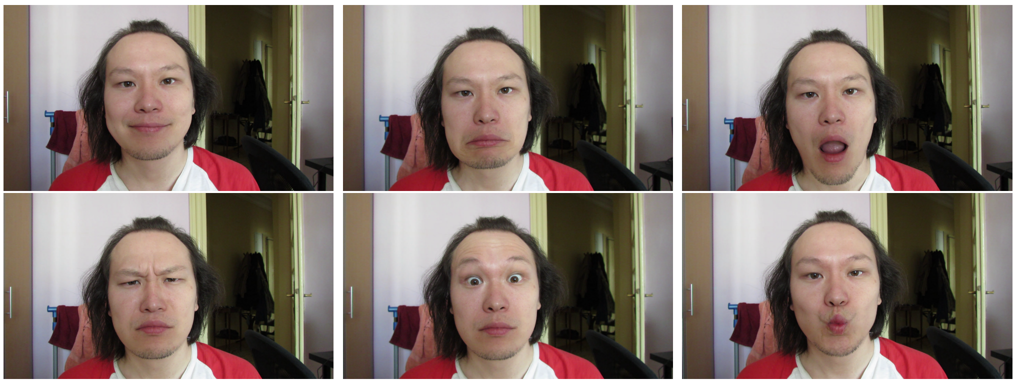


Figure 14 Experimental poses

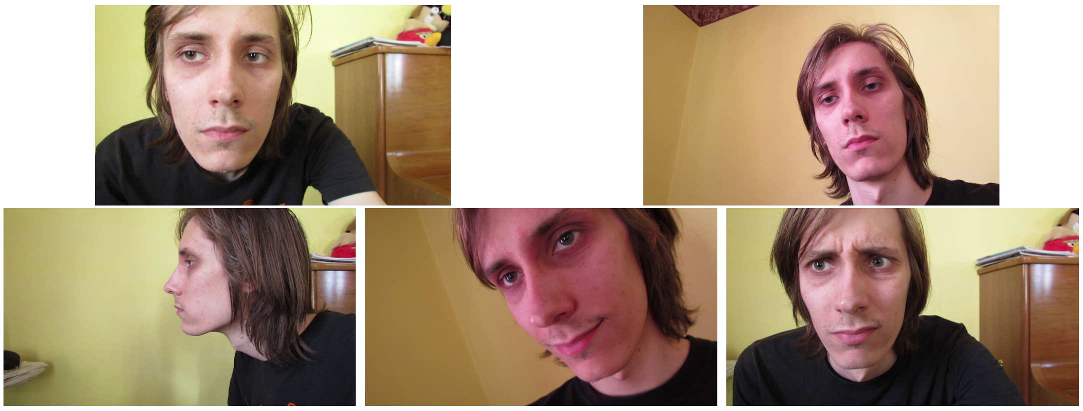


Figure 15 Experimental setups

For each video file we then marked keypoints locations on first frame and some frames with groundtruth contraction and jaw opening values. For third setup where we see only half of the face, we assumed absolute symmetry on the face and thus setup right face part keypoints locations to the same spots as left.

As keypoints we used following facial landmarks (see figure 16):

1. Eyebrows corners
2. Eyes corners
3. Nose tip
4. Left and right most points of the nostrils
5. Lips cupid bow
6. Lips left and right corners
7. Forehead middle point

5.3 Opticalflow parameters

We started experiments by looking for good parameters of optical flow that would result in fastest calculation but robust. To do so we added scaling factor which was used to scale input frame before performing optical flow, parameter to control window size used by LK tracker (window was also square). We used $1.0, 0.75, 0.5, 0.25$ as possible values for scale and $60, 50, 40, 30, 20, 15, 10, 5$ as possible window sizes.

We then run our tracking program on all test videos using all combinations of scale and window size. Average time for each frame (figure 17) as well as average error (figure

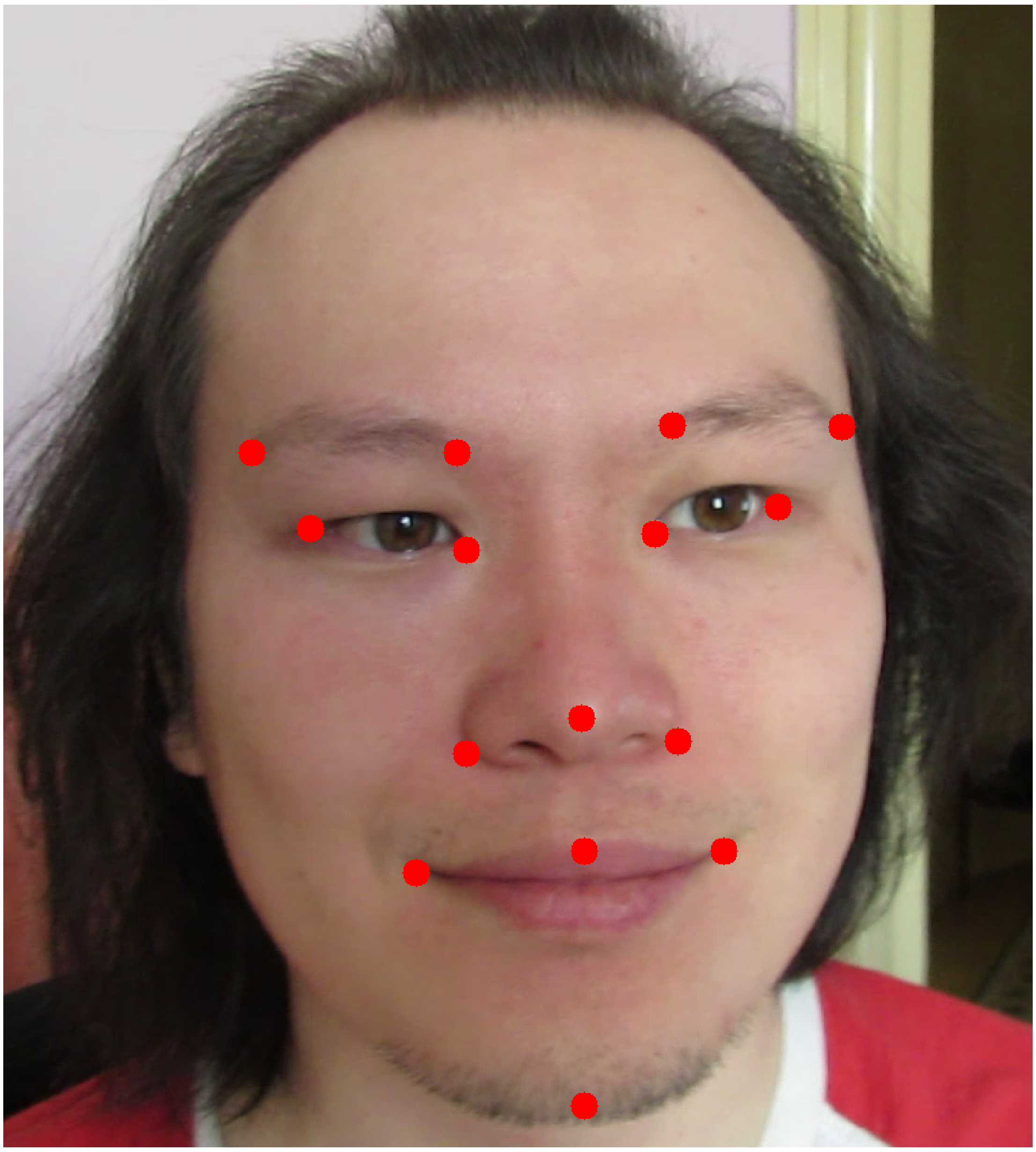


Figure 16 Landmarks

18) for all tests were calculated. As one can see window of size 5 is significantly faster then others but also produce larger error. From other options, $scale = 0.5$ is fastest and has lowest error with $windowsize = 20$, larger window sizes produce very close error on all scales without significantly slowing down. One can also notice that selected options have average processing time per frame around 42 msec which is almost enough to process 24 frames per second. This shows that our approach can be used in real-time applications. For further experiments we will use $scale = 0.5$, $windowsize = 20$.

5.4 Model analysis

We captured input for different persons which have dissimilar faces (one Caucasian, other Asian) and we have an option to choose different facial meshes to use for algorithm. We have prepared three different facial meshes (figure 19), two of them were made

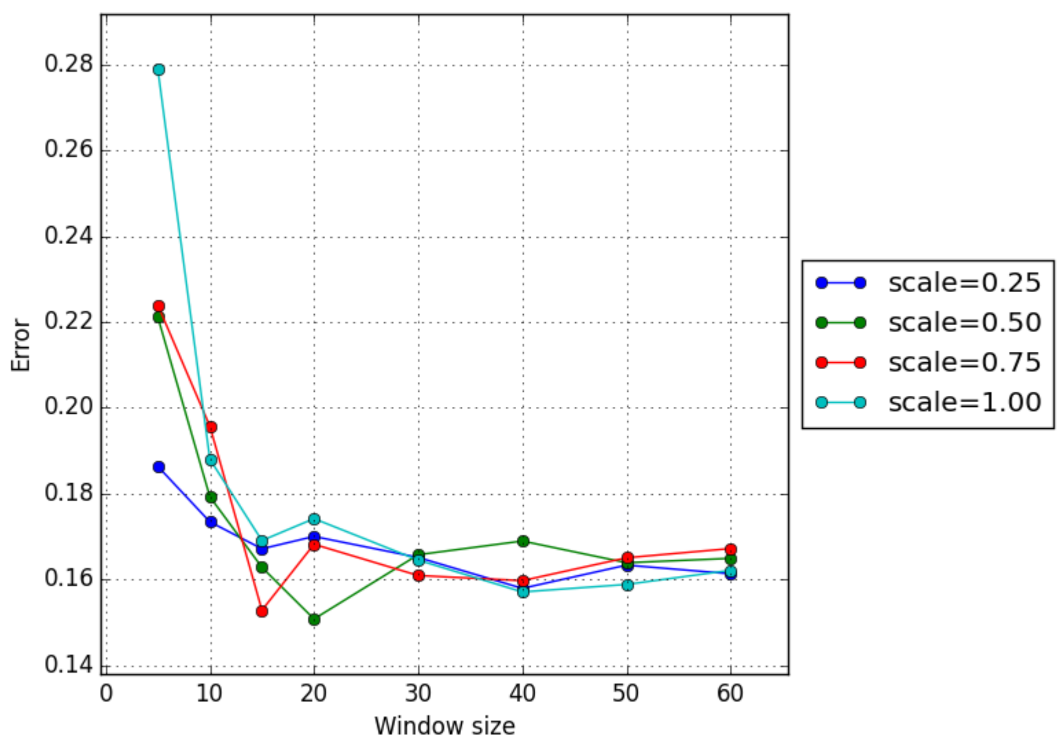


Figure 17 Effect of different optical flow parameters on error

similar to the real faces, and the third one was made dissimilar to both. We then run tracker with different facial mesh as basis for each person and calculated error for each setup (figures 20a, 20b). One can see that mesh does not affect error significantly, that shows robustness of method to dissimilarity in mesh and real face.

For further experiments we used error produced by using similar face mesh for each person.

5.5 Setups analysis

On figure 21 you can see error for each setup averaged for both persons.

First setup, which may be seemed as simplest, shows that model tracker actually works as intended producing low (smaller than 0.12) error on average.

Second setup has the same performance as first, showing that camera position can vary on some angles without losing any quality.

Third setup produced larger error, however it is still reasonable. That proves that method works even for extreme angles if facial landmarks image positions are correct.

Fourth setup for both persons shows worse performance compared to others. This is primarily caused by inaccurate optical flow results (see figure 24) which in turn has large impact on the model parameters. On figure 22 we show what maximum distance in pixels that muscle can move keypoints (in frontal view), we can see that it is about 35 pixels for most of the muscles, which is quite small if whole image is 1920x1080 pixels large and thus small error in optical flow results can have big impact on the model parameters. On figure 23 you can see that quality of results degrades as more movement happens.

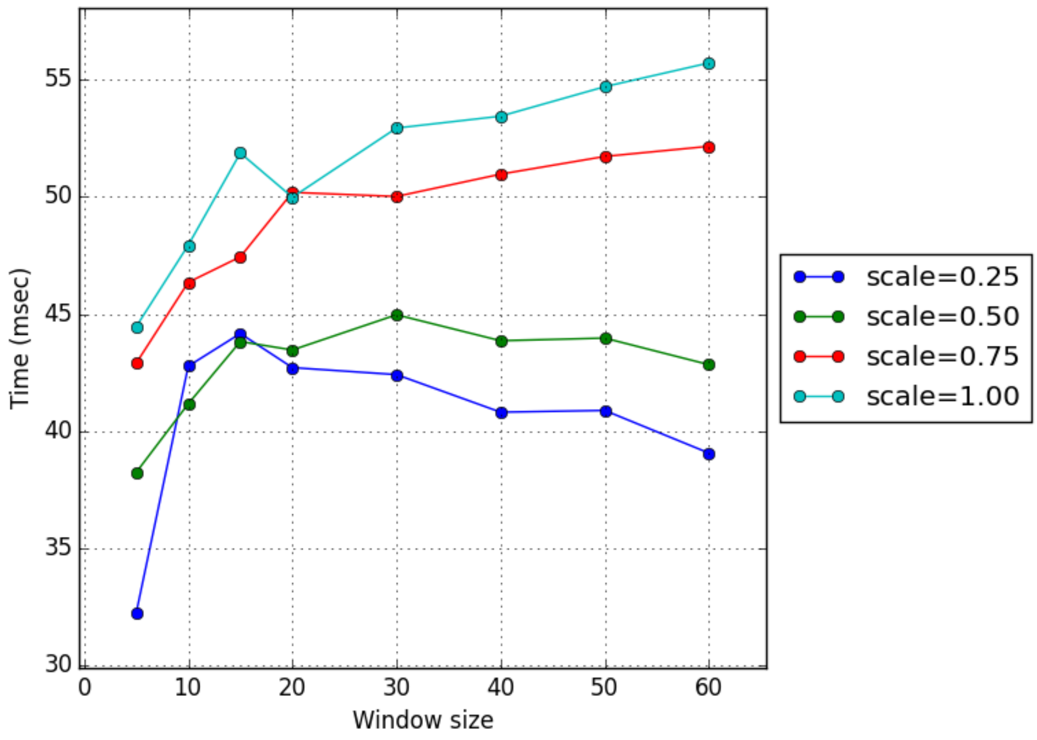


Figure 18 Effect of different optical flow parameters on processing time

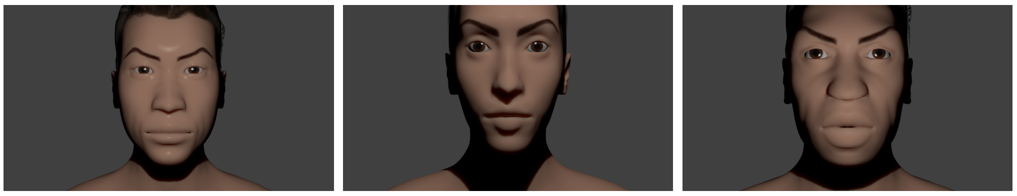


Figure 19 Meshes used in experiments

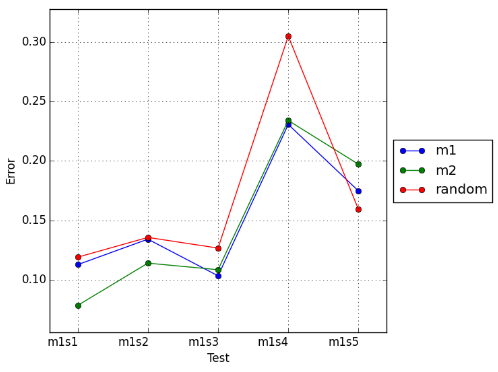
Fifth setup also shows reasonable error (smaller than 0.2) even when several muscles are contracted which proves that tracker is working not only for simple cases.

5.6 Per muscle error

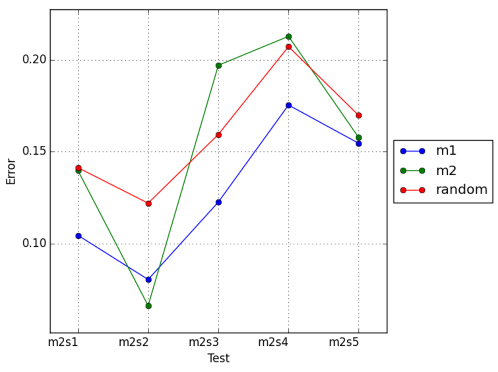
On figure 25 we present error per each muscle for each test. One can see that lips muscles have worse results then other. That can be caused by their flexibility and worse optical flow results then other facial landmarks.

5.7 Rendering virtual avatar

We then used resulting animation data and loaded it to blender and rendered some chosen frames, you can see input frame and corresponding rendered frames on figures 26 and 27.



a) Person one



b) Person two

Figure 20 Error for different meshes and tests

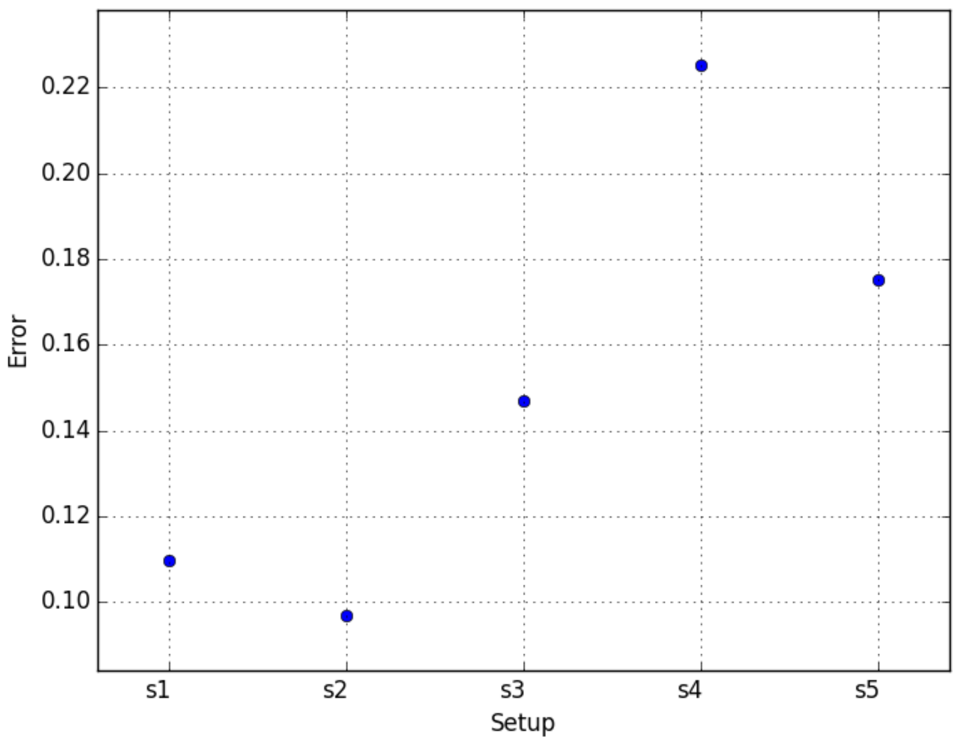


Figure 21 Error for each setup

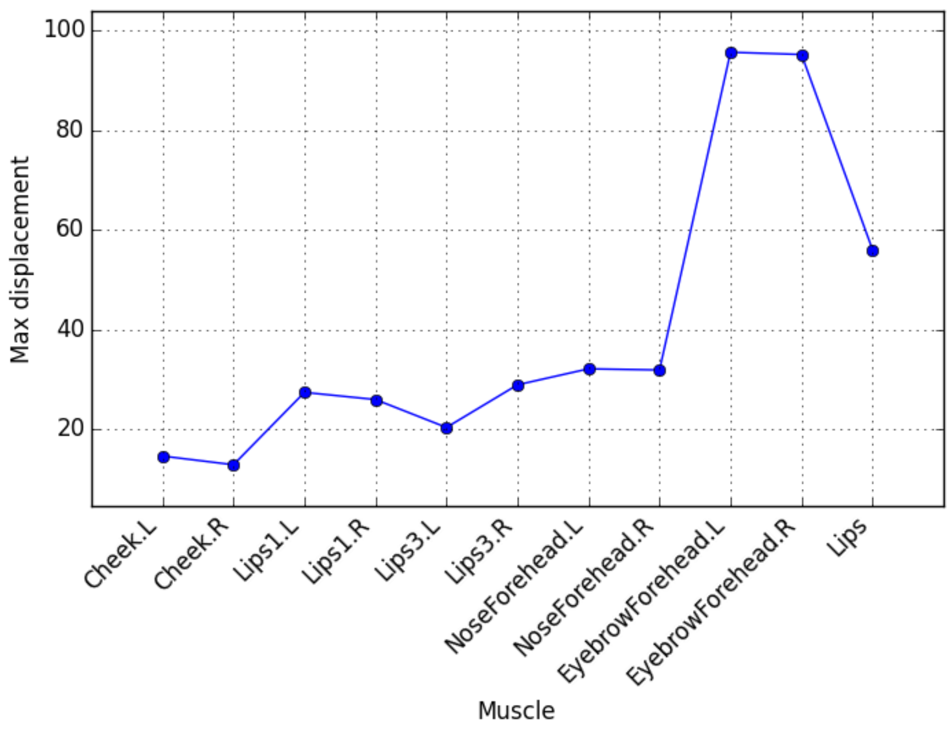


Figure 22 Maximum displacement of each muscle for frontal view

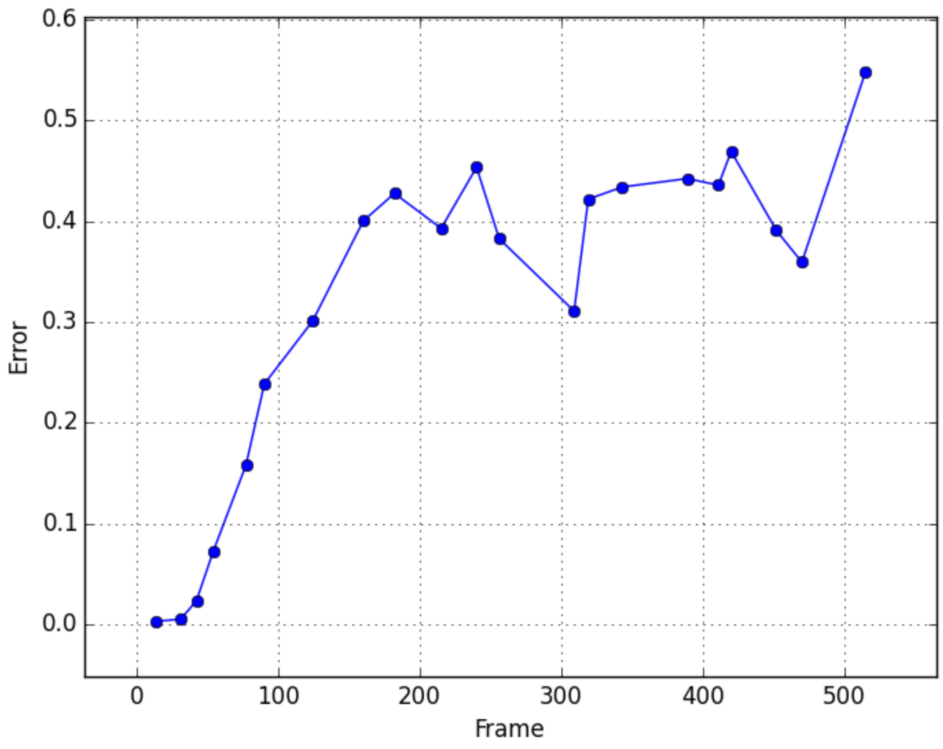


Figure 23 Error for selected keyframes in test m1s4

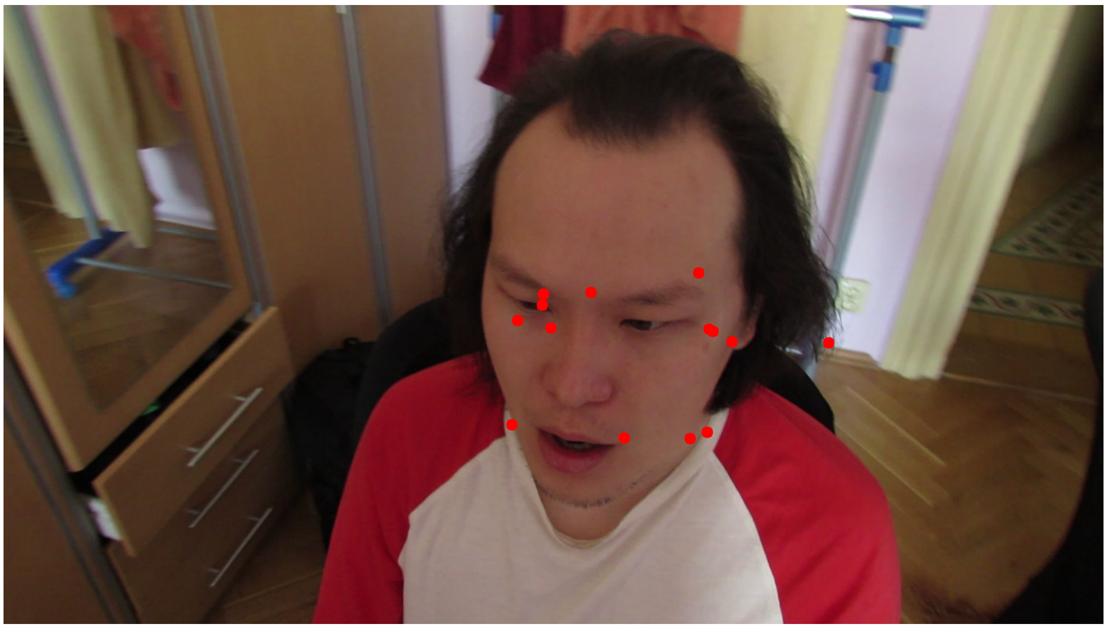


Figure 24 Incorrect optical flow results

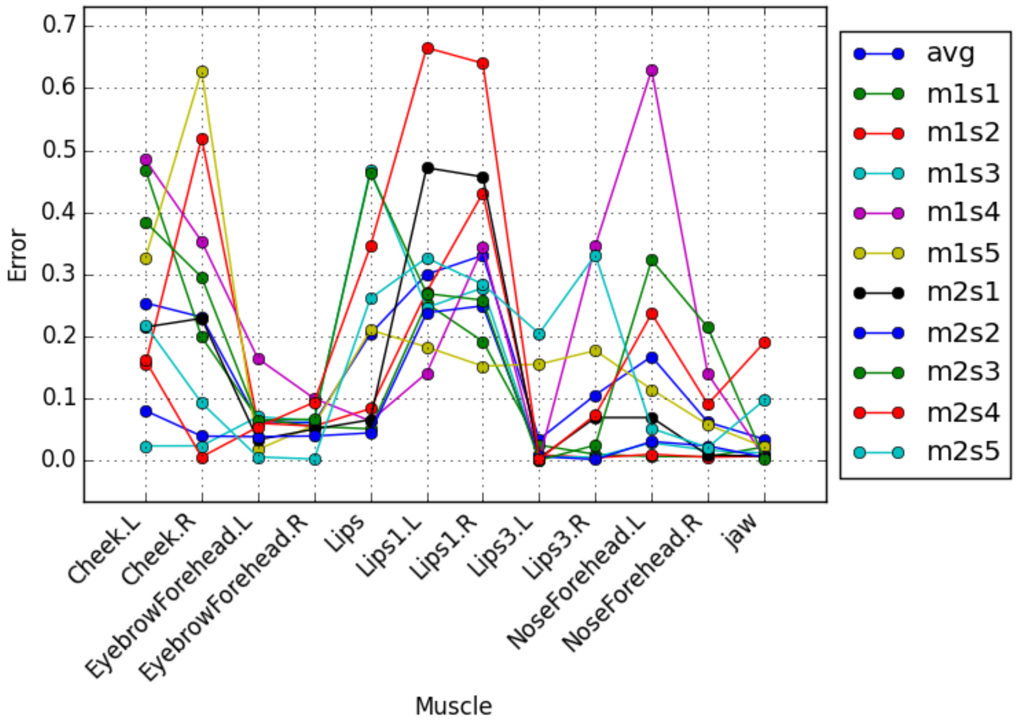


Figure 25 Per muscle error averaged for all setups and persons

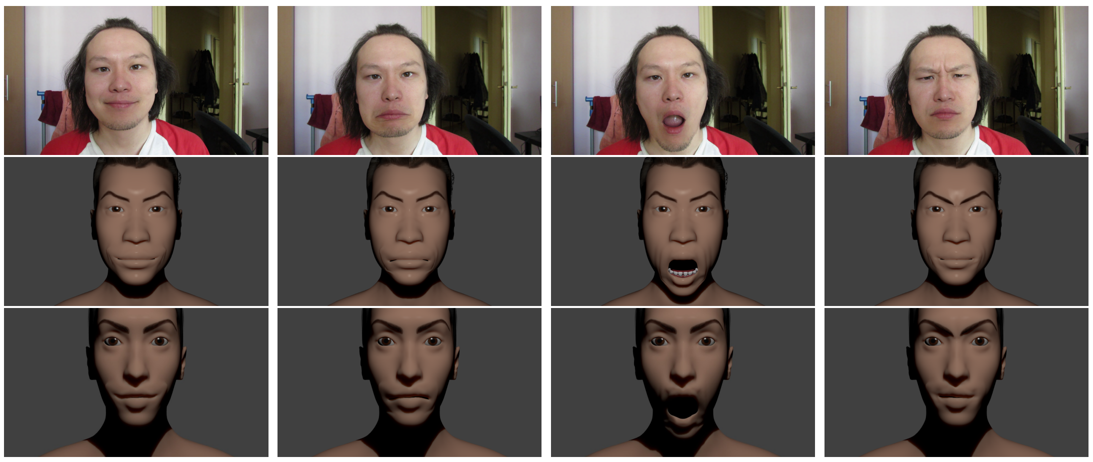


Figure 26 Renders for first person performance

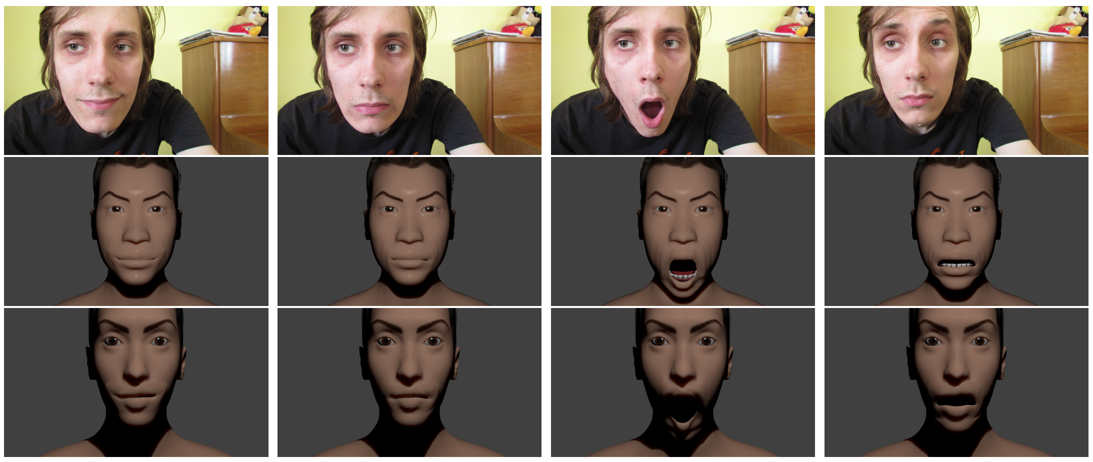


Figure 27 Renders for second person performance

6 Conclusion

6.1 Results

This thesis proposes a muscle model along with technique for facial capture based on that model. Proposed model was tested on five performances produced by two persons. During testing it has been shown that such model can be successfully used for real-time motion capture with adequate error and further animation of virtual character using a single consumer-level camera. Animation data can be easily retargeted to another virtual characters created in Makehuman.

6.2 Future Work

Although the proposed tracker is functional there are a lot of possible improvements:

- Method suffer from inaccurate opticalflow results, in further work ot can be replaced by a better alternative, for example Cascaded Pose Regression method [30].
- Current implementation uses only one core for computations, with modern processor having up to 8 cores rewriting code to multithreaded version can result a significant speed improvement.
- User is required to enter landmarks positions on first frame, that makes almost impossible to use method online. This issue can be solved by using some method to automatically detect landmarks, it would also help to recover from situation where tracking fails or provide inadequate results.
- Better integration with Blender may reduce number of steps required to do whole process from motion capture to rendering.

The thesis provides necessary framework for continuing elaborations.

Appendix A

Documentation

This appendix describes how to use tools created during this thesis.

A.1 Blender

As was described, muscle object and modifier for Blender were created by patching its source code. You may try to use compiled version (for Windows) available on CD in archive “blender/Release.zip”, you should be able to run it after unpacking, without any additional steps. If you wish to build it yourself, there is a full source code available on CD in archive “blender/blender_src.zip”. For compile instruction please visit blender website [31].

Most of the operations can be done only through space menu. Space menu is a blender menu which appears if you press *space* on keyboard when you are not inputting text you can see example on figure 28, there are a lot of commands available, their list can change depending on context (for example selected object, mode). You can type to filter commands by name.

Muscle objects can be added through space menu command *Add muscle* which adds two *Linear* muscles on some distance from each other. You can then use any blender tools to move them around scene. You can find muscle parameters under tab with camera sign (see figure 29).

Muscle modifier can be added to mesh as any other modifier on tab *Modifiers* and configured there (see figure 30). It has a list of muscle, vertex group, contraction items in it and initially is empty. You can add and delete items from that list by plus and

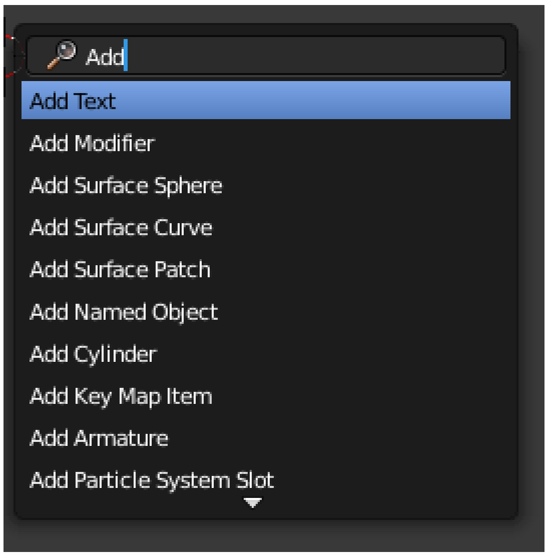
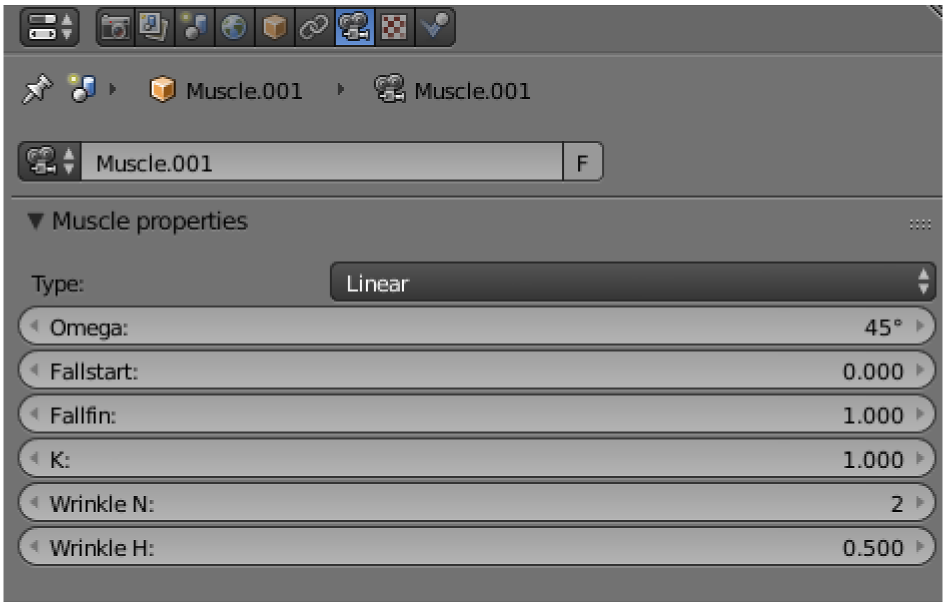


Figure 28 Blender space menu



Muscle.001 › Muscle.001

Muscle.001 F

▼ Muscle properties

Type:	Linear
Omega:	45°
Fallstart:	0.000
Fallfin:	1.000
K:	1.000
Wrinkle N:	2
Wrinkle H:	0.500

Figure 29 Muscle properties

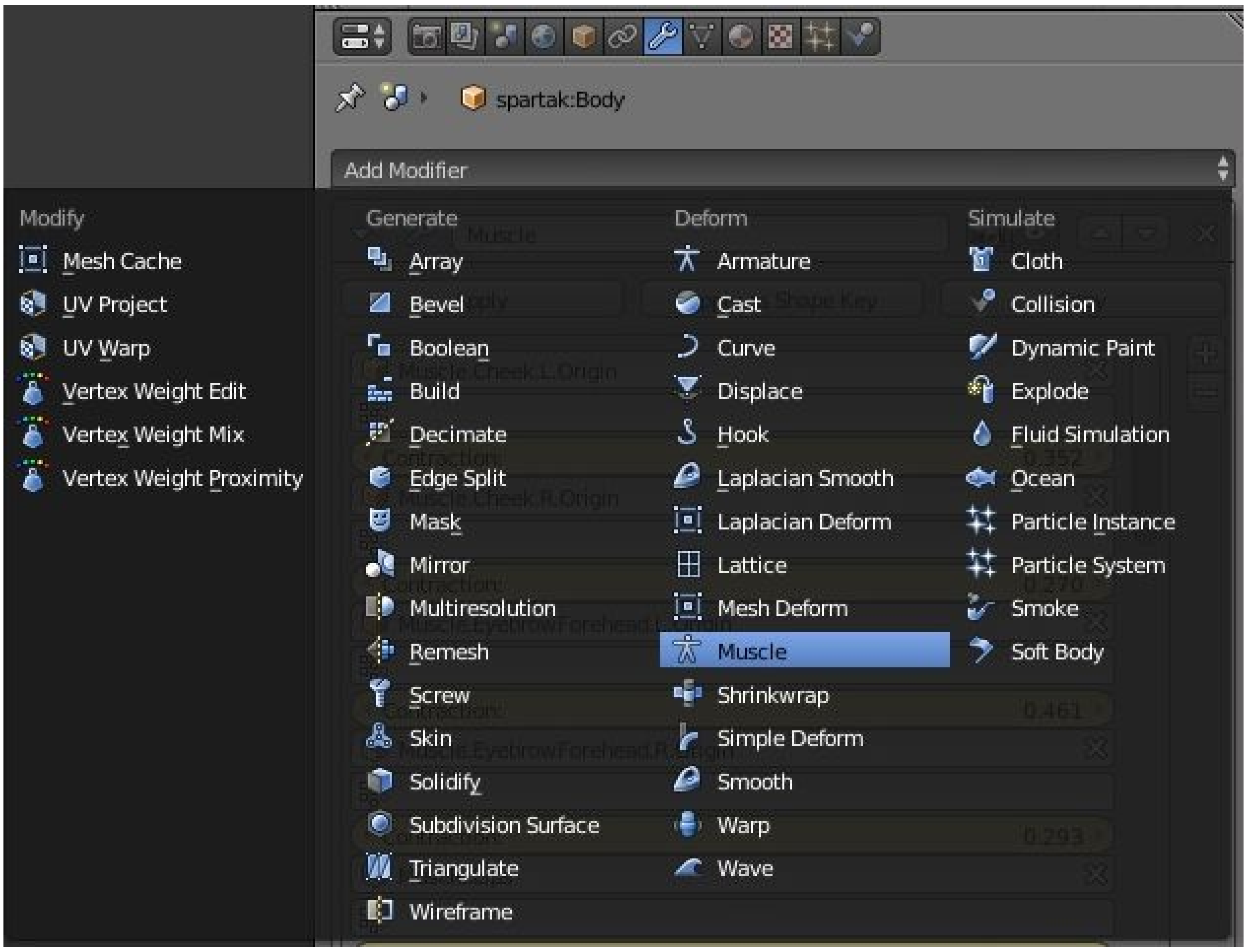


Figure 30 Adding modifier

minus buttons on the right side of the list. For each item you should choose origin of the muscle, that should always be parent. If you choose wrong muscle (for example it does not have enough or have too much children) modifier will just skip it.

A.1.1 Tools

To use tools you will probably need to enable plugin, you can do that by going into *File* -> *User preferences* in main menu. Then on tab *Addons* you should select *Testing* support level, and *Muscles* category. You should then see single addon on the right, to enable it tick the checkbox. All following operations are available through space menu, their actions were described in section 4.1.4:

- Load muscle keyframes from file
- Save muscles positions
- Load muscles from file
- Save muscles absolute positions
- Save keypoints

A.2 Tracker

To run tracker you will need python interpreter version 2.7 with numpy, scipy, opencv packages installed. Please follow instructions on websites of these project to do that [32] [33] [34].

Tracker is represented as a single python file *muscles.py* in directory *tracker* and provides several classes. One may use *Tracker* class to track proposed model parameters directly, or can use class *MTest* which handles loading of required data from test files. You can also run the file directly which will run set of predefined tests. If you would like to change some parameters, or add/delete tests to the process list, you can edit code in the bottom of the file. Commented parts in the end of file can be used to produce data which was used to generate experimental results.

There is also a script named *runtest.py* which you can use to run single test by passing test name (without extension), scale, and window size as arguments.

A.3 Example

This section will guide you through all steps from Makehuman model to rendering animation in Blender. You may skip some steps and use provided data instead of making your own.

A.3.1 Exporting model from Makehuman

There are three mhx files provided on CD in directory testdata, you can use them if you wish to skip that step.

After you have create model in Makehuman, you should export it in *MHX* format, to do that you should go to *Files* -> *Exports* tab, select *Blender Exchange (mhx)* as mesh format with all other settings in their default values. After specifying filename and clicking export it should be done.

A.3.2 Importing model, creating muscles

There are three blend files with models imported, muscles and muscle modifier configured on CD in directory testdata, you may use them if wish to skip that step.

Next step is to import Makehuman model to Blender. This can be done using *Import MHX* space command or through menu *File -> Import -> Makehuman(.mhx).* If you do not have such options you need to install Makehuman plugin to Blender, please follow instructions on Makehuman website [35] to do that. After that step you should see human mesh in Blender 3d window.

You should now add some muscles to the face. If your model have eyebrows, you can use provided muscles file and retarget them to your mesh. You should first join eyebrows mesh with body. To do so first select eyebrows, then while holding shift select body mesh, next use operator *Join* (Ctrl+J or through space menu). Now load muscle data from file using operator *Load muscles from file*, provide file is in directory *testdata* and is called *muscles.txt.* Muscle objects across face should appear.

If your model does not have eyebrows, or you want to use your own muscle set, you can build it by adding some muscle objects. You can save it and then retarget using operator *Save muscle to file.*

Next we should add muscle modifier. You can test it by adding some muscles to its list and changing contraction parameter. If you use proposed muscle set, you would probably want to limit effect of Lips3 muscles using vertex group because they deform nose otherwise. Please look in provided blend files for example of vertex group.

A.3.3 Exporting data from Blender for tests

There are three pairs of files on CD in directory testdata called muscles_m1, keypoints_m1, muscles_m2, keypoints_m2, muscles_m3, keypoints_m3 which correspond to models m1, m2, m3 respectively. You can use these files if you want to skip this step or you use one of the provided models. Next step is to export muscle data and keypoints information from Blender for test application. To do that, choose body mesh and run operator *Save muscles absolute positions* to save muscle data. To save keypoints you should create vertex group called *keypoints* and add keypoint vertices on mesh to it. Then you can save their locations using operator *Save keypoints.* However you will need to modify that file by specifying if the keypoint is *static*, *dynamic* or is it used for jaw opening estimation (there should be only one *jaw* point). It is hard if you save all points at once so we suggest to add them one by one, or use provided data.

A.3.4 Creating test file

There are several tests in directory testdata, you can use them if you wish to skip that part

All filenames in following are relative to *tests* directory and for simplicity should be placed there.

After you acquired video from some camera, you may prepare test file. Its first and second lines should be a file name with muscles data and file name of keypoints produced on previous step. On third line there should be name of video file. Forth line should be a name of file with calibration matrix. This file may be numpy saved data (it should end in .npy) or text file with matrix rows on lines and columns separated by single space. Next line should be coordinates of *jaw* bone origin, you can see that in Blender by selecting *jaw* bone, going to edit mode and in tab *Bone* you can see coordinates.

Next line contains n number of keypoints, it should equal number of keypoints specified in keypoints file. Next n lines should contain image coordinates of corresponding keypoints on the first frame. You can use script *choose_points.py* to simplify that step.

After keypoints image locations there should be two numbers on the line separated by space: number of tested muscles m and ground truth frames l. Next line should contain exactly integer l numbers separated by space - ground truth keyframes numbers. Next $m * 2$ lines should contain name of muscle on one line followed by l float ground truth values for that muscle and corresponding frames. Please refer to provided test for examples.

A.3.5 Running test

To run test you may either add it to the list of tests in *muscles.py* file, or use *runtest.py* passing test name, scale, and window size as arguments. Window should appear showing currently tracked positions of landmarks and current estimation of model parameters. You may end test processing by pressing *Escape* on keyboard. After test has completed, two files should appear in tests directory: *%testname%_%scale%_%windowsize%_error.csv* a file with error information for each muscle in csv format and average time required to process frames, *%testname%_%scale%_%windowsize%_result.txt* also a csv file which contains tracked model parameters, can be imported to Blender.

A.3.6 Importing animation to Blender

You can import model parameter data produced by tracker to Blender by selecting body mesh and running operator *Load muscle keyframes from file*. It should add keyframes starting from currently chosen. You should be able now to render animation or selected frames with corresponding model parameters.

Bibliography

[1] Derek Bradley et al. “High Resolution Passive Facial Performance Capture”. In: *ACM Transactions on Graphics* (2010).

[2] Thibaut Weise et al. “Realtime performance-based facial animation”. In: *ACM Transactions on Graphics (Proceedings SIGGRAPH)* (2011).

[3] Sofien Bouaziz, Yangang Wang, and Mark Pauly. “Online Modeling For Realtime Facial Animation”. In: *ACM Transactions on Graphics (Proceedings SIGGRAPH)* (2013).

[4] Ferderic I. Parke. “Parameterized Model for Facial Animation”. In: *IEEE Computer Graphics & Applications* (1974), pp. 61–68.

[5] “Animating Facial Expressions”. In: *Proceedings ACM SIGGRAPH Computer Graphics*. 1981, pp. 245–252.

[6] Keith Waters. “A Muscle Model for Animating Three-Dimensional Facial Expressions”. In: *SIGGRAPH ’87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. Ed. by Maureen C. Stone. 1987.

[7] Jane Wilhelms. “Modeling Animals with Bones, Muscles, and Skin”. In: (1994).

[8] Jane Wilhelms and Allen Van Gelder. “Anatomically Based Modeling”. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 173–180. ISBN: 0-89791-896-7. DOI: 10.1145/258734.258833. URL: http://dx.doi.org/10.1145/258734.258833.

[9] Ferdi Scheepers et al. “Anatomy-based Modeling of the Human Musculature”. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 163–172. ISBN: 0-89791-896-7. DOI: 10.1145/258734.258827. URL: http://dx.doi.org/10.1145/258734.258827.

[10] L.P. Nedel and D. Thalmann. “Real time muscle deformations using mass-spring systems”. In: *Computer Graphics International, 1998. Proceedings*. June 1998, pp. 156–165. DOI: 10.1109/CGI.1998.694263.

[11] Lance Williams. “Performance-driven Facial Animation”. In: *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’90. Dallas, TX, USA: ACM, 1990, pp. 235–242. ISBN: 0-89791-344-2. DOI: 10.1145/97879.97906. URL: http://doi.acm.org/10.1145/97879.97906.

[12] Irfan Essa et al. “Modeling, Tracking and Interactive Animation of Faces and Heads Using Input from Video”. In: *Proceedings of the Computer Animation*. CA ’96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 68–. ISBN: 0-8186-7588-8. URL: http://dl.acm.org/citation.cfm?id=791215.791488.

[13] Volker Blanz and Thomas Vetter. "A Morphable Model for the Synthesis of 3D Faces". In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 187–194. ISBN: 0-201-48560-5. DOI: `10.1145/311535.311556`. URL: `http://dx.doi.org/10.1145/311535.311556`.

[14] Jin-xiang Chai, Jing Xiao, and Jessica Hodgins. "Vision-based Control of 3D Facial Animation". In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '03. San Diego, California: Eurographics Association, 2003, pp. 193–206. ISBN: 1-58113-659-5. URL: `http://dl.acm.org/citation.cfm?id=846276.846304`.

[15] Chen Cao et al. "FaceWarehouse: a 3D Facial Expression Database for Visual Computing". In: *IEEE TVCG* (2013).

[16] Yanlin Weng et al. "Real-time Facial Animation on Mobile Devices". In: *Graphical Models* (2013).

[17] Cao Chen, Qiming Hou, and Kun Zhou. "Displaced Dynamic Expression Regression for Real-time Facial Tracking and Animation". In: *ACM Transactions on Graphics* (2014).

[18] Heather Brannon. *Epidermis anatomy*. URL: `http://dermatology.about.com/od/anatomy/ss/epidermis.htm` (visited on 10/05/2015).

[19] I. A. BROWN. "A scanning electron microscope study of the effects of uniaxial tension on human skin". In: *British Journal of Dermatology* 89 (1973), pp. 383–393. DOI: `10.1111/j.1365-2133.1973.tb02993.x`.

[20] Allan Forsman. *Histology of muscle*. URL: `http://faculty.etsu.edu/forsman/histologyofmuscleforweb.htm` (visited on 10/05/2015).

[21] Kolja Kähler. "A Head Model with Anatomical Structure for Facial Modeling and Animation". Doctoral dissertation. Saarbrücken: Universität des Saarlandes, Dec. 2003, p. 150.

[22] A. Burton and J. Radford. *Thinking in Perspective: Critical Essays in the Study of Thought Processes*. Psychology in progress. Methuen, 1978. ISBN: 9780416858402. URL: `http://books.google.cz/books?id=CSgOAAAAQAAJ`.

[23] S. S. Beauchemin and J. L. Barron. "The Computation of Optical Flow". In: *ACM Comput. Surv.* 27.3 (Sept. 1995), pp. 433–466. ISSN: 0360-0300. DOI: `10.1145/212094.212141`. URL: `http://doi.acm.org/10.1145/212094.212141`.

[24] *Optical flow*. URL: `http://en.wikipedia.org/wiki/Optical_flow` (visited on 10/05/2015).

[25] *Lukas-Kanade method*. URL: `http://en.wikipedia.org/wiki/Lucas%C3%A2%C2%80%C2%93Kanade_method` (visited on 10/05/2015).

[26] Duy Bui. "Creating emotions and facial expressions for embodied agents". Dissertation. University of Twente, 2004.

[27] *Makehuman website*. URL: `http://makehuman.org/` (visited on 10/05/2015).

[28] Stephen G. Nash. "Newton-Type Minimization via the Lanczos Method". In: *SIAM Journal on Numerical Analysis* 21.4 (1984), pp. 770–788. DOI: `10.1137/0721052`. eprint: `http://dx.doi.org/10.1137/0721052`. URL: `http://dx.doi.org/10.1137/0721052`.

[29] *Unofficial python binary packages.* URL: `http://www.lfd.uci.edu/~gohlke/pythonlibs/` (visited on 03/05/2015).

[30] P. Dollár, P. Welinder, and P. Perona. “Cascaded Pose Regression”. In: *CVPR*. 2010.

[31] *Blender build instructions.* URL: `http://wiki.blender.org/index.php/Dev:Doc/Building%5C_Blender` (visited on 10/05/2015).

[32] *Python website.* URL: `https://www.python.org/` (visited on 10/05/2015).

[33] *Installing the SciPy pack.* URL: `http://www.scipy.org/install.html` (visited on 10/05/2015).

[34] *OpenCV website.* URL: `http://opencv.org/` (visited on 10/05/2015).

[35] *Makehuman blender tools installation.* URL: `http://www.makehuman.org/doc/node/mhblendertools_download_and_installation.html` (visited on 10/05/2015).