MUSCLE-BASED FACIAL ANIMATION

Terence Benja Küderle

University Of Applied Science Wedel (FH)

Diploma Thesis

Course of Study: Media Computer Science

Subject:

Muscle-Based Facial Animation

Author:	Terence Benja Küderle
	D-
	Tel.:
	Email:
Date of Submission:	Thursday, 24 February 2005
Referee (FH Wedel):	Prof. Dr. Christian-Arved Bohn
	University of Applied Science Wedel
	D-
	Tel.:
	Email:
Referee:	Volker Helzle
	Filmakademie Baden-Württemberg
	D-
	Tel.:
	Email:

DECLARATION BY CANDIDATE

I hereby declare that this thesis is my own work and that it contains no material which has been accepted for the award to the candidate of any other degree, except where due reference is made in the text of the thesis. To the best of my knowledge and belief, it contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Signature:……

Date:……...

Table of Content

List of Figures

List of Tables

List of Equations

List of Source Code Examples

List of Abbreviations

[AAYY]	Literature Reference [Initials of Author(s), Year of Publication]
[AAYY**I**]	Internet Reference [Initials of Author(s), Year of Publication, **I**]

abbr.	abbreviation
cf.	confer
De.	Deutsch, German
e.g.	for example; for instance
etc.	et cetera; and so on
FK	See Chapter A.5
ID	identification
IK	See Chapter A.5
n.d., ND	not dated
n.p.	no publisher
NURBS	See Chapter A.5
ODE	See Chapter A.5
UML	See Chapter A.5
ut sup.	ut supra; as above

1 Introduction

"A man's face as a rule says more, and more interesting things, than his mouth, for it is a compendium of everything his mouth will ever say, in that it is the monogram of all this man's thoughts and aspirations"[1]

Nowadays, 3D animation is playing an increasingly important role in the film industry. Movies comprising special effects such as Jurassic Park, The Matrix, or Lord of the Rings would not be the same, if they could not use 3D animation to give a face-lift to raw film material. Considering the constant growth of computer industry and specifically the increase in computational power, in 3D animation more realistic results are enabled. In this context, facial animation represents one of the biggest challenges. Since the age of three, children recognise and assign facial expressions. The fact that every human being is an expert in identifying facial expressions makes it very complicated to produce convenient representations. Conventional methods of facial 3D animation such as blend shape-based (cf. Chapter 4.3.1) or joint-based (cf. Chapter 4.3.3.1) systems have not succeeded in delivering believable photorealistic humanoid characters. Apparently, the last step towards realism is to simulate muscles connected to the skull and to the skin to generate proper facial animation.

The research project *"Artificial Actors"* at the Filmakademie Baden-Württemberg consists of programmers and artists who address the task of making the representation of virtual actors more credible.

1.1 Abstract

It is not exaggerated to consider physiognomy as the primary language of mankind. Through the appearance of the face, a baby recognises its mother and makes its first mental contact. Scientists even discovered animals using facial expressions as silent communication.

Speech is the major form of interpersonal conversation. Nevertheless, the display of emotions or thoughts by individual facial actions is much more intuitive. Besides that, the ability to perceive and comprehend facial expressions has been put in our cradle.

[1] **Arthur Schopenhauer**

Currently, the major developments in 3D animation are concentrated on the authentic representations of the human world. Within this complex field of research and development, creating anatomically realistic facial animation is proving to be one of the most complex challenges. Gathering theoretical knowledge in this field appears to be a task as boundless as improving the actual techniques of facial animation.

In this diploma thesis, we describe a model that uses an anatomically-based virtual representation of a human head. To make the appearance of the face more lifelike, we simulate a skull, skin, fatty tissue, and finally the most important part for facial animation: the muscles.

This model is implemented in Alias Maya®. The goal of this diploma thesis is not only to achieve best results in terms of appearance but to make it usable for an animator who is sufficiently capable of using Maya® and to make it possible to use it in real-time compromising quality. We will propose several techniques to simulate the anatomical structure of a head. To get an idea why these techniques are introduced, the anatomical constitution of the head will play a crucial role in this thesis. The major goal of this thesis is to design a system that combines the following qualities: real-time, usability, and the authenticity of appearance. To demonstrate the implementation discussed in this thesis, we will elaborate on the software we developed for this project.

1.2 Outline

Following the introduction (cf. Chapter 1), in the second chapter we will address the anatomy of the head (cf. Chapter 2). This serves as a foundation for further investigation considering the mathematical and physical models used. Besides the structure and behaviour of muscles, the constitution of the skin and the skull will be an issue. Special characteristics and functions of every component of the head will be described, to give a better understanding of how every component can be simulated.

At the beginning of Chapter 3, the background to the development of this thesis will be discussed. We will be going into Maya® in-depth; especially the script language MEL and the Maya® API will be introduced. Finally, we will address some important properties of the C++ programming language.

In Chapter 4, we will introduce existing approaches to the subject at hand. We will comment on prior models relating to this thesis. First, we will present a brief description of the status quo of research in this field which is fundamental to understand further illustrations. Next, we will delve into several models that are relevant to the simulation of anatomy.

In Chapter 5, the final implementation of the software for this thesis will be described. We will introduce several experiments that led to the final realisation. The final implementation of the software will be elucidated in-depth.

Chapter 6 forms the conclusion of this work. The results produced by our software will be evaluated. Furthermore, we will provide an outlook on what can be done in the future, predominantly in terms of optimisation, portability, and applicability in other research activities.

2 Anatomy of the Head

2.1 The Skull

The skull represents the uppermost part of the human skeleton. The outer appearance and the structure of the human skull are functionally related to the erect posture of the human. The spherical shape of the skull eases the free balancing of the head on the spine and serves as an ideal protection of the brain and the sense organs. The skull is composed of a number of bony portions. 8 can be found in the skull proper (neurocranium), 14 in the facial area (splanchnocranium or viscerocranium), and 7 associated bones (6 ear bones and the hyoid) in the skull. The skull proper, also called the cranial area, is the part of the skull directly surrounding the brain [MSN05]. The facial region consists of all the other bones of the skull. On the exterior, the cranial bones comprise the two frontal bones constituting the forehead. They fuse together in adulthood [MSN05].

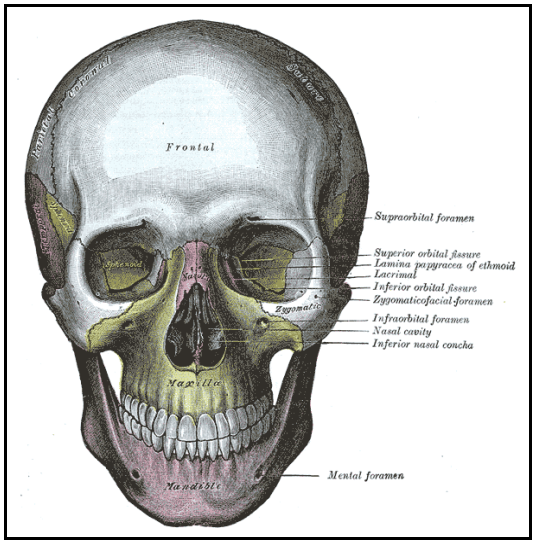


Figure 1: The skull from the front [GC84]

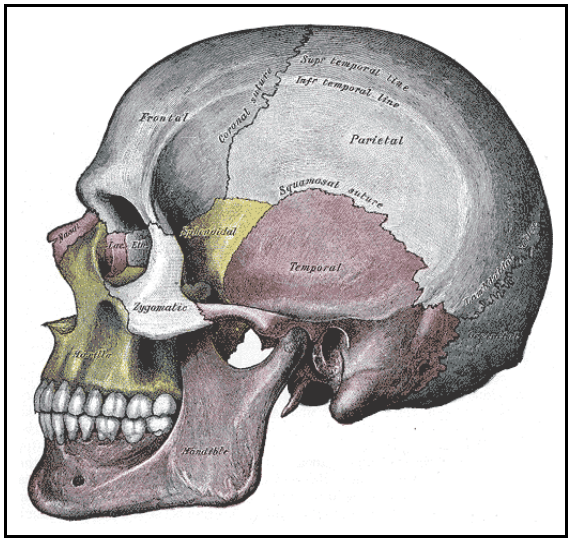


Figure 2: Side view of the skull [GC84]

2.1.1 The Cranial Bones (Neurocranium)

The distinction between the neurocranium and other skull portions is arbitrary due to the fact that its bones also participate in the configuration of the skeleton of the face. Bones of the neurocranium are single as well as paired [Hac00I]. The paired bones are represented by the tempo-

ral and parietal bones (see Figure 2). The single ones are embodied by the frontal, sphenoid and occipital bones (see Figure 1). The temporal bones form part of the skeleton of the sides and base of the skull. The frontal bone makes up the skeleton of the forehead. It represents a greater part of the roof of the circular cavity as well. The sphenoid (see Figure 2) is located at the base of the skull, taking part in the formation of part of the skull base, a small part of the side of the skull, and parts of the orbital and nasal cavities. The occipital bone forms the skeleton of the hind part of the skull [Hac00I].

2.1.2 The Facial Bones (Viscerocranium)

The viscerocranium is the region of the skull that forms the bones of the face [Hop00I]. The viscerocranium includes the following bones: nasal, lacrimal, maxilla, palatine, vomer, zygomatic, and mandible (see Figure 1 and Figure 2). A large portion of the face and neck is derived from structures known as pharyngeal arches. There are five pharyngeal arches numbered from 1 to 6; the viscercranium is primarily formed from Arch 1 and 2 [Hop00I]. Each contributes not only to the development of a particular portion of the skull but also to the creation of specific muscles, nerves, and blood vessels.

2.1.3 Function and Special Characteristics

The muscles and the bones compose the movement apparatus. The musculoskeletal system enables both: flexibility and body strength. The bones determine our human stature. They represent the framework which supports the body and protects the organs. They maintain the static structure of the head. In Chapter 2.3, it will be described how muscles are attached to the skin and to the skull. One part of the muscle, the origin, is attached to the skull. Consequentially, this part stays fixed to the skull. In Chapter 5.1.2.1, the mandible will play a crucial role because of its flexibility. Muscles attached to the lower jaw make up chewing movements including up and down as well as side to side grinding motions.

As a matter of principle, bones are the easiest elements to implement as they stay rigid and do not have to be deformed. On the other hand, they influence the superjacent elements significantly so that they have to be implemented as realistic as possible.

Unlike the rest of the human skeleton, the skull exclusively contains one joint: the mandible. Other joints and bones of the human body are commonly presented in a highly complex hierarchy which allows animators to easily control the character's motion. To animate a skeleton, the following methods can be used: forward kinematics (FK), inverse kinematics (IK) or IK/FK blending [Ali04a] (cf. Chapter 4.3.3.1, Chapter 5.1.1.2). In Chapter 5.1.1.2, an IK system will precisely be discussed. However, for the model used in this thesis, these methods will not mat-

ter. The skull will just be used as a reference point for the muscles and the skin. It will be represented as a polygonal geometry.

In this thesis, we will introduce a method how to copy the skull and fit it into a completely different target geometry (cf. Chapter 6.2.2), e.g. the head. So at last, only the shape of the skin has to be renewed but not the skull. By setting reference locators, the skull of the former skin can then be matched to the new skin shape.

2.2 The Skin

The skin covers the body and protects the deeper layers from injury from drying and from the invasion by foreign organisms. It is riddled with the endings of many sensory nerves. An adult's skin comprises between 15 and 20 percent of the total body weight. Each square centimetre has 6 million cells, 5000 sensory points, 100 sweat glands and 15 sebaceous glands [All05]. With all these different properties, the skin serves a diverse range of functions [MceNDI]:

- **Support -** The skin acts as a flexible physical support and cover for underlying tissues.
- **Excretion** - Waste materials such as salts and water are removed from the body through the skin's sweat glands.
- **Sensory function** - Through the extensive network of sensory receptors, we have perceptions of pressure, texture, temperature, and pain.
- **Protection** - The epidermis prevents dehydration of the internal. It prevents absorption of unintentional and potentially hazardous chemicals.

The skin is composed of the epidermis and the dermis. Below these layers the hypodermis is located which is not usually classified as a layer of skin.

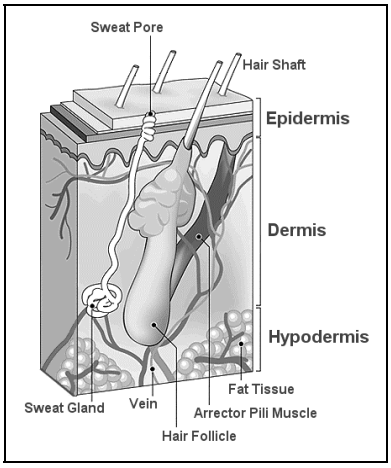


Figure 3: The skin [MceNDI]

2.2.1 Epidermis

The epidermis is the skin's outermost layer. The thickness of the epidermis varies in different kinds of skin. It is the thinnest on the eyelids with .05 mm and the thickest on the soles with 1.5 mm. The epidermis is composed of five layers [Bra04].

From bottom to top, the layers are denoted as stratum basale, stratum spinosum, stratum granulosum, stratum licidum, and stratum corneum. The stratum basale representing the bottom layer is composed of cells that are formed like columns. In this layer, the cells divide and push already built cells into higher layers. Cells moving into higher layers flatten and maybe die. The top layer of the epidermis, the stratum corneum, consists of dead, flat skin cells that peel away about every two weeks which means that every new layer of skin has a soft, glowing appearance [Bra04]. The lower levels of living cells are supplied with blood from underneath, while the upper dead cells solely need water to ensure they are kept fat and smooth. The epidermis is responsible for the colouring as it holds the skin's pigment.

There are three kinds of specialised cells in the epidermis. Melanocyte produces pigment (melanin), the Langerhans' cell is the vanguard defence of the skin's immune system, and the Merkel's cell's function is unknown.

2.2.2 Dermis

The dermis is the layer that lies below the epidermis; it consists entirely of living cells. It is composed of bundles of tough fibres which give the skin its elasticity and strength. Besides to that, there are blood vessels which supply these areas with vital nutrients.

The most important function of the dermis is breathing. The countless tiny blood vessels or capillaries end here in finely drawn networks from where they feed the outermost skin layer. The dermis also determines the tone and the appearance of the skin.

The dermis is seamlessly connected to the epidermis. It contains two layers: the papillary dermis and the reticular dermis. The papillary dermis, being the thin upper layer of the dermis, lies below the epidermis. It is composed of loosely interwoven collagen. In the deeper layers, the thicker reticular dermis can be found with its coarser and flat running bundles of collagen. Collagen fibres make up 70 % of the dermis and provide structural stiffness and strength. Elastin fibres are loosely arranged in all directions and are responsible for the skin being elastic. They are most commonly located near hair follicles and sweat glands and less in the papillary dermis.

2.2.3 Hypodermis / Subcutaneous Tissue

The hypodermis is the innermost layer of the skin. It consists of fat and connective tissue that contains larger blood vessels and nerves. This layer significantly contributes to the regulation of

the temperature of the skin and the body. The size of this layer varies throughout the body and from person to person.

The hypodermis is a cushion for the skin. It acts as a shock absorber to protect the bones. The fat of the hypodermis increases the tension of the skin. The development of the subcutaneous fat depends on sex, age, body region, mechanical demands, hormonal influence, nutrition, and other factors. Some areas of the face do not contain any fat such as the eyelids and the external ear. This tissue makes up the most important part considering the simulation of skin.

2.2.4 Function and Special Characteristics

Since skin determines the outward appearance of a human, it has to be realised as believably as possible. Therefore, some anatomical characteristics of this organ should be acquainted. In constitutive description, human skin is a non-homogeneous, anisotropic, non-linear plastic viscoelastic nearly incompressible material.

Non-homogeneity, anisotropic: Soft tissues are multi-composite materials including cells, intracellular matrix, fibrous, and other microscopical structures.

This means that mechanical properties of living tissues may vary from point to point within the tissue. Essential for modelling is the spatial distribution of material stiffness as well as the organisation of fibrous structures such as collagen and elastin fibres (cf. Chapter 2.2.2) which have some preferential orientation in the skin. Non-homogeneity denotes the dependence on coordinates along the same direction. If a material property depends on spatial direction, such material is denoted anisotropic. Facial tissue is both non-homogeneous and anisotropic. Nevertheless, there exists no quantitative data about these properties and their importance for modelling of relatively thin facial tissue is vague [Gla03].

Non-linearity, plasticity, visco-elasticity, quasi-incompressible material: Non-linear describes the behaviour of the skin considering the stress-strain relationship. The response of soft tissue is linear at low strains. At average strains, the tissue stiffness increases so the stress-strain relationship does not show linearity. At high strains, the stress-strain relationship becomes linear again until material destruction occurs.

Plasticity stands for the destructibility of the material. To the point the material tears, it shows visco-elastic characteristics which denote the combination of time-dependent fluid and solid properties. One characteristic of tissue time-dependent behaviour is stress relaxation or recovery.

Incompressible material refers to material that does not change its volume by the deformation. As soft tissue contains both incompressible and compressible material, in this thesis the facial tissue is called quasi-incompressible [Gla03].

The most important property of the skin is its elasticity. The ability to stretch the skin is not the same in every region of the human body. Summerfield [Sum83] measured the physical displacements of the facial feature points, especially around the mouth and his results indicate that displacements rarely exceed 25 mm during any kind of articulation. At some parts of the body, the skin has an immediate connection to the bones which makes it less flexible. The elasticity does not only differ from dissimilar parts of the body but from person to person. The skin decreases its elasticity with age and differs from man to woman. Women have weaker connective tissue due to hormonal causes. This results in a looser attachment of the skin and the underlying layers of tissue. With increasing age, the ability of the skin to regenerate the collagen fibres slows down. So the older one grows, the firmer the skin gets.

Another significant feature of the skin and very important for the recognition of facial expressions is the wrinkling, i.e. the skin bulges under muscle compression. These skin bulges may remain permanently with increasing age and constitute an essential attribute of the face.

At last, when discussing the appearance of the face, bumps and uneven spots have to be mentioned. They do not attract attention but, in the majority of cases, their effect is underestimated. Pores, pimples, or gleaming blood vessels change the exterior of a face so much that a person with a perfectly smooth skin would not look realistic.

2.3 The Muscles

The muscles are connected with bones, cartilages, ligaments, and skin either directly or through fibrous structures called tendons. In contrast to bones, they refine the general shape of the surface. Muscles compose nearly half of the total mass of a male adult body and fill in almost completely the space between the bones and the skin.

The muscles of the head can be classified into two groups: muscles of facial expressions (cf. Chapter 4.1.1) and muscles of mastication. The muscles' function depends on the movement of each muscle, the type of joint it is associated with, and to which side of the joint the muscle is fixed to. Muscles are typically attached to two places: one end is fixed to an immovable or fixed part. This end is called origin. The other end is attached to the other side of a joint that is able to move. In the case of facial muscles that end is not necessarily attached to the bone. The term for that end is insertion. When muscles of the face contract, the insertion end is pulled towards the origin [DON04].

It is important to say that muscles of the face are responsible for helping to communicate our feelings through facial expression. These expressions will be further discussed in Chapter 4.1.1..

Muscles of mastication have to prepare food for swallowing and digestion. Four pairs of muscles in the mandible make chewing movements possible. In Chapter 4.1.1, these and every other muscle of the face that has an important meaning in creating facial expressions will be classified more explicitly and they will be implicated with characteristic movements of the face.

Muscles are composed of muscle fibres. One element within the fibres are myofibrils. The basic component of myofibrils is sarcomere. Sarcomeres give facial muscles their striped appearance reflecting the alternating regions of isotropic (cf. Chapter 4.5.2) and anisotropic (cf. Chapter 2.2.4) material inside muscle cells. The isotropic bands contain actin fibres; the anisotropic bands contain myosin and actin fibres. Both fibres actin and myosin are important for cell movements. They can be found in all body cells, especially in muscle cells. Muscle cells are grouped together into bundles of fascicles. The muscle is finally made up of groups of these bundles. The fibres in a single fascicle are parallel, but the alignment of fascicles in skeletal muscles can differ as well as the relationship between fascicles and the associated tendon.

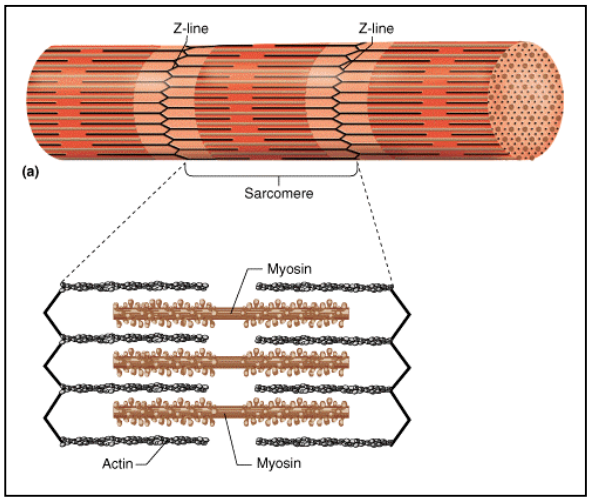


Figure 4: Sarcomere [Gra05I]

Four patterns of fascicle organisation form parallel muscles, convergent muscles, pennate muscles, and circular muscles (sphincters) [May04I] [Pai99]:

Parallel (Fusiform) muscles: Most of the muscles of the face are of the parallel type. This kind of muscle has its fibres lying primarily parallel to its longitude axis.

Convergent muscles: Convergent muscles are based over a broad area but congregate at one attachment point. The chest muscle is one of this kind, for example.

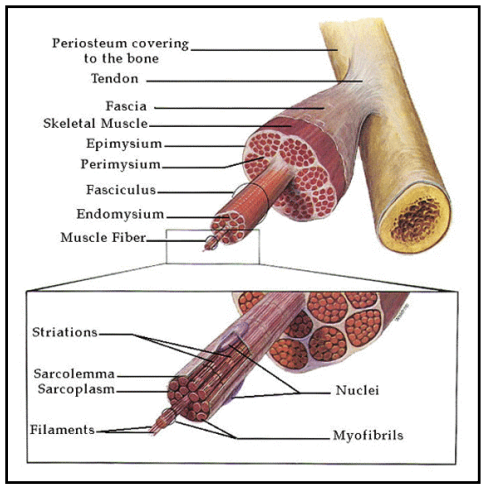


Figure 5: Muscle structure [Dub04]

Pennate muscles: In a pennate muscle, the fibres all shape a common angle with the tendon. These muscles do not contract as much as a fusiform muscle, but they provide more direct force.

Circular muscles, sphincter: In a circular muscle, the fibres are concentrically arranged around an opening. When the muscle contracts, the radius of the opening decreases. Sphincter muscles guard entrances and exits of internal openings like the mouth.

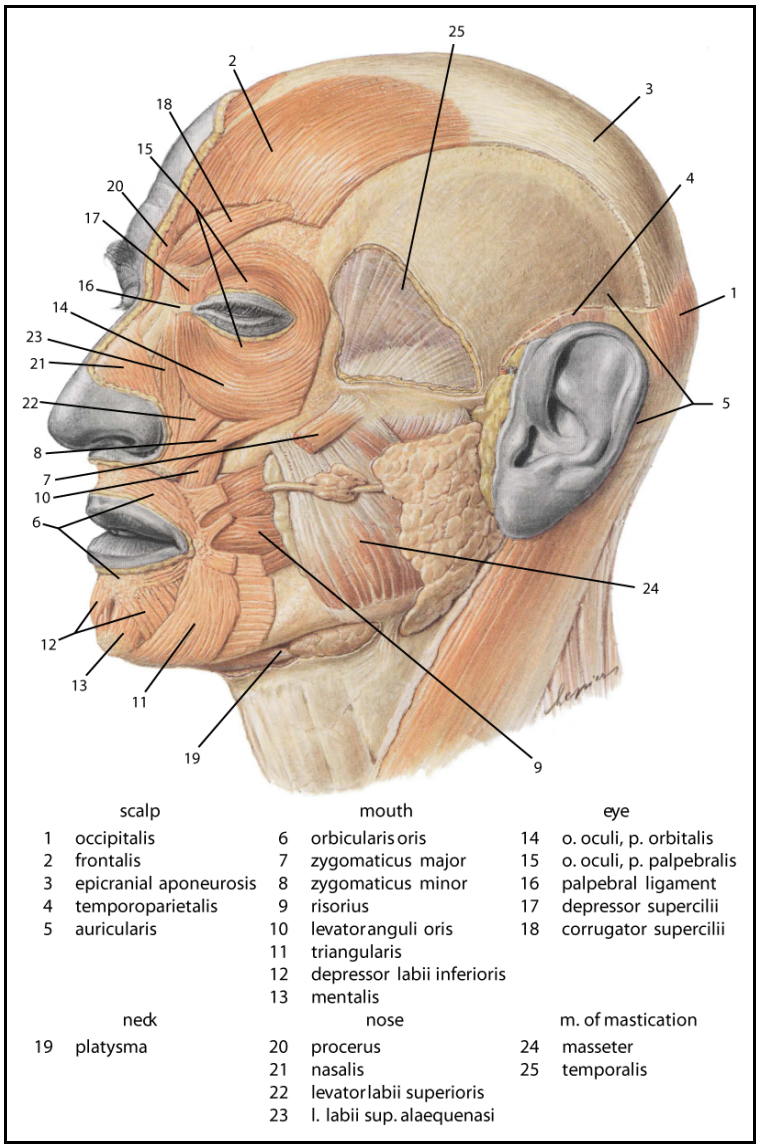


Figure 6: Facial muscles, side view [Kae03]

To get a better understanding of the parts that make up the facial musculature, all muscles are divided into groups [FacNDI]:

2.3.1 Scalp

The scalp consists of long, flat, smooth muscles that are able to move forward and backward. The limber scalp is very important to the nutrition of the skin and hair, helping to keep the

glands and blood vessels healthy. Three little muscles which are positioned in front, above, and behind the external part of the ear are also counted among the muscles of the scalp. They move very little but intertwine with other muscles and tissues (see Figure 6: muscle 1 – 5).

2.3.2 Mouth

Three multifunctional muscle groups surround the mouth. They make us pucker or grin, help other muscles with chewing and pull the corners of the mouth up or down, and numerous other necessary task. One of these task is to give expression and structure to the face (see Figure 6: muscle 6 – 13).

2.3.3 Eye

The muscles of the eye include the eyelid, the surrounding circular muscle and the inside of the eye socket. These muscles provide voluntary and involuntary blinking, movement of the eyeball as well as tear duct control (see Figure 6: muscle 14 – 18).

2.3.4 Nose

Around the nose, several small muscles can be found that extend and interconnect with other muscles running from the scalp to the upper lip. They control the nostrils so that it is possible to open or compress them as well as to lift the upper lip and pull down the brows at will (see Figure 6: muscle 20 – 23).

2.3.5 Muscle of Mastication

The muscles of mastication can be categorised into two different purposes. The first group contains three pairs of muscles that raise the mandible to close the mouth. The second group includes one pair that can open the mouth and make grinding actions possible (see Figure 6: muscle 24, 25).

2.3.6 Neck

The neck consists of only one muscle that belongs to the facial musculature. This muscle tenses the skin of the neck and pulls corners of the mouth slightly.

2.3.7 Function and Special Characteristics

Two types of contraction are classified. Isotonic and isometric contraction. If a stimulated muscle is in a state of not being shortened, it simply tenses. The correct term for this is *"isometric"* contraction. Isometric means *"same length"*. In contrast to the isometric contraction, the isotonic

contraction denotes *"same tension"*. That means the muscle is allowed to shorten. As the longitudinal axis of the muscle shortens, the other two axes extend almost equally so that the volume of the muscle stays nearly constant. Since it is proved that the volume is just tending to preserve [WG97], further muscle models can extremely be simplified.

To characterise a muscle, it has to be stated that it is represented by its shape and its biomechanical model of deformable soft tissue. This model claims forces to act along fibre tangents [Gla03]. Parallel muscles that are also fusiform muscles have fibres running parallel to one another. Since the cross-sectional area of the tendon is much smaller than the cross-sectional area of the central region of the muscle, the fibres must taper significantly as they approach the tendon connection. When the fibres in the centre of the muscle are activated, they tend to shorten and bulge. Neighbouring fibres push away each other, causing peripheral fibres to be displaced farther away from the central axis of the muscle, forming a curve finally. Pennate muscles act the same way but can produce their bulge in different directions. Sphincter muscles do not have a bulge, but they also try to preserve their volume constantly along their shape. To understand why the bulge of a parallel muscle shapes a curved gradient, a closer look at tendons has to be taken. Tendons are stiff because they represent the junction to the bones. Thus, they cannot be deformed during contraction. As mentioned above, the muscle itself has other properties concerning its stiffness.

The geometrical arrangement of the fibres and connective tissue plays an important role in determining muscle force and mechanical behaviour. Tendons vary considerably in shape, ranging from flat bands to cylindrical cords. They may be short and thick or long and thin. The shape of the tendon affects its physical properties and also determines how the muscle fibres are geometrically arranged. Muscle fibres generally converge to tendons in parallel arrangements [Kat03I].

Jenson and Davy assess that in biomechanical analysis, whole muscles are usually represented as single vectors with a certain action line and a variable force magnitude[JD75]. The action line of a muscle may be considered to go directly from origin to insertion, or it may be assumed to follow the centres of the cross sections of the muscle. The force vector is generated by a pennate array of muscle fibres running along the action line of the tendon that contributes to the force and motion at the origin or insertion sites. The other components of the muscle are perpendicular to the line of action of the tendon and are not involved in forcing along this line. The greater the pennation angle, the less force a muscle fibre contributes along the line of action [Kat03I].

3 Background Development

3.1 Alias Maya®

Maya® is a system for character animation and visual effects designed to be used by a professional animator. Based on a procedural architecture denoted as the dependency graph, Maya® offers incredible flexibility for generating digital images of animated characters and scenes [Ali03a].

With Maya®, 3D models can be created and edited in a variety of modelling formats and they can be animated using Maya®'s suite of animation tools. Maya® allows us to create convincing visual simulations of rigid and soft body objects interacting in the physical world using computational dynamics and particle tools. Maya® also provides a range of tools that allow us to render an animated 3D scene resulting in photo realistic imagery and animated visual effects [Ali04a].

3.1.1 Basics

Maya® offers a very flexible workflow that does not imply numerous constraints. In most cases, Maya® provides more than one way to solve a problem. The user is free to choose his most convenient approach to come to a successful solution.

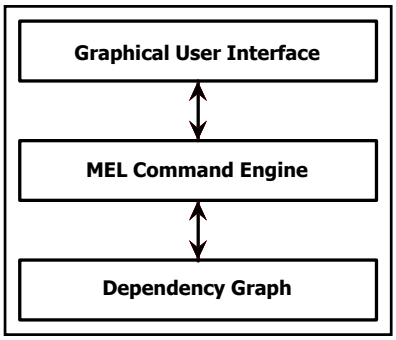


Figure 7: Schematic diagram, Maya [Gou03]

Figure 7 shows Maya®'s major components and the way Maya® functions. The interaction with the Maya® core works over the Graphical User Interface. Every change a user makes, every selection and interaction is translated into a MEL (Maya Embedded Language) command that is conveyed to the command engine. This command engine works as an interpreter that finally executes the MEL commands.

Most of these MEL commands operate on the dependency graph that represents the scene in Maya®. The dependency graph is like a series of instructions storing the information how to get the current scene starting from scratch. It embodies the data flow model Alias implemented in Maya®. The dependency graph is composed of a quantity of modules called nodes that can be connected to each other. Each of these nodes holds a specific information. This information

is commonly referred to as attributes. An attribute is a particular property of a given node. When an attribute of a node is modified, the dependency graph verifies if the attribute affects any output. If it affects any output, each of these outputs is marked dirty meaning that its cached value is obsolete and needs to be recomputed (see Figure 8). In Chapter 3.1.3, the computation of the output data will be illustrated more precisely.

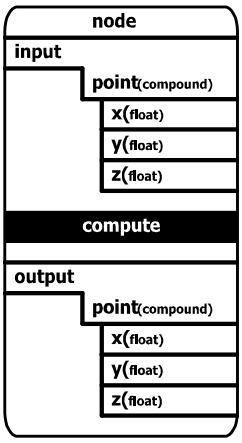


Figure 8: Node structure [Gou03]

As a result of this structure, nodes can be connected to other nodes and a complex network of interacting nodes can be constructed. To construct this collection of nodes, MEL can be used. If a more low-level access is required or performance plays a significant role, the API using C++ should be preferred.

3.1.2 MEL

MEL (Maya Embedded Language) is a powerful command and scripting language that provides direct control over Maya®'s features, processes, and workflows. Due to the fact that Maya®'s user interface is set up by MEL scripts and procedures, typing in MEL commands is a quick alternative to the selection of menu items or executing other actions.

The language MEL is derived from UNIX shell scripting. Both scripting languages share the property of having commands that are directly executable. MEL commands are interpreted and, in contrast to the API data structure, they do not have to be manipulated.

MEL includes over 600 commands and 75 functions [Gou03]. The most users generally execute MEL commands to perform tasks such as creating custom effects, writing macros, bypassing the user interface or creating procedures and scripts for custom modelling, animation, dynamics, and rendering tasks. In the software of this thesis, MEL serves to prepare the scene for further computations. It helps to combine several steps to one bundle that can directly be executed. This prevents the user from accomplishing single tasks that are time-consuming and inconvenient.

As mentioned above, MEL is a simply to use programming language. Following is an example of a MEL command to create a sphere:

```
sphere -radius 27.5 -name mySphere;    // create sphere object
```

When a MEL command is executed, actually one of Maya®'s C++ functions is called. When a sphere is created from the user interface, a *"sphere"* command is indirectly called which actually creates the sphere object. In Chapter 3.1.3, the focus is on creating objects like a sphere.

MEL cannot only create nodes for polygonal (cf. Chapter 4.2.2) or NURBS (cf. Chapter 4.2.3) objects but deformer nodes that can be used to change control points of an input object, for instance. In this software, MEL creates a deformer or connects attributes. It also performs tasks such as creating a NURBS sphere or determining the number of control vertices of a NURBS curve. But the more important part of this software consists of deformer nodes that are developed utilising the C++ API of Maya®.

3.1.3 Maya® C++ API

The standard C++ programming language can be used to extend existing Maya® functionality. With the C++ API, a developer is enabled to create native Maya® plug-ins that fit into the Maya® environment seamlessly. These plug-ins are a powerful technique to extend Maya®. API stands for *"Application Programming Interface"* and refers to a set of C++ classes that provide internal access to Maya® tools.

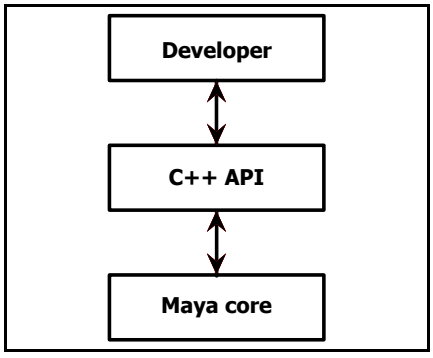


Figure 9: Programming Interface [Gou03]

According to Figure 9, the API resides on top of Maya®'s core so the developer has no direct access to Maya®'s internal functions and data. On the one hand, having no access to the Maya® core leads to several limitations. On the other hand it is a common procedure having the developer abstracted from the actual details of Maya®'s current implementation. By not exposing the present implementation to the developer, Maya®'s engineers are able to change and improve Maya®'s core without running the risk to destroy code written by external developers. The API also provides a certain level of protection from possible misuse so it can prevent a plug-in from deleting critical data. Hence, nearly all methods return an error message which simplifies debugging and preventing from having a false implementation.

"Source code compatibility is a goal of the Maya API. That means that plug-ins written for earlier Maya releases can recompile without any source code changes in order to create a plug-in for the current version. If a

source code incompatibility occurs, the documentation for the new release will contain detailed instructions on what changes need to be made to the plug-ins." [Ali04b]

All Maya® class names start with a capital M (cf. Table 1), for example: **`MObject`**. **`MObject`** is used to get an access to all the different data types. In reality, the **`MObject`** is just a handle to another object inside the core so only the Maya® core knows exactly what this handle refers to. To create, edit, and delete such an **`MObject`**, the developer needs a handle on it. This handle is denoted as function set. The classes that provide suitable function sets begin with an MFn (cf. Table 1). For example, an **`MFnNurbsCurve`** can be instantiated with an **`MObject`** of type **`kNurbsCurve`** (object kind: NURBS curve). A function set is owned by the user and allows operations on internal objects of Maya®'s core.

Prefix	Logical Grouping	Examples
M	Maya® class	MObject, MPoint, M3dView
MPx	Proxy object	MPxNode
Mit	Iterator class	MItDag, MItMeshEdge
MFn	Function set	MFnMesh, MFnDagNode

Table 1: Class name prefixes

All Maya® classes starting with MPx define proxy objects. In this case, proxy means that the actual node, a developer defines, is not actually used in the dependency graph. In fact, Maya® creates two objects. One is just a referenced second object kept internally. The other one the user owns does not exist in the dependency graph. The internal object is the real object, the other one is just a proxy. **`MPxDeformerNode`** is an example of a proxy. It is derived from **`MPxNode`** and serves to create a node that has a geometry as input and produces a deformed output geometry. **`MPxCommand`**, for example, enables the developer to create a command like *"sphere"* (cf. Chapter 3.1.2) that allows creating a sphere by determining the control points or polygons.

Maya® also provides iterator objects, starting with MIt that allow the developer to iterate over elements of an object such as the dependency graph or a mesh. **`MItGeometry`**, for instance, allows the user to iterate over control vertices of a NURBS curve (cf. Chapter 4.2.3).

So why should the developer use the C++ API instead of MEL? With MEL, the user can perform almost every task. The C++ API is typically used for specific functionality that cannot be found in the MEL interface. Another undeniable advantage of the C++ API is its performance rate. Since it is not interpreted as MEL and much more machine-oriented, this makes it indispensable for elaborate tasks such as intricate calculations or processing of a high amount of elements.

Following is an extendable list of all areas and functionality in Maya® which can be extended using the API [Gou03]:

- **Commands**
- **Dependency Graph Nodes**
- **Tools/Contexts**
- **File Translators**
- **Deformers**
- **Shaders**
- **Manipulators**
- **Locators**
- **Fields** (a volume in which various forces can be applied)
- **Emitters** (controller that determines the properties of a particle flow)
- **Shapes**
- **Solvers** (controller of a series of bones for an own IK)

In Chapter 5, the MEL and C++ API implementation will be detailed. For further information on specific MEL commands or API classes, the Maya® Online Help can be used.

3.2 .NET – C++

The plug-in of this thesis was developed using the Microsoft Visual C++ .NET environment. In the preferences window of the .NET project, the developer is able to define events that are executed before or after the compilation of the plug-in. With *"mcp.exe",* an application provided in the Maya® development environment, it is possible to communicate with Maya® by opening a command port in Maya®'s script editor. This way, .NET induces to execute scripts to control the testing environment.

Special classes or methods used in the plug-in will not further be explained as the Maya® Online Help provides sufficient information. In addition to using standard C++ statements such as the for-loop, the if-statement, or method calls, the plug-in deploys a set of error macros that display the error message including the line and the file where the error occurred.

A big advantage of the Maya® API is that it prevents from handling intricate memory management. Memory does not need to be reserved explicitly for arrays or objects. An easy use of methods is enabled without caring about memory garbage, the plug-in produces.

4 Previous Work

4.1 Facial Animation

4.1.1 The Facial Action Coding System (FACS)

Dr. Paul Ekman was one of the first scientists who described facial expression, analysed the muscular actions that create them, and finally categorised them. Scientists and technicians interested in pattern detection use FACS in their work when they need to distinguish the exact actions that the face can perform and to know which muscles produce them.

Dr. Paul Ekman found cross-cultural agreement in the recognition of happiness, sadness, surprise, fear, anger, and disgust, the top categories of facial expression (Action Units or AUs, correlation of muscle movements and facial expression). Thus, he tried to sort them with a number of limitations. *"One constraint in the development of FACS was that it deals with what is clearly visible on the face, ignoring invisible changes […], and discarding visible changes too subtle for reliable distinction. […] Another limitation was that FACS would deal with movement, not with other visible facial phenomena. These other facial signs would be important to a full understanding of the psychology of facial behaviour, but their study requires a different methodology."* [Ekm78]

His approach to systematise the facial language is to break it into minimal units. With these minimal units, combinations can be created that produce almost any behaviour.
At the research project *"Artificial Actors"* at the Filmakademie Baden-Württemberg, FACS served as basis for every approach of character animation. In Chapter 4.3.4, this will emerge more clearly. Hence, the results of FACS are the motivation of using a muscle-based system for animating a virtual actor. Compare http://research.animationsintitut.de where a detailed description of FACS can be found (→Expression Repertoire).

In Chapter A.2, each action unit is listed. Compare Figure 6 to locate the facial muscles that contribute to specific facial expressions.

Figure 10: AUs, results of the Adaptable Facial Setup

Figure 10 shows an example of the results created by the *"Adaptable Facial Setup"* that was developed at the research project at the Filmakademie Baden-Württemberg (cf. Chapter 4.3.4). The first expression is composed of AU 12 and AU 25. The second of AU 16 left and AU 1. The third example shows disgust: a composition of AU 10, AU 16, and AU 4.

Ekman categorised the types of messages conveyed by nonverbal behaviours. Emotions, including happiness, sadness, surprise, fear, anger, and disgust express messages about physical and emotional states and are relatively involuntary and stereotyped. Most of the specific messages are communicated by the face and resemble in all known human cultures. Culture-specific symbolic communicators are learned such as the wink or the handshake that convey messages similar to short verbal phrases. But it is safe to say that the variety of configurations of equivalent behaviours at this level make them difficult for computational measurement. Therefore, FACS is a good foundation for further scientific approaches on this topic.

4.2 Computer Graphics Basics

"Computer graphics (CG) is the field of visual computing, where one utilises computers both to generate visual images synthetically and to integrate or alter visual and spatial information sampled from the real world" [Wik01I]. It deals with the conversion of a geometric or mathematical definition of an object into a visualisation, a two-dimensional projection that synthesises the appearance of a real object.

Conventionally, computer graphics creates pictures by generating a very detailed geometric description, applying a series of transformations that the viewer is able to perceive, and objects in three-dimensional space. If the object is described by a mesh of triangles, finally, the individual triangles can be coloured. This process is known as rendering.

This section touches only some of the subjects relating to computer graphics. Since we assume the readers of this thesis to have basic knowledge about computer graphics, only the issues concerning this thesis will be addressed. This will be the topics where a basic familiarity is recommended to understand further proceedings.

4.2.1 Vectors and Matrices

4.2.1.1 Vectors

A vector is a specific type of representing related elements in a one column table format. It represents both the direction and the magnitude of a quantity such as forces, velocities, or the position of a point in world space. A vector uses the variables x, y and z to show its magnitude in its axes in the world coordinate system. A standard vector looks like the following:

$$\vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Equation 1: A standard vector

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \bullet \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = x_1 * x_2 + y_1 * y_2 + z_1 * z_2$$

Equation 2: The dot product

The magnitude or length of a vector is defined as: $|\vec{v}| = \sqrt{x^2 + y^2 + z^2}$

The normalised vector of v is calculated as follows: $\vec{v}_{norm} = \frac{\vec{v}}{|\vec{v}|}$

Vectors enable mathematical procedures such as addition, subtraction, and multiplication. There are two methods of vector multiplication. The first produces a scalar result represented in a single value including the magnitude of the multiplication (cf. Equation 2). The second produces a vector result (cf. Equation 3). The scalar result is determined by a method known as dot product and is produced by the addition of the products of all corresponding values of every axis.

The second method of vector multiplication is known as cross product. The cross product is calculated by using the following formula that produces a vector perpendicular to the plane defined by two multiplied vectors [Duc00]:

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} x_1 * z_2 - z_1 * y_2 \\ z_1 * x_2 - z_1 * z_2 \\ x_1 * y_2 - y_1 * z_2 \end{pmatrix}$$

Equation 3: The cross product

The vector type used for computer graphics contains four values denoting the x-, y- and z-axis and the homogenous value w. The w value, for example, is used to project three-dimensional world coordinates to the two-dimensional screen. The w value is generally set to 1.

4.2.1.2 Matrices

Matrices comprise their elements in a table. The elements are presented in rows and columns forming a grid that is enclosed in brackets. Matrices are used to transform points in 3D space and are of size 4 x 4. To transform a vector by a matrix, the vector has to be multiplied by the matrix to produce a new vector [Duc00].

$$\begin{pmatrix} x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x_v \\ y_v \\ z_v \\ w_v \end{pmatrix} = \begin{pmatrix} x_1 * x_v + y_1 * y_v + z_1 * z_v + w_1 * w_v \\ x_2 * x_v + y_2 * y_v + z_2 * z_v + w_2 * w_v \\ 0 * x_v + 0 * y_v + 0 * z_v + 1 * w_v \end{pmatrix} = \begin{pmatrix} x'_v \\ y'_v \\ z'_v \\ w'_v \end{pmatrix}$$

Equation 4: Matrix vector transformation

4.2.1.3 Transform Matrices

There are numerous different matrices which cover a specific kind of vector transformation in 3D space. Equation 5 shows the translation matrix that moves a point in 3D space relative to its origin by the distance d_x, d_y, and d_z [Duc00]:.

$$\begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Equation 5: The translation matrix

$$\begin{pmatrix} s_y & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Equation 6: The scaling matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Equation 7: Rotation matrix about the x-Axis

The scaling matrix transforms an object relative to the origin of the world coordinates. s_x, s_y, and s_z denote the scale factor in every dimension (cf. Equation 6). Equation 7 depicts a rotation matrix. In the illustrated case, it concerns a matrix to rotate a vector about the x-axis. ϕ represents the angle to rotate about.

A transformation matrix can serve to describe a transformation from the local space of an object to the scene's world space. To get the transformation matrix that combines the translation, the scaling, and the rotation matrices they have all to be multiplied:

$$\mathbf{M = T * S * R_x * R_y}$$

Equation 8: Matrix multiplication

4.2.1.4 Inverse Matrix

If a transformation in space is done by a matrix, it can be done conversely. If one computes the world space vector of a local space vector by multiplying the corresponding transformation

matrix, the original local space vector can be computed from the world space vector by multiplying the inverse transformation matrix. Matrices that shall be inversed are required to be square matrices meaning that they require the same amount of columns and rows. There are several computation methods to obtain the inverse matrix such as the Gauss-Jordan elimination or the Gauss elimination method (also serves for solving a linear equation described in Chapter 4.2.1.5).

The inverse of an n x n matrix A is another n x n matrix A^{-1} such that $A^{-1}A=I$ where I denotes the identity matrix. The solution to the linear system Ax=b can be written $x=A^{-1}B$ (cf. Chapter 4.2.1.5 for a detailed description of solving a linear equation)

4.2.1.5 Linear system of equations

To solve a matrix equation of the form Ax=b, the Gaussian elimination [WolNDI] can be preformed. The Gaussian elimination starts with the system of equations:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}$$

Equation 9: System of equations

Next, the *"augmented matrix equation"* has to be composed:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1k} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2k} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} & b_k \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}$$

Equation 10: Augmented matrix equation

The column vector in the x variables is carried along for labelling the matrix rows. Now, elementary row operations are performed to put the augmented matrix into the upper triangular form[WolNDI]:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1k} & b_1 \\ 0 & a_{22} & \cdots & a_{2k} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{kk} & b_k \end{array}\right]$$

Equation 11: Upper triangular form

Finally, the equation of the kth row for x_k has to be solved, then it has to be substituted back into the equation of the (k-1)st row to obtain a solution for x_{k-1}, etc. according to following formula:

$$x_i = \frac{1}{a'_{ii}}\left(b'_i - \sum_{j=i+1}^{k} a'_{ij} x_j\right)$$

Equation 12: Solving of every row

Linear equations can often be found in solving geometric problems. An example, where the solving of a linear equation is required, is the calculation of NURBS (cf. Chapter 4.2.3) or the Radial Basis Function (cf. Chapter 4.3.2.1)

4.2.2 Polygons

Simple polygonal meshes are used to display geometric detail in applications that are easy and interactively to use. These meshes of planar polygonal facets are used to approximate objects in a 3D scene.

In the simplest case, a polygonal mesh is a structure consisting of polygons represented by a list of linked vertex coordinates. Therefore, the description of an object is finally stored in a list of vertices. Usually, other geometric information such as polygonal normals, vertex normals, edge attributes, or vertex attributes such as texture coordinates (cf. Chapter 4.2.2.4) are also stored in this list.

Polygonal models can be generated by using a three-dimensional digitiser by a mathematical description or manually. If a polygonal mesh is created manually, it usually originates from one existing polygon, vertex, or edge. Next, vertices or polygons can be attached. Given a polygon, it can be splitted, cut, extruded, subdivided, or it can be merged with other polygons. Vertices can

be moved to create or deform a mesh. Consider that moving a vertex changes all adjacent polygons.

4.2.2.1 Generating polygonal meshes

A polygonal object can be generated using a mathematical description. A commonly used method is the creation of a two-dimensional shape described by a curve. This shape is discretised into a polyline. Next, a 3D shape is generated by sweeping this cross-section along a path. If the cross-section is scaled, twisted, teetered, or bevelled along the path or morphed into another shape, standard 3D primitives such as a cube, a sphere, cylinder, a cone, or a torus can be created. These methods are called *"lofting"* or *"extrusion"*. Given a 2D shape lofted or extruded along a path, the resulting object is finally discretised into a mesh of polygons. To discretise a curve that represents the path or the cross-section, intervals are defined. It can then be stepped through these intervals along the curve or around the cross-section and polygons can finally be constructed.

A commonly used method to create smooth polygon meshes such as a head is called box modelling. Most of professional 3D applications provide the mesh smooth function that smoothes the mesh and tessellates it. Given a box modelled and smoothed, an animator is able to create a sphere with the correct configurations specified. Next, the animator is able to bulge the sphere by attaching another box to the existing box. Based on this technique, smooth and detailed objects can be generated. In the majority of cases, the original box model serves as a proxy object so that manipulation to the smoothed mesh can immediately be noticed. Compare Figure 11 to see a simple shape of a hand generated by using the box modelling technique.

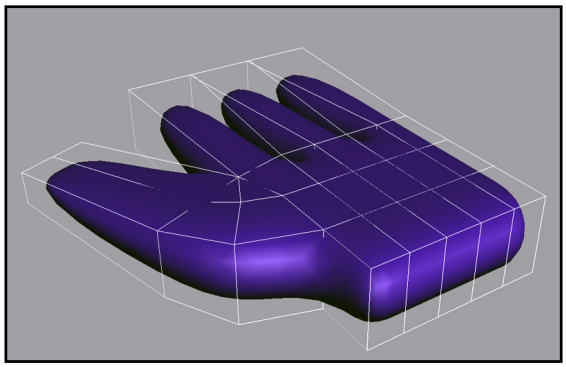


Figure 11: Box modelling a hand

Another method to create polygons is by applying procedural generation methods as it is done to produce fractal objects. However, these methods are not of particular interest for our work.

4.2.2.2 Deforming polygonal meshes

To deform polygonal meshes, every 3D application provides different methodologies. In most cases, the meshes are deformed by configuring attributes of a deformer or by influencing a deformer object. The twist deformer winds an object according to its attributes such as the angle

or the high and low bound. Other examples for deformers where only attributes are set are the bend, the flare, the wave, the squash, the taper, or the noise deformer.

If the deformer uses another object as influence, the deformer translates the data of the input influence object and with the additional information of the attributes, the deformer manipulates the data of the object. A popular deformer that has an additional influence object is the free-form deformer (FFD). This deformer has a lattice as input that surrounds the deformed object. By moving vertices of these lattices, the object is deformed. Other deformers of this kind are the blend shape (cf. Chapter 4.3.1), the cluster (cf. Chapter 4.3.3.2), or the wrap deformer (cf. Chapter 5.1)

Compare Chapter 5.1 to get a better grasp how a deformer works particularly in Maya® and how it is developed.

4.2.2.3 Barycentric Coordinates

Barycentric coordinates are the coordinates of a point defined by its weighted association to other points [WolNDI]. The barycentric coordinates are commonly used in relation to the vertices of a triangle. The barycentric coordinates (b1, b2, b3) of a point P with respect to a triangle with vertices V1, V2, V3 are depicted in Figure 12. It is important to say that for a point inside the triangle, the coordinates always sum to one. Equation 13 shows how to compute the barycentric coordinates and how to define the point inside the triangle knowing the barycentric coordinates [Par01].

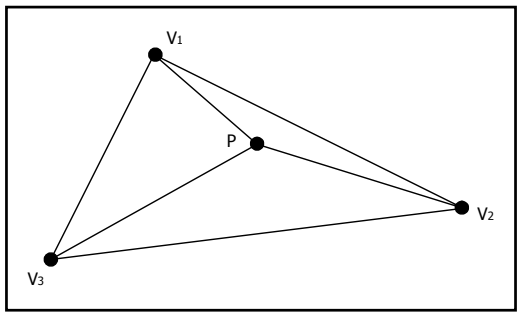


Figure 12: Barycentric coordinates of a point

$$b1 = \frac{Area(P,V_2,V_3)}{Area(V_1,V_2,V_3)} \quad ; \quad b2 = \frac{Area(P,V_3,V_1)}{Area(V_1,V_2,V_3)} \quad ; \quad b3 = \frac{Area(P,V_1,V_2)}{Area(V_1,V_2,V_3)}$$

$$P = b_1 * V_1 + b_2 * V_2 + b_3 * V_3$$

Equation 13: Calculation of barycentric coordinates

In the case of our software, the barycentric coordinates are calculated by solving a linear system of equation defined by the last formula of Equation 13.

Based on Equation 14 and the method we proposed in Chapter 4.2.1.5, the barycentric coordinates can be determined by the following formula:

$$\begin{bmatrix} x_{V1} & y_{V1} & z_{V1} \\ x_{V2} & y_{V2} & z_{V2} \\ x_{V3} & y_{V3} & z_{V3} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} x_P \\ y_P \\ z_P \end{bmatrix}$$

Equation 14: Determination of barycentric coordinates

Barycentric coordinates are used to implicate a triangular polygon of a mesh with a point on its surface. For example, if the closest point on a surface to another object has to be defined, the barycentric coordinates of this point are stored if the user wants to store the relative position of the closest point to the surface.

4.2.2.4 UV Texture Coordinates

Another method to store relative surface coordinates is to identify UV texture coordinates. UVs are 2D coordinates that provide the required information to apply textures to a surface. UVs of a polygon need a specific arrangement so that textures appear properly when applied to the surface material. The task of the rendering engine is then to find the appropriate UV coordinate for pixels inside a polygon.

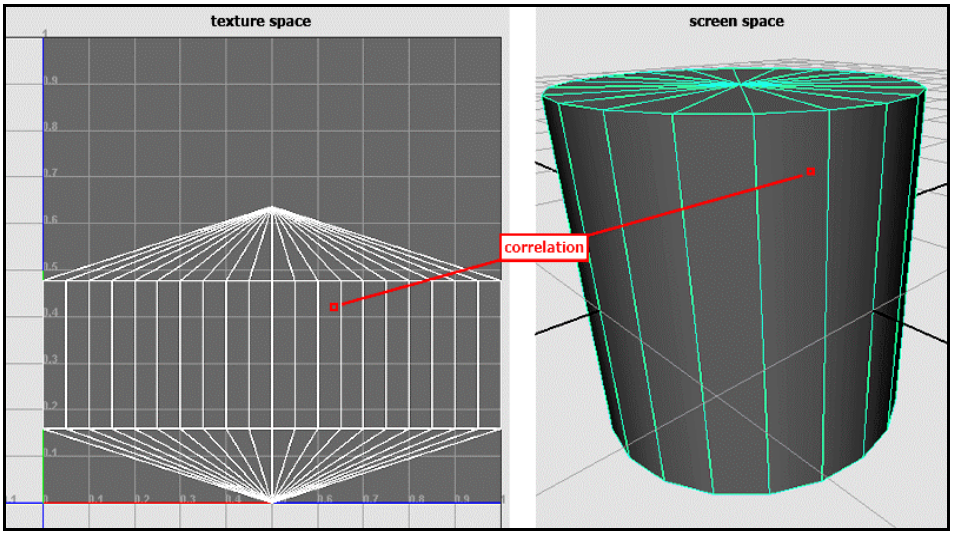


Figure 13: Correlation texture space to screen space

The most common way to bring a 2D texture onto a 3D surface is the inverse mapping. This technique brings the information of every single pixel seen on the screen into the image data of the texture space. Forward mapping is the other way round whereas a texture coordinate (texel)

produces a pixel on the screen (see Figure 13). This algorithm causes problems due to holes or overlaps in the texture image.

If UV texture coordinates serve to locate the relative position of a surface point, overlapping polygons lead to indeterminate correlations. Given the UV coordinate of one relative position, several allocations may result when mapping back. That is because one UV coordinate inside the overlapping polygons shares several points on the surface.

Maya® provides a function to prevent overlapping polygons in texture space called *"Layout UVs"*. The texture coordinates will be put into a new layout where all intersecting polygons are separated. Avoiding overlapping polygons in texture space the polygonal mesh maintains its allocation of UV coordinates independent of shape deformation. We assume that Maya® internally defines UV coordinates by using a similar approach as introduced in the previous chapter.

4.2.3 NURBS curve

NURBS [AltNDI] stands for Non-Uniform Rational B-splines. Altmann describes a NURBS curve as follows [AltNDI]: a NURBS curve is defined by a set of weighted control points, the curve's order and a knot vector. The advantage and the reason why NURBS are used in many 3D animation applications is the wide range of flexibility they provide. NURBS are easy to use to describe analytical shapes such as a sphere or a cone or free form shapes such as a curve.

A NURBS curve C(u), which is a piecewise rational polynomial function based on vector values, is defined as:

$$C(u) = \frac{\sum_{i=0}^{n} w_i * P_i * N_{i,k}(u)}{\sum_{i=0}^{n} w_i * N_{i,k}(u)}$$

w_i : weights of control points
P_i : control points(vector)
$N_{i,k}$: normalized B - spline basis functions of degree k

Equation 15: NURBS function

These B-splines are recursively defined as:

$$N_{i,k} = \frac{u - t_i}{t_{i+k} - t_i} * N_{i,k-1}(u) + \frac{t_{i+k+1} - u}{t_{i+k+1} - t_{i+1}} * N_{i+1,k-1}(u)$$

$$N_{i,0}(u) = \begin{cases} 1, \text{if } t_i <= u < t_{i+1} \\ 0, \text{else} \end{cases}$$

Equation 16: B-splines function

Where t_i are the knots defining a knot vector $U=\{t_0, t_1, \ldots, t_m\}$.

4.2.3.1 The Knot Vector

The knot vector U uniquely determines the B-splines following Equation 16 [AltNDI] . The relation between the count of knots (m+1), the degree (k) of $N_{i,k}$, and the number of control points (n+1) is given by m=n+k+1.

The degree is a mathematical property of the curve that defines how many control points per span are available for modelling. It ascertains the curve's degree to bend. Degree 1 causes the control points to be directly connected by straight lines (also denoted as *"polyline"*). Degree 2 can have one bend between edit points. Most 3D animation software applies degree 3 (cubic NURBS curve) that necessitates four control points per span.

The sequence of knots is assumed to be non-decreasing, i.e. $t_i <= t_{i+1}$. Each successive pair of knots represents an interval $[t_i, t_{i+1}]$ for the parameter values to calculate a segment of a shape. Since the knot spacing may be non-uniform, the B-splines are not limited to resemble for each interval $[t_i, t_{i+1}]$ due to the fact that the degree can change. However, we recommend to use a uniform knot spacing in this software as Maya® by default creates NURBS of this kind (cf. Chapter 4.2.3.2). Non-Uniform B-Splines must not implicitly have a correlation of knots to control points causing further calculations to be more difficult.

4.2.3.2 Edit Points versus Control Points in Maya®

Maya® provides a tool that enables the user to determine points where a NURBS has to run through. These points are called *"edit points"*. There are several ways how to deform a NURBS curve. The user is able to move edit points or control points that have been defined on creation of the curve (see Figure 14). Compare Chapter 4.2.3 to get a better grasp how control points affect the curve. If the user aims at deforming the curve by dragging points that lie on the curve, he is allowed to use edit points. Setting the positions of the edit points by MEL (cf. Chapter 3.1.2) is also possible. Nevertheless, Maya® does not provide an access on edit points using the C++ API (cf. Chapter 3.1.3). If the user drags an edit point, the control points are influenced as

well. Hence, if it is possible to control both at once the control points and the edit points the computation gets confusing.

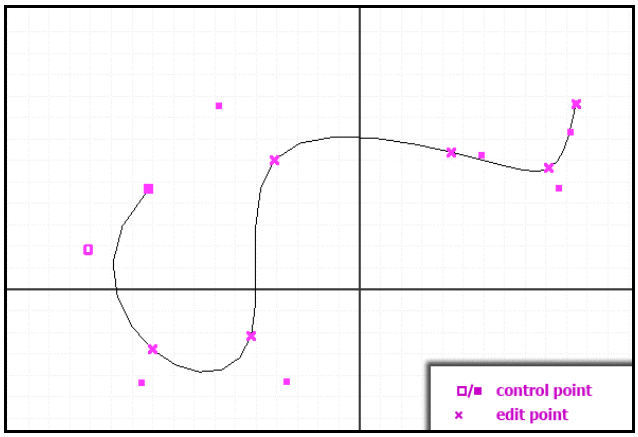


Figure 14: NURBS curve, control points / edit points

Regrettably, Alias solely provides a function to iterate over control points excluding edit points. We opened a support case at Alias to get information how control points are computed internally due to the condition having the required information about the edit points. However, they could not release additional information as they would free Alias property. We achieved them to open a suggest case for their developers to implement a setEP method included in the NURBS curve function set.

In Chapter 5.1.2, it will turn out why a NURBS curve is used in this software. Moreover, we will provide an effective technique to circumvent mixing the use of edit points and control points.

4.3 Facial Animation

One of the most challenging tasks in computer graphics is the construction and animation of authentic human facial models. Traditionally, facial models have tediously been animated by controlled facial mesh deformations such as the blend shape approach (cf. Chapter 4.3.1) or by kinematical approximation of muscle or bone actions as it is done with a facial rig (cf. Chapter 4.3.3). In the following chapters, we will expand on previous approaches and techniques that are nowadays still in use. We will compare the results and evaluate the effort caused to receive appropriate results.

4.3.1 Blend Shapes (Morphing)

Blend shapes successfully work on any NURBS (cf. Chapter 4.2.3) or polygonal geometry (cf. Chapter 4.2.2) morphing a source geometry to the shape of the target. In *"Lord of the Rings – The*

Return of the King", the computer animated character Gollum was animated using 950 blend shapes [SchNDI]. The face of this character was animated by morphing between those 950 blend shapes where each represents a facial expression.

Blend shape animation is a keyframe-based method that allows us to warp a source shape to a target shape. Based on a set of control entities such as vertices of a polygonal mesh, a source shape has to be created. A copy of the source shape serves for further development on the target shape. The fundamental condition to blend between two shapes is the same constitution concerning the amount of control entities. Assuming a polygonal shape serves as source and target shape, both must have an identical vertex count. Every vertex of the source shape is iterated through, and its position is determined. In the same manner, every vertex of the target shape with the same index as the vertex of the source shape, is iterated through. The position of the source shape's vertex and its corresponding target shape's vertex form a line. Interpolating between source and target vertices means that it is stepped along this line based on the size of intervals. This method is called linear interpolation (abbr. LERP) of the in-betweens. If the morph deformer is combined with weights per vertex this interpolation is computed differently for each vertex. If a vertex has a weight of 0 assigned, the deformer will not have any effect. Given a weight of 1, the vertex will be deformed from source to its target position. A weight of 0.5, for example, interpolates between the source vertex and the centre between the source and target vertex.

If several target shapes are used, the relative deformations of each vertex are added. Each vertex of different target shapes builds a vector with the vertex of the source shape. Depending on how much the single target shape contributes to the final deformation, these vectors are summed for each target shape and for every vertex of the source shape. Thus, several facial expressions or action units (cf. Chapter 4.1.1) can be combined.

There are systems that are content with interpolation of vertex positions. Such as real-time game engines that support lightning. The model data probably contains one normal information per vertex that has to be interpolated over time. In Maya®, the animator does not have to care about the computation of vertex-normals.

Due to the fact that computation of blend shape deformers consists of simple linear calculations, it produces best results concerning performance. The effort concerning time is proportional to the vertex count and the interpolation steps. Interpolating between vertices involves simple vector addition and multiplication. This is why it can easily be used in real-time systems.

Comparing the results concerning appearance, it must be admitted that the motion sequence does not look believable enough. Following closer investigation on real facial animation one will notice that it does not resemble linear interpolation between one and another expression con-

sidering single reference points. In Chapter 4.3.4, we will elaborate on the trajectories of characteristic feature points of the face.

The limitations mentioned above can be circumvented using blend shapes for in-betweens of the animation. However, creating a blend shape is a time-consuming task. Several techniques accelerate the production of facial expressions for one base shape. If the animator does not produce the polygonal mesh by using techniques such as box modelling (cf. Chapter 4.2.2.1) but by dragging every single vertex, this may lead to enormously time-consuming work.

At the research project *"Artificial Actor"* at the Filmakademie Baden-Württemberg a team of artists, computer scientists and us work on different systems to make facial animation in 3D computer graphics more credible. Another intention is to accelerate the process of producing facial animations. This includes finding a way to reduce the effort caused by creating blend shapes for different facial expressions. Therefore, we developed a tool that enables the animator to clone facial expressions. In the following chapter, we will comment on this tool in particular.

4.3.2 Expression Cloning Tool

The aim of the expression cloning tool is the partial reuse of already existing deformation information especially in the realms of facial animation. The tool allows cloning specific facial expressions on completely different target geometries. The tool is integrated as a plug-in in Maya®. The Siggraph 2001 paper *"Expression Cloning"* by Jun-yong Noh and Ulrich Neumann [NN01] serves as theoretical background.

To get a better grasp how the tool works, it will be explained in the following example: It takes a source and a target shape. Both may differ in shape and look but not in their constitution concerning holes in the polygonal mesh. For instance, given a head as source shape that has a mouth showing a hole, the head which serves as target shape must show the same hole (see Figure 15).

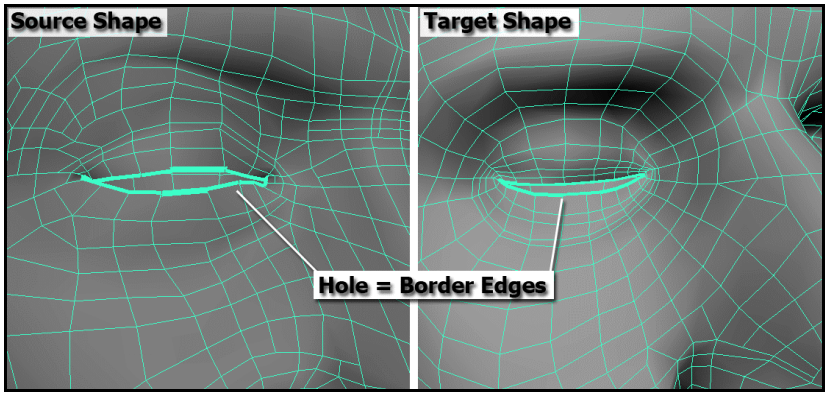


Figure 15: Corresponding holes

Next, the animator has to place locators on the source and the target geometry at characteristic places (anthropometric landmarks). If the animator puts a locator on the tip of the nose on the source geometry, he has to put a locator at the same place on the target shape. It finally requires facial expressions represented by blend shapes that originate from the source shape. Afterwards, the expression cloning tool can be applied. The source shape will be morphed and projected onto the target shape to receive correlations of the vertices of both shapes. In the following two chapters, we will go into detail concerning the morphing and projection algorithm as they both serve as foundation for another tool we developed to simplify the workflow of transferring the existing muscle-based system.

4.3.2.1 RBF Morpher (Thin Plate Spline Approach)

To morph the source to the target shape, it requires the feature point locators (anthropometric landmarks) of the source shape, the source shape itself, and the corresponding feature points locators of the target shape. The theory behind the thin plate spline RBF (radial basis function) morpher is to interpolate values between arbitrary feature points in arbitrary dimensions. To be more precise: RBF interpolation is basically an N-dimensional to M-dimensional function. Given a count of N input and N output samples, the positions of the shape counting M vertices are recalculated.

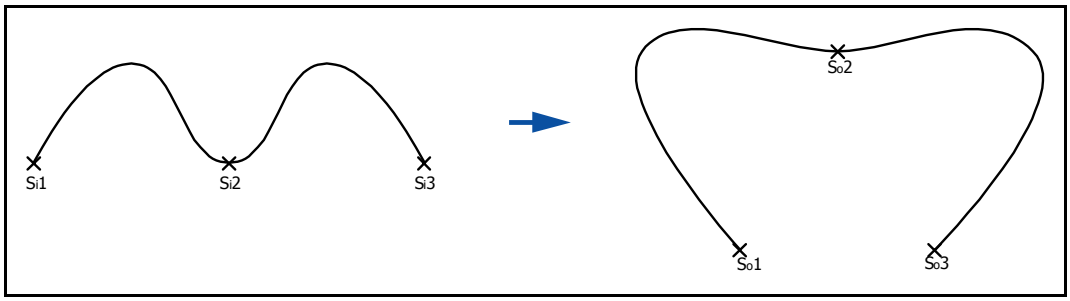


Figure 16: RBF morphing of shape with input feature points to output feature points

The thin plate spline RBF morpher is a point-valued function that contains the positions of the input and output feature points. With the information of the feature points, every point on the shape obtains a new position. This position is calculated depending on the weights every input and output feature point produces concerning its original position. Thus, the thin plate spline RBF Morpher smoothly interpolates between the feature points. Compare Figure 16 to see how the shape is transformed. It will be noticed that the original curvature between the feature points will stay the same. Furthermore, it has to be mentioned that a point of the shape that shares the same position of an input feature point will share the same position of the corresponding output feature point after being morphed.

To get a better grasp how this method works, we will explain the approach in three-dimensional space. The base function for the calculation is derived from [Ort96I]:

$$f(x,y) = a_o + a_1 x + a_2 y + a_3 z + \sum_{i=1}^{N} w_i U(|(x,y,z) - p_i|)$$

Equation 17: Thin plate spline base function

where p_i are the feature input points with a count of N.

$$U(r) = r^2 \log(r^2)$$

Equation 18: U-function

U is a function that represents the influence of a feature point on the evaluation point depending on the distance of both points. U has its function derived from the characteristics of a thin plate and tries to minimise the bending energy. Other approaches apply other U functions and produce similar results. That is because the logarithm is not necessarily fast to compute.

The weights w_i of these influence coefficients and (a_0, a_1, a_2), defining an additional inclined plane, influence the values of the function. Determining the weights w_i and the plane (a_0, a_1, a_2) is the major intention of defining the function based on the feature points.

To adapt the function, we implement the output feature points:

$$f(p_i) = q_i$$

Equation 19: Adaptation of function

In Equation 19, q_i denotes the output feature points. It follows that we have N equations that are linearly determined by the weights w_i and the plane (a_0, a_1, a_2). From the equations mentioned above and derived from the work of Taghvakish and Amini, it follows a system to determine the parameters [TA04]:

$$K = \begin{pmatrix} 0 & U(|p_1 - p_2|) & \cdots & U(|p_1 - p_N|) \\ U(|p_2 - p_1|) & 0 & \ddots & \vdots \\ \vdots & \ddots & 0 & U(|p_{N-1} - p_N|) \\ U(|p_N - p_1|) & \cdots & U(|p_N - p_{N-1}|) & 0 \end{pmatrix} \quad Q = \begin{pmatrix} 1 & x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & y_N & z_N \end{pmatrix}$$

Equation 20: Splitting up system of equations

If x or y have infinite limits, the system has to be defined so that the interpolated values of the function are determined by the base plane. Due to the base plane, four additional equations are attached. According to Taghvakish and Amini, it follows [TA04]:

$$\sum_{i=1}^{N} w_i = 0; \sum_{i=1}^{N} w_i x_i = 0; \sum_{i=1}^{N} w_i y_i = 0; \sum_{i=1}^{N} w_i z_i = 0$$

Equation 21: Equations to limit system

Combining the two matrices and Equation 21 leads to the following equation [TA04]:

$$\begin{pmatrix} K & Q \\ Q^T & 0 \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_N \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} q_1 \\ \vdots \\ q_N \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Equation 22: Final calculation of the weights and the base plane

With this equation, all parameters of the base function (cf. Equation 17) can be calculated. It might be conspicuous that it requires a linear system of equations to be solved to receive the essential parameters (cf. Chapter 4.2.1.5). If the weights w_i and the plane (a_0, a_1, a_2) are calculated, just insert them into the base function (cf. Equation 17).

Assume for now a source and a target shape such as a head geometry is given (see Figure 17, top). The source shape has to wrap up the target shape. Having set the input feature points (with the help of locators) on the source shape and the output feature points on the target shape on corresponding locations (see Figure 17, bottom), the thin plate spline RBF morpher can be executed. Applying the thin plate spline RBF morpher means that the source shape is morphed depending the input and output feature points. For instance, if input and output feature point are located on the tip of the corresponding nose, the source shape's tip of the nose will be placed on the target shape's tip of the nose after being morphed.

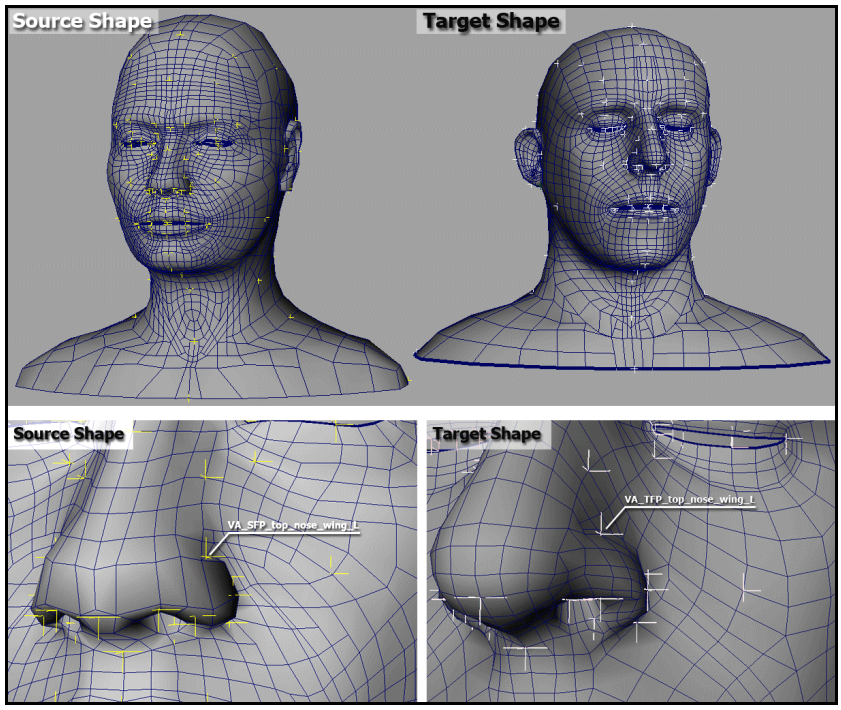


Figure 17: Top: source and target shape; bottom: corresponding feature points

It is important to know that after having the morpher applied, the shapes are not exactly the same. There is still a difference in topology and shape. To receive the source geometry with the same shape as the target geometry, the morphed source geometry is projected onto the target geometry (cf. Chapter 4.3.2.2).

4.3.2.2 Mesh Fitting

The mesh fitting algorithm serves to project the source shape onto the target shape so that both share the same shape but not the same topology. This means they do not share the same arrangement and count of vertices. The mesh fitting algorithm makes only sense after having the

thin plate spline RBF morpher applied. There are two ways of projecting the source to the target shape. The first method is denoted as *"intersect method"*.

The intersect method utilises the source shape and shoots a ray from each vertex into the direction of its normal. The new position of the vertex is represented by the position where this ray intersects with the target shape. This way, every vertex of the source shape lies seamlessly on the target shape.

The other method is called the *"closest point method"*. This method considers every vertex of the source shape and searches for its closest point on the target shape. This point allocates the vertex of the source shape. The experience proved that using the closest point method has to be preferred.

4.3.2.3 Generation and Amplification of Action Units

The source shape should have several equivalent blend shapes for different facial expressions. From now on, these blend shapes will be referred to as source blend shapes. Furthermore, we will distinguish between original source shape and fitted source shape. Such as the blend shape deformer, the motion vector from every vertex of the source shape to its corresponding vertex of the source blend shape is calculated. This motion vector is now translated to the space of its corresponding position on the target shape to create target blend shapes.

A local coordinate system has to be constructed for each vertex of the original and the fitted source shape. This local coordinate system is composed of the vertex normal that defines the x-axis, the first adjacent edge defines the z-axis, and the y-axis is defined by computing the perpendicular of the plane the x- and z- axis spans. By comparing the local coordinate systems of each vertex of the original and the fitted source shape, transformation matrices can be computed. After having the motion vectors determined, we can multiply these transformation matrices. Consequentially, the motion vectors are moved and rotated so that the resulting facial animation fits to the target shape that differs from the source shape concerning curvature and arrangement of facial parts.

If the source shape is fitted to the target shape, the vertices of the target shape are lying on the source shape but not on vertices of the source shape necessarily. They probably lie somewhere on a polygon of the target shape. It is important to say that these polygons are triangles. Recall Chapter 4.2.2.3, where a point lying inside a triangle is associated with the triangle coordinates. In the previous chapter we projected the source on the target shape. Now we do the same vice versa not to move the vertices but to receive the information about the relative positions of the target vertices depending the source shape. Due to this information, we can compute barycentric coordinates (cf. Chapter 4.2.2.3). Utilising these barycentric coordinates we can calculate the resulting motion vectors of the target shape. Hence, the motion vector of a vertex

of the target shape is calculated using its three surrounding points of the triangle of the source shape mesh. Each of these three surrounding points have one independent motion vector assigned and the barycentric coordinates serve as their coefficients finally adding up to compute the resulting motion vector of the target shape:

$$\vec{v}_r = b_0 * \vec{v}_0 + b_1 * \vec{v}_1 + b_2 * \vec{v}_2$$

$\vec{v}_r$: resulting motion vector of target shape

b_i : barycentric coordinates

$\vec{v}_i$: motion vectors of source shape

Equation 23: Calculation of motion vectors

Next, the resulting motion vector of every vertex of the target shape is added to its corresponding vertex. This way, the facial expression is translated and added on the target shape:

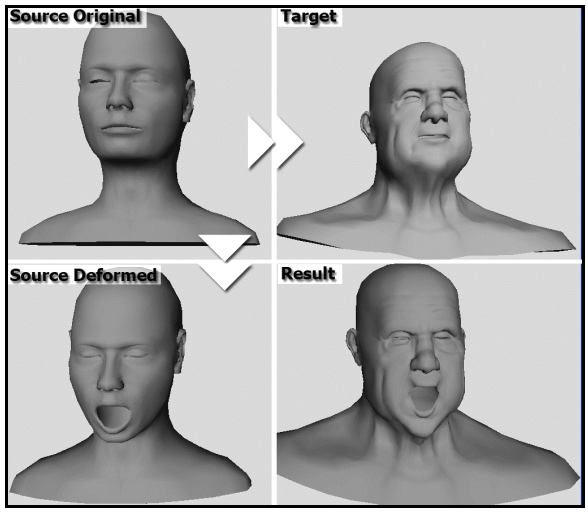


Figure 18: Workflow of expression cloning tool

Finally, the animator is able to adjust the generated deformations by amplifying or weakening the facial expression depending on the original target shape. The amplification is computed by scaling the resulting motion vectors as described above. This way, exaggerated or faint facial expressions can be created. Thus, if blend shapes exist that cover every area of the face, almost every realistic facial animation can be generated by combining and adjusting them.

4.3.3 Facial Rig

As described in Chapter 4.3.1, using blend shapes must not produce best results concerning authenticity of animation. Even mixing blend shapes could result in generating unbelievable faces if conflicting shapes are used. A facial rig is the easiest and most common approach to simulate muscles and bones of a head.

Comparing the results of the blend shapes method with the facial rig, it has to be assessed that using blend shapes produces better results concerning facial expressions but that generating convincing facial animation is a time-consuming task. Following the motion path of a muscle or a bone such as the mandible, it will be noticed that they do neither have a start nor an end point where they are moving linearly between. The motion path can be very curvaceous, and it is depending on many influences. Observing characteristic points of the face while moving, it will be noticed that they follow a special path and that not every point follows the same path. Approximating the motion path of every feature point of the face is the intention of a facial rig. In Chapter 4.3.3.1, we will describe how to use bones (joints) to set up a facial skeleton. In Chapter 4.3.3.2, we will introduce a method that describes how details can be put into the animation based on the facial skeleton. In Chapter 4.3.3.3, we will show a method to further improve generated facial expressions.

4.3.3.1 Face Skeleton

The advantage of using bones is that they are able to rotate about a joint, depending on a degree of freedom (DOF). Bones such as the mandible or the spine are easy to be synthesised using joints. If a facial feature point runs along an arc, bones should be preferred to draw the motion path. The animator even benefits from the fact that bones can be grouped and that they are subject to a hierarchical structure.

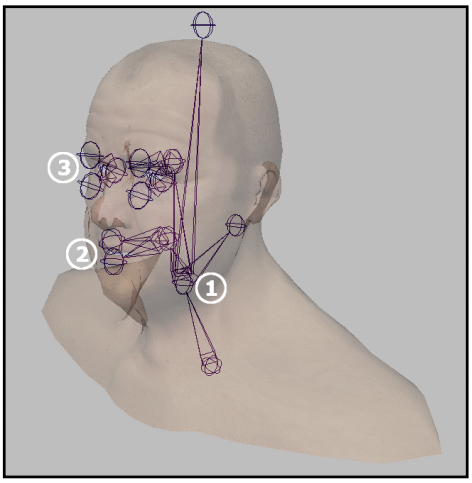


Figure 19: Face skeleton

How do the joints have to be set up? The first and most important joints that are placed are the head and neck joints that simulate the spine. They allow us to rotate the head (see Figure 19, (1)).

Next, to make the animation realistic, the animator binds the skin to the skeleton and determines the influence area of each joint. Maya® provides two methods to bind skin to a system of joints. The first is the so called *"rigid bind"* that provides *"articulated deformation effects by enabling joints to influence sets of deformable object points. […] With rigid skinning only one joint can influence each control vertex"* [Ali04a] If the animator intends to have smooth transitions between the influence

areas of different joints he can use *"flexors"* that are special deformers which provide round deformation effects inside the influence area of the joints.

With the smooth bind method, a control vertex can be manipulated by various joints. *"By default, their influence varies with distance, but you can edit or paint the skin point weighting on a joint-by-joint basis."* [Ali04a] Every control vertex has an accumulated weight of 1. The bigger the weight of one influence object, the more it is influenced concerning translation or rotation.

To guarantee the animation of the head to appear credible, the weights are painted after the skin is smooth bound to the skin. The first bottom joint should influence the entire shoulder and neck for instance.

If the spine is approximated, the jaw area can be rigged (see Figure 19, (2)). The DOF of the single joints are limited so that the animation does not appear exaggerated. By painting the weights, the animator can determine how strong the mouth area is influenced.

Finally, the eye joints can be placed (see Figure 19, (3)). The positioning of the eye joints has to be done in particular as the eye lids or the eye balls are treated separately. It has to be pointed out that these joints do not really belong to the skeleton as this area does not contain separate joints.

Detail in facial animation cannot be created using this rough setup of joints. The animator places additional joints or he can set up an additional system of cluster deformers (cf. Chapter 4.3.3.2).

4.3.3.2 Weighted Cluster Deformation

To gain more control over certain parts of a face, cluster deformers are used. A cluster deformer has a handle that can be translated or rotated and that influences a set of points depending on their assigned weights. It is important to mention that a joint internally works as a cluster deformer but that a cluster deformer may not have several influence objects; its influence area smoothly fades to adjacent influence areas. In fact, joints are used but as they do not work in a hierarchical structure, we will refer to them as cluster deformers.

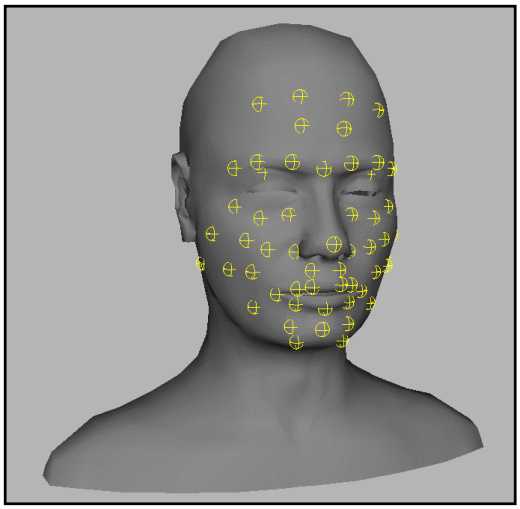


Figure 20: Weighted cluster deformation

They are positioned all over the face wherever an animation takes place. If these locations cannot intuitively be found, the FACS (cf. Chapter 4.1.1) can be consulted to understand which cluster deformers simulate which muscle and thus which facial expression. Next, the cluster

deformers are connected to each other and to additional control entities. Maya® provides several methods to create dependencies between cluster deformers or between cluster deformers and controllers. One method is the so called *"Set Driven Key"* method. Another way to bring two attributes into connection is to set *"Expressions"*: the animator connects attributes and creates dependencies by combining these connections with arithmetic functions. At this time, groups of cluster deformers can be built. If the animator intends the facial animation to be convincing, he has to guarantee the handles for the cluster deformers to move along a path. The animator draws the according trajectories and constrains the cluster deformer to them. Having a few cluster deformers constrained to a motion path, they can influence the remaining cluster deformers. This can be done by creating connections over *"Expressions"*.

Next, weights are distributed to every control vertex concerning each cluster deformer. This represents the fine adjustment of the facial animation and must be done intuitively. In Chapter 6.2.2.2, we will present a method how this process of painting weights can be tremendously accelerated.

Since using weighted cluster deformation to generate authentic facial animation is not sufficient for smooth skin deformations, resulting facial expression is corrected by adding blend shapes. In the following chapter, this approach will be discussed in-depth.

4.3.3.3 Corrective Blend Shapes

In fact, the corrective blend shapes do not differ from normal blend shapes. Using blend shapes on a rigged face is limited the way that they can only be applied on the original shape that means at the so called *"bind pose"*. Therefore, blend shapes are created for the appropriate expression, then, the corrected vertices are remapped to the bind pose shape.

At the research project *"Artificial Actors"* at the Filmakademie Baden-Württemberg, a team of artists and computer scientists worked on a tool that allows the animator to create a corrective blend shape while the original shape is showing a facial expression. This tool also remaps the blend shape to the bind pose and incorporates it into the facial rig system.

To remap the corrective blend shape, the transformations of every vertex concerning the joints (or cluster deformers) are conversely calculated. Considering the weights of every vertex and the transformations the joints cause, the bind pose shape can be determined. This calculation serves to define the corrective blend shape as it appears at bind pose. The corresponding blend shape is involved in the system so that it is driven in when the appropriate facial expression appears.

The whole setup previously described may generate convincing facial expressions. As setting up the system is a time-consuming task and controlling the animation is not intuitive enough,

decreasing the effort should be a major challenge. In the following chapter, we will present a setup that simplifies the process of creating realistic facial animation.

4.3.4 Adaptable Facial Setup

The *"Artificial Actors"* research project at the Filmakademie Baden-Württemberg focuses on finding new approaches to automate various aspects of character animation. Apart from improving the automation of animation, the major target is to increase the credibility of resulting facial animation.

"The innovation in the approach of the Adaptable Facial Setup consists of the unusual employment of facial motion capture data and its adaptation." [HBSL04] Facial motion capture data of single, isolated facial animations according FACS (cf. Chapter4.1.1) is assigned to a set of feature points that are located on the face. *"The tracked feature points contain nonlinear animation curves which in combination precisely describe the motion in all regions of the face during each Action Unit performance. The motion capture data is cleaned, normalized and separated from the global head movement"* [HBSL04] Regarding the physiognomic characteristics of a head, an animator is able to assure the facial animation to appear authentic by placing feature points. In this context, data adaptation means the animator can tune and adjust the deformation. Thus, the motion sequences can be amplified, weakened, or the animation can even be made unreal (see Figure 21).

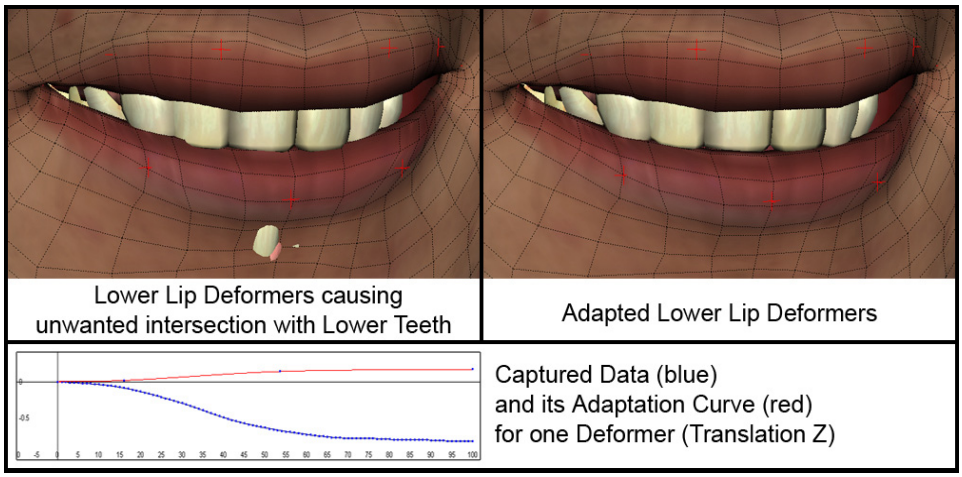


Figure 21: Data Adaptation for facial deformers [HBSL04]

The resulting animation curves represent the motion paths for the corresponding cluster deformers as described in Chapter 4.3.3.2. The weights are painted causing the face deform smoothly. Finally, compare Chapter 4.3.3.3 to get a better grasp how the generated expressions are fine-tuned.

Comparing the results of the adaptable facial setup with the results of the techniques previously discussed, it must be concluded that this setup is fast to construct and that it produces good looking results. It provides the best composition of an easy-to-handle setup for animators and facial animation close to reality.

There exist other approaches to animate a face. One approach that only covers details in facial animation is the usage of displacement maps. A displacement map lets the animator use an image to specify surface on a geometry. Chadwick developed another approach that uses free-form deformations (cf. Chapter 4.2.2.2) to shape the skin in a multi-layer construction containing bones, muscles, and skin. There are several more techniques that, as a matter of fact, are not as commonly used as the approaches previously introduced.

4.3.5 Pro Muscle-Based Systems

To understand the motion sequence referring to points of a face, scientists such as biologists or even artists always attempted to understand the relationship between exterior shape and the structure creating it. To synthesise anatomically caused behaviours, this bottom-up approach is mandatory. If an artist paints a human figure, he has to know about its physiques. Animating a character is depending on complex biological processes that differ from human to human or from human to animal. Gender and age are some important aspects of differing biological constitution of humanoid characters. It has to be stated that until now, the biological complexity could not completely be transferred into computer graphics. Hence, using a muscle-based system is another step towards the simulation of anatomy close to reality.

The results of facial animation are not convincing enough yet. Commonly used approaches for facial animation we mentioned in the previous chapters lack naturalness. They claim not to be anatomically correct. Furthermore, they do not allow us to use the results obtained by simulating real anatomy. The computation of wrinkles and bulges is a good example. Wrinkles result from the skin being compressed and depending on its constitution and on the underlying structures such as the muscles that deform and move in a certain direction.

The constitution of the layers, connecting the surface of the skin to a muscle, has so many properties as described in Chapter 2. These properties are responsible for a special behaviour of the surface, when contracting a muscle that cannot be captured by just observing the superficial behaviours of a body.

In addition to previously mentioned reasons, the muscle-based system enables to bridge to other research activities. Plastic surgery is only one to name. Simulating muscles may help planning medicinal interventions (cf. Chapter 6.2.3).

Finally, as our approach is based on the FACS (cf. Chapter 4.1.1) which categorises facial expressions depending on muscles, we synthesise muscles to create reasonable and believable facial animation.

4.4 Muscle Models

4.4.1 History

Every endeavour has been made to construct expedient muscle systems. History proves that research activities increase that are committed to realistic simulation of anatomy. In the 19th century, the French neurologist Duchenne was the first who described contracting muscle fibres by applying electrical currents. He was the first who was able to create artificial facial expression on living humans with the help of electricity [Kat03I]. The next milestone in the history of muscle-based systems is the work of Ekman [Ekm78] (cf. Chapter 4.1.1). Nowadays, the FACS still is the most commonly used foundation for 3D facial animation. In 1974, Parke [Par74] introduced the first *"Parameterized Model for Facial Animation"*. Parke's approach introduces structural understanding to facial animation that is supplemented by parameters based on observation. At the beginning of the 80's, Platt and Badler [PB81] presented their work of *"Physically Based Muscle-Controlled Face Model"*. They propose an abstract structure of muscle fibres with skin, muscle and bone nodes which are connected by springs. In 1987, K. Waters [Wat87] presented *"A Muscle Model for Animating Three-Dimensional Facial Expression"* that, in contrast to the model of Platt and Badler, does not regard the skin and muscle as a separate layer.

After Waters had introduced his vector muscle model, research committed to muscle-based models reached a new level of attention. Many researchers followed Waters' approach (cf. Chapter 4.4.2), others proposed new methods.

In the following chapters, we will introduce the models that we think are milestones in terms of muscle-based animation and that have an immediate connection to our approach. As many models share common ideas, we will categorise the proposed methods.

4.4.2 Single Layered Mesh Model

In this section we describe methods for facial animation that are based on the properties of the skin, the muscles, and the bones. In contrast to other models, the following approach has only one layer, the facial skin layer which is manipulated. The effect of muscle motion is simulated directly by the skin neither driven by an underlying shape of a muscle nor by the skeleton.

Waters is one of the pioneers in the history of anatomically-based 3D computer animation. His paper about *"A Muscle Model for Animating Three-Dimensional Facial Expressions"* [Wat87] provides the development of a parameterised facial muscle process. He is the first who breaks down the connection of bones, muscles, and skin into one comprising layer. In his opinion, existing

facsimile of anatomy does not allow a rich variety of facial expressions. He aims at making facial expression controllable by a limited number of parameters: *"What is required is not the exact simulation of neurons, muscles and joints, but a model with a few dynamic parameters that emulate the primary characteristics. These methods are relatively abstract, and do not attempt to model the biomechanical or neurophysiological mechanisms."* [Wat87] He describes the skin as a mesh of nodes that forces are applied on. The forces on every node are depending on the tear resistance of the skin, the muscle strength, the distance from muscle to node and from muscle to bone, the squeezing of muscles, and the skin thickness.

He states that only a proportion of the muscle's force affects the skin. This proportion is calculated as follows:

$$prop_{lc} = l_m * \cos(\alpha_{ms})$$

$prop_{lc}$: proportion line of contraction on surface
l_m : length of muscle fibre
α_{ms} : angle of attachment muscle to surface tissue

Equation 24: line of contraction on surface

In his model, Waters describes the surface deformation in correlation to a muscle. Feature points on the skin surface have a maximum and a minimum influence area and in this area differing intensities of displacement.

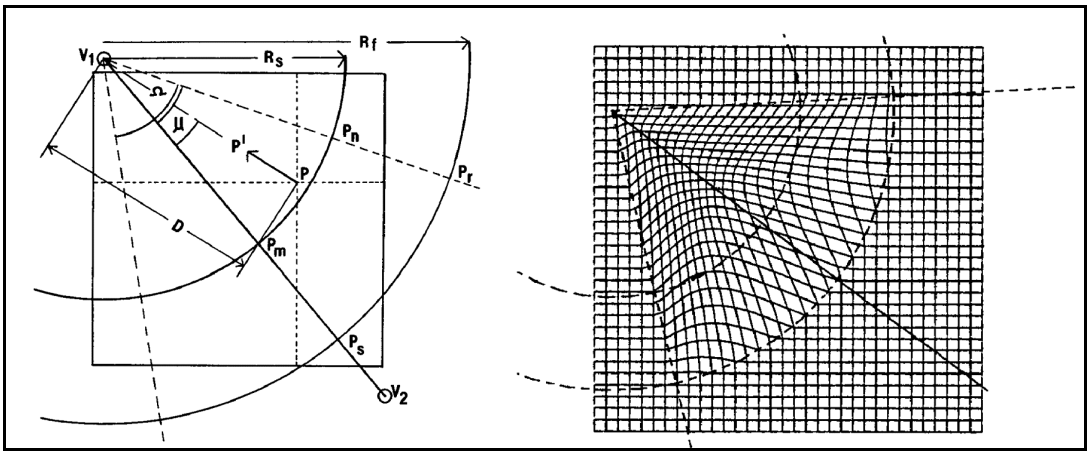


Figure 22: Muscle vector model [Wat87]

The left sketch of Figure 22 is showing the displacement of the point P in 2D space. P is displaced towards V_1 along the vector PV_1 by the factor K, A and R.

$$A = \cos\left(\frac{\mu}{\pi} * \frac{\pi}{2}\right) \quad (1)$$

$$R = \cos\left(\frac{D\text{-}R_s}{R_f - R_s} * \frac{\pi}{2}\right) \quad (2)$$

$$P' = V_1 + K*A*R*\overrightarrow{(PV_1)}_{\text{norm}} \quad (3)$$

Equation 25: Displacement of P

See Figure 22 to read the definitions of the parameters determining the displacement. K is the muscle spring constant that has to be predefined. R_s represents the radius to start, R_f the radius to finish the falloff, both as a distance from V_1. D denotes the distance from P to V_1. μ is the angle between V_1V_2 and V_1P. V_1 and V_2 denote the start and end point of the muscle vector. Modifying K causes different contraction appearances in the same area. The calculation represented in 2D space can easily be transferred into 3D space by adding the third dimension: the z-value. To simulate visco-elastic behaviours, Waters creates ramps that are defined to control the gradient of the contraction in the area specified by μ.

Waters describes the sphincter muscle, in contrast to the parallel muscle, as a muscle with one reference point. The feature points of the skin are drawn together towards the reference point that defines the centre of contraction. In 3D space, this would mean that in a spherical area the feature points are displaced towards the centre point. This spherical area can be made ellipsoidal by defining different magnitudes in the direction of the x-, y- and z-axis. The factor that finally determines the contraction is the spring constant K.

Waters prefers polygonal data structures as they are simply to use. He suggests avoiding polygonal intersections by increasing the density of the polygonal mesh at curvaceous areas. For curvaceous muscles he suggests to create muscle vectors that curve around an underlying structure. Finally, he recommends to use groups of muscles to express facial actions as that may take place on a basis of a linked structure.

Bui et al. [BHPN03] propose a method that extends Waters' linear muscle system in terms of portability of the muscle vector data. He augments Waters' model concerning multiple muscle interaction and the production of wrinkles.

Bui et al. [BHPN03] use the RBF algorithm (cf. Chapter 4.3.2.1) to transfer the existing muscle data from a source to a target face. The RBF algorithm he applies is not the same but produces similar results as the approach previously described (cf. Chapter 4.3.2.1). In contrast to

the expression cloning tool (cf. Chapter 4.3.2), Bui et al. only clone muscle data for each major emotion: happiness, sadness, surprise, fear, anger, and disgust:

```
For expression E=Happiness, Surprise,…
 RBF_mapping(M_source, L_source, L_target)->M_target
 Repeat(Interactive Genetic Algorithm)
  Show different versions of expression E
  The user chooses some "favoured" faces
  The system calculates new faces
 Until the user satisfies with the faces
 Add head and tail of muscles expressing E to L_source and L_target
```

Source Code 1: Pseudo code for muscle data cloning

Source Code 1 describes the steps to transfer the muscles. L_{source} denotes the source and L_{target} the target feature points (cf. Chapter 4.3.2.1, feature points). M_{source} denotes the data of the muscles, especially the vertices that describe them. M_{source} represents the muscle data of the source, M_{target} the muscle data of the target face. M_{target} is finally computed.

Bui et al. [BHPN03] use an interactive genetic algorithm that allows us to easily verify the remaining parameters that define the process of muscle contraction. They presume that a count of approximately ten parameters define one muscle contraction. The interactive genetic algorithm is a search algorithm that is based on the rating of a user. These ratings are used to generate further possible solutions. With the definition of a certain range of one parameter, the algorithm generates a number of possible solutions. This means the user has a choice of several variations of one emotional expression. Having several variations means that the parameters may change in their defined range. The user picks several versions of the generated solutions. With this selection, a new generation of versions of muscles is generated. The process continues until a satisfying version is found. This is repeated for every major expression described in Source Code 1.

Finally, Bui et al. [BHPN03] transfer region divisions for faster computation of the muscle contractions. Assuming a polygonal mesh serves as face shape, each vertex has to be considered when computing the muscle contraction. Dividing the face shape into regions enables the exclusion of unaffected vertices. This provokes the computation speed to increase. In Chapter 6.2.2.2, we present a method that works more detailed but has the same effect as the method of Bui et al. [BHPN03].

Bui [Bui04] presents an approach to generate bulges and wrinkles. He affirms that *"Pseudo-muscles or parameterization approaches usually fail to create wrinkles because they ignore the underlying anatomy of the face"* [Bui04]. He differentiates between bumpmapping techniques and physically-based

models such as defining wrinkle functions. Bui proposes a technique that extends Waters' muscle model:

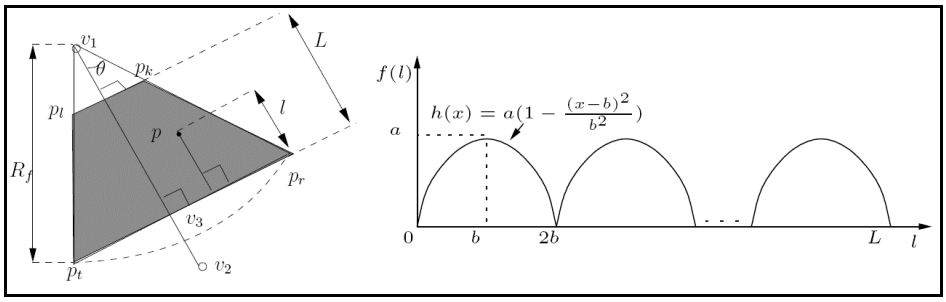


Figure 23: Zone of wrinkles and the wrinkle function [Bui04]

All the vertices in the $p_lp_kp_rp_t$ area are included in the calculation of the wrinkle amplitude (see Figure 23, left).

$$L = \frac{3}{4}|v_1v_3| = \frac{3}{4}R_f \cos\theta \quad (1)$$

$$u(l) = l - \left\lfloor \frac{l}{b} \right\rfloor b \quad (2)$$

$$b = \frac{L}{2N_w} \quad (3)$$

Equation 26: Wrinkle functions

Equation 26, (1) defines the distance from p_lp_k to p_tp_r. The wrinkle amplitude at a vertex p is a function of the distance l from p to p_tp_r. The distance l is periodically mapped to [0,2b] with a frequency of N_w (cf. Equation 26 (2)). Finally, b is calculated as shown in Equation 26 (3). It has to be mentioned that $\lfloor ... \rfloor$ rounds down a real number to an integer number.

Figure 23, right shows the shape of the wrinkle function which is formulated as follows:

$$f(l) = a\left(1 - \frac{(u(l)-b)^2}{b^2}\right)$$

Equation 27: Function describing series of parabolas

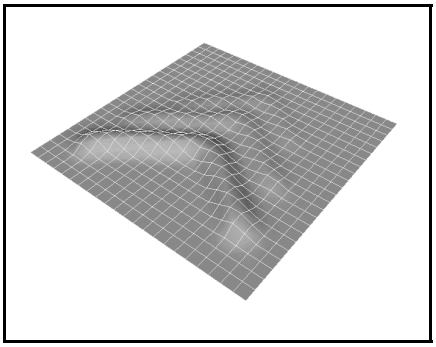


Figure 24: Wrinkles as result of the muscle contraction

In Equation 27, 'a' denotes the amplitude of the wrinkles. *"The wrinkle amplitudes are applied to the direction of the normal of the vertices after the vertices are displaced by the muscle contractions. For vertices that are inside the zone of influence of multiple muscles, only the maximum wrinkle amplitude caused by these muscle is used."* [Bui04] Bui remarks that the parabola function is fast to compute.

Breton et al. [BBP01] propose to enlarge approaches such as Waters' model by treating difficult areas separately: *"Muscles cannot perform the animation of particular parts of the face[…], that is to say: eye, eyelids, jaw and neck.[…] the lips also need to be identified."* [BBP01]

Eye blinking that contains rotating eyes and shutting eyelids is independent from facial expression. Breton et al. [BBP01] claim the eyeballs and eyelids to be pieces of a sphere. They affirm that eyes and eyelids are rotated around the centre and the axis of a sphere. The centre, the axis, and the diameter of this sphere receive their dimensions from the two vertices of the eye that are most distant to each other. They trigger the eye blinking on an arbitrary timing basis but based on biological influence characteristics as well.

Given the jaw or the neck, they apply another approach. They define three vertices that describe the location of the jaw: v_{neck}, v_{pivot}, v_{chin}. These three vertices form an angle $\angle(v_{neck}, v_{pivot}, v_{chin})$. In the end, all vertices including the area $(v_{neck}, v_{pivot}, v_{chin})$, are rotated around v_{pivot}. Breton et al. [BBP01] point out that maintaining a smooth transition from jaw to neck, the rotation has to fade for the vertices close to the v_{pivot}, v_{neck} segment to generate a kind of swell.

The neck is defined by four vertices that limit the neck: one at the top front, one at the top back, the bottom front and the bottom back. All vertices between the segment v_{top1}, v_{top2} and the segment $v_{bottom1}$, $v_{bottom2}$ define the neck, all above v_{top1}, v_{top2} define the head. The centre of rotation of the head is defined by the centre of the bounding box that obtains its dimensions from v_{top1}, v_{top2}, $v_{bottom1}$, $v_{bottom2}$. Finally, the rotation fades from the top of the neck maximally rotated to the bottom of the neck minimally rotated.

In the lips' case, Breton et al. [BBP01] provide an algorithm that computes a distinction of the upper and the lower lip. The user has to determine the boundary points of the lip shape. Via several boundary vertices, they generate a distinction between the upper and lower lip even being the same mesh. Nevertheless, the lips are animated using the sphincter muscle model of Waters [Wat87].

Zhang et al. [ZST04] introduce a technique that enables the adaptation of a generic control model of a face depending on the geometry of a specific person's face. Zhang et al. [ZST04] do not explicitly claim to extend Waters' model. However, their method contains the adaptation of muscle data that is similar to the muscle model Waters used to describe muscle contraction and the skin deformation the muscle produces. The adaptation algorithm is based on a 2D image of

the generic and individual face model. This 2D image represents the 'projected' image as it appears on screen. Next, anthropometric landmarks are determined on the 2D image. Zhang et al. [ZST04] propose a technique that computes 3D positions of the landmarks by a projection-mapping approach. Due to the required information, the adaptation is computed. Similar to Waters' model, he describes a muscle having an attachment and an insertion point. These points are transferred to the individual target face and thus the individual face is made controllable. It has to be indicated that the approach, Zhang et al. [ZST04] use for the adaptation of the source muscle data, is similar to an approach using the RBF morpher (cf. Chapter 4.3.2.1).

Finally, Breton et al. [BBP01] address an important issue that encourages us not to use the same model described by Waters [Wat87]. *"The displacement vectors do not take curvature into account and displacements are carried out according to plan. This does not usually pose a serious problem, as most of the time, the muscles are applied in fairly flat areas such as cheeks or forehead."* [BBP01] We do not agree that this fact may be remained out of consideration. To receive more detail in realistic facial animation, the curvature is implicitly considered. The contraction of the skin is not depending on a vector between two vertices that indicate a parallel muscle. The model of Waters does just as little go deeper into the issue of volume preservation of muscles and the bulge a muscle can produce.

In the following chapters, we concern models that simulate anatomy with real muscle shapes. The following approaches do not handle direct skin manipulation without considering the properties of a muscle. This means that these approaches will not only have a single layer to produce facial animation driven by the motion of a muscle. These models include a separate skin, muscle, and skeleton layer.

4.4.3 Deformed Cylinder Model

Wilhelms and Gelder [WG97] suggest to use triangle meshes to model muscles, bones, and generalised tissue. For the sake of speed, they consider the muscle to be modelled as a *"deformable discretised cylinder"* [WG97]. Their model allows us to change the shape of the muscles by moving joints. The skin is generated by *"voxeling the underlying components, filtering, and extracting a polygonal isosurface"* [WG97].

Wilhelms and Gelder take a look at body muscles that have two attachment points. Both origin and insertion (cf. Chapter 2.3), can be fixed to a bone. In the case of facial muscles, the insertion can also be attached to tissue. Moreover, Wilhelms and Gelder [WG97] claim the diameter and the shape of the muscle to change depending on the relative positions of origin and insertion.

In their work, they describe a muscle as a discretised cylinder that is running along a path (action line) starting at the midpoint of the origin and ending at the midpoint of the insertion (see Figure 25).

The cylinder is divided into 7 sections that are separated by 8 elliptical cross sections. The cross sections form an elliptical shape. These ellipses have their centre lying on the curve path previously described and depicted in Figure 25, bottom. Each ellipse is discretised on a basis of intervals and in the same succession as the radial points of the shape. The polygonal mesh is finally composed of the elliptical shapes' radial points connected to neighbouring cross sections. Each cross section lies on a plane that has its orientation obtained from a local x, y, z coordinate system. The origin of the local coordinate system lies in the centre of the cross section. The z-axis is pointing towards the curve path that deforms the cylindrical muscle. The x-axis stands perpendicular to the z-axis and points away from the bone. The y-axis is perpendicular to the x- and z-axis. The x- and z-axis also define the dimension of the elliptical cross section. To imitate the volume preservation of the contracted muscle, the dimensions of the y-axes are manipulated along a curved path from origin to insertion so that at the centre of the muscle the most significant bulge appears.

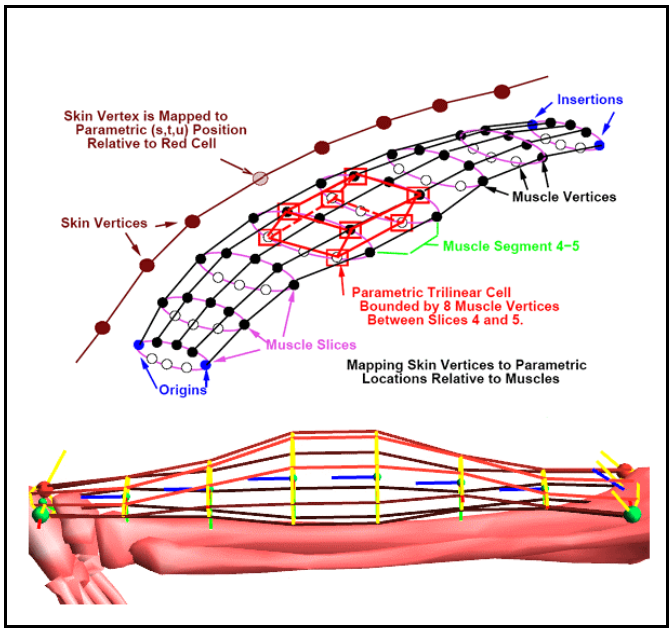


Figure 25: Muscle as discretised, deformed cylinder model [WG97]

The approach of Wilhelms and Gelder allows us to further adjust the muscle geometry. One component that facilitates to modify the geometry is a pivot that can be placed. The curve path of the deformed cylinder is then running through this pivot with the insertion and origin position at its ends. Furthermore, this pivot is assigned to one slice. The model enables to control

each slice, its dimensions in particular. The parameters that can be set are the height and width of each slice, the x- and y-value, the location in x-, y- and z-coordinates, and the orientation of the slice. This implies that each slice can additionally be rotated. Wilhelms and Gelder affirm that nearly all muscles can be simulated with this model [WG97].

The muscle animation is controlled by the motion of the skeleton joints. The joints control the origin, the insertion, and the pivot of the deformed cylinder. The dimension of every slice is recalculated when the muscle contracts to simulate volume preservation. Wilhelms and Gelder make a distinction between the rest and the present length whereas the rest length denotes the length of the curve path in rest position [WG97]. Present length denotes the length of the curve at present. This length changes during animation where the curve path deforms. The slices are scaled in x- and y-direction by the factor $\sqrt{restlength / presentlength}$. Regrettably, Wilhelms and Gelder [WG97] do not source the following statement: *"In any case, exact volume preservation of muscles is not biologically justified."* [WG97].

The attachment of the muscles to the skin and its consequential skin deformation that Wilhelms and Gelder [WG97] provide in their proposal is described in Chapter 4.5.1.

In our work, we also follow their approach to a certain degree and experiment with some of the techniques Wilhelms and Gelder propose. In Chapter 5, we will explain why we decided to omit major approaches provided by Wilhelms and Gelder.

4.4.4 Ellipsoid Model

Our experience revealed that so far, the use of ellipsoids is the most commonly used technique to synthesise muscles. It represents an simply to use method that claims to generate realistic simulations and a fast computation. In the movie *"Hellboy"*, this approach is used as basis for muscle-based animation.

Wilhelms [Wil94] was one of the first to present a muscle model based on the use of ellipsoids. The following equation shows the specification of an ellipsoid:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1\,;V=\frac{4\pi abc}{3}$$

Equation 28: Ellipsoid

Based on this equation (Equation 28, left), it can be easily identified if a particular point is inside or outside the ellipsoid. a, b, c denote the major axes. The model enables applying geometric

transformations on the ellipsoid such as translation, scaling and rotation. In Wilhelms' model, a muscle is composed of three ellipsoids: one for the muscle belly and two for tendons (see Figure 26, left).

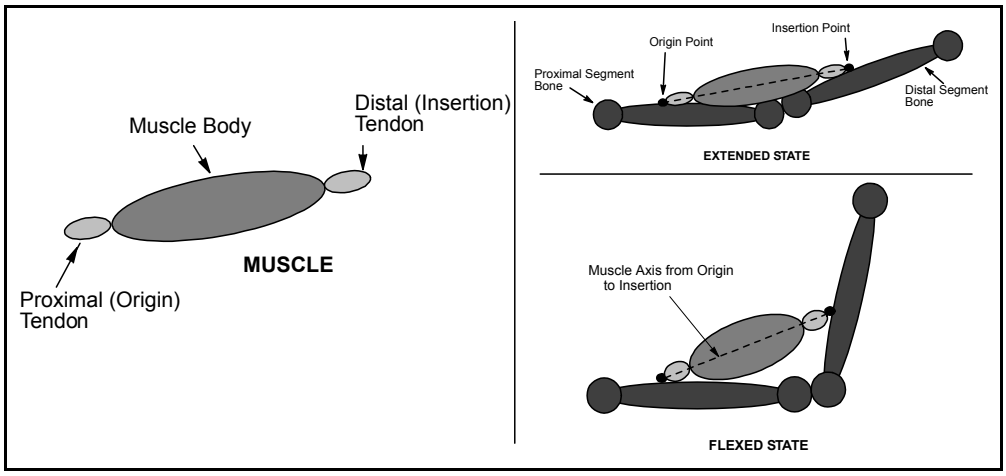


Figure 26: Muscle and bones in flexed and extended state [Wil94]

Wilhelms uses the term *"muscle coordinate frame"* which signifies the local coordinate system of the muscle ellipsoid. This coordinate frame includes all geometric properties of an ellipsoid. The muscle coordinate frame arises from the origin point of the muscle, and its z-axis points towards the insertion. These reference points stay fixed to the bone while animated. Thus, when the joints move, the muscle is still spanned between the origin and the insertion. It has to be mentioned that each of the tendons makes up 20 % of the distance from insertion to origin whereas the muscle body makes up 60 %. The width of each ellipsoid has a portion of 40 % of the entire muscle length. This represents a secure method of constant volume preservation. Since the muscle grows larger towards its centre, it does not start at the bone segment but begins and ends slightly shifted from the fixed attachment points.

The muscle bulge is computed as follows: the muscle length is stored by the time it is created and attached (l_{rest}). If the joints move, the current length of the muscle changes ($l_{present}$). The new scale dimensions are received by relating to the factor $l_{present}/l_{rest}$. If V is the ellipsoid volume and r is a/b in rest position, it comes to following conclusion:

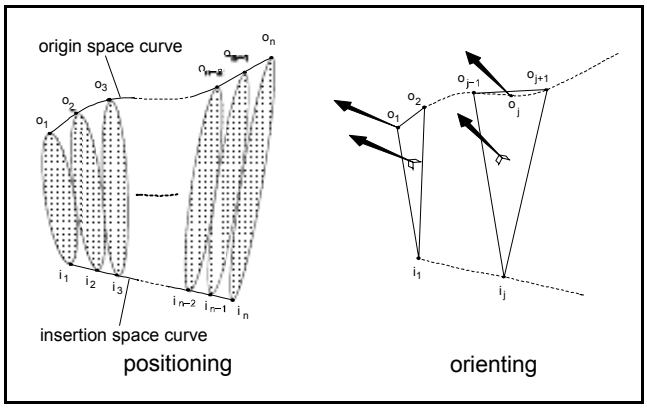
$$c_{present} = \frac{l_{present}}{l_{rest}} \quad (1) \quad ; \quad b_{present} = \sqrt{\frac{3V}{4c_{present} r\pi}} \quad (2) \quad ; \quad a_{present} = b_{present} r \quad (3)$$

Equation 29: New ellipsoid dimensions

Finally, the ellipsoid obtains its orientation by matching the z-axis with the vector defined by origin and insertion.

Scheepers et al. [SPCM97] extend Wilhelms model [Wil94]. They modify Wilhelms approach by obviating the use of three ellipsoids describing a fusiform muscle (cf. Chapter 2.3). Tendons are not placed along one shared axis. Each of the tendons has a unique start and end point. The start points are attached to the bones defining the insertion and origin location. The end points define the start and end point of the muscle axis. In contrast to Wilhelms model, they maintain the length of the muscles while animated. As a matter of fact, the tendons remain stiff in lifelike anatomy.

Scheepers et al. [SPCM97] propose an approach to simulate muscles with more complex shapes: the so called *"multi-belly muscle model"*. Neither the insertion nor the origin are represented by a point. Both are defined by a spline curve (cf. Chapter 4.2.3). The muscle bellies with a count of n receive their two attachment points by determining a position on the two curves (see Figure 27, left). To compute the orientation of the muscle bellies, an up-vector is required. This up-vector is defined by the normal vector of the plane spanned by three sample points described in Equation 30 and Figure 27, right. The number n of bellies is determined. Next, the bellies are attached to the insertion and origin curve by sampling the curves in certain intervals defined by the count of bellies and the curve length.



$$o_j, o_{j+1}, i_j \quad \text{if} \quad j = 1;$$
$$o_{j-1}, o_{j+1}, i_j \quad \text{if} \quad 1 < j < n; \quad \text{and}$$
$$o_{j-1}, o_j, i_j \quad \text{if} \quad j = n;$$

Equation 30: Sample points

Figure 27: Location and orienting of muscle bellies [SPCM97]

Since this approach offers satisfying results in terms of authenticity and a rapid computation, this technique can compete with all other approaches. Ellipsoids are primitives which are fast and easily to generate and deform. Therefore, the interactive creation of character sets is enabled. This strategy convinces many people interested in 3D character animation.

Miller and Thuriot developed a tool named *"Maya Techniques™ | Custom Character Toolkit"*. This tool is based on the approach of Scheepers et al. [SPCM97]. In addition to the approach of

Scheepers et al. [SPCM97], they provide the methodologies and techniques involved in building a character pipeline. As previously indicated, Thuriot already applied these techniques for the muscle-based animation of *"Hellboy"*.

The technique introduced by Scheepers et al. [SPCM97] also serves as the foundation for the cgmuscle-project [SE04I]. The members of this community-based project address themselves to the creation of an open source muscle-based system.

4.4.5 Mass-Spring Model

Muscle deformations can be simulated by using one of the following approaches: biomechanical models, physically-based models as well as geometric models [NT98]. Nedel and Thalmann [NT98] work on a physically-based model. They describe the forces a muscle produces along lines of action (cf. Chapter 2.3.7) that all run through the origin and insertion of the muscle. Each line can be embodied by a polyline that composes the muscle model having the shape of the muscle associated. Nedel and Thalmann [NT98] use an approach that is based on the usage of mass points linked by springs. Every single point has two horizontal and two vertical neighbours in a polygonal mesh. The action line serves as the up-reference of each point.

Given a scarce geometry of a fusiform muscle (cf. Chapter 2.3), they propose a technique to automatically resample the muscle shape. Based on the number of slices and the count of points referring to each slice, the algorithm serves to evaluate the entire action line. The algorithm is responsible for drawing an imaginary circle around every appropriate position. Between the action line and the slice, imaginary lines are drawn. Each line and the initial muscle shape intersect. The resulting intersection represents the new muscle point. This allows us to reconstruct muscle models obtained by 3D scans.

3D scans can be received by the National Library of Medicine. This library originates *"The Visible Human Project®"* [VHP86I] that is concerned with the creation of complete, anatomically detailed, three-dimensional representations of the usual male and female human bodies. They sectioned a human cadaver (see Figure 28) and created a digital image dataset of the body. They then divided the body into a quantity of voxels so every point of the body became discretised. Every voxel carries distinct information, for example, which part of the body it belongs to. Vice-versa, by determining a part of the body, a 3D model can be generated. This model lacks smoothness, causing the model of Nedel and Thalmann

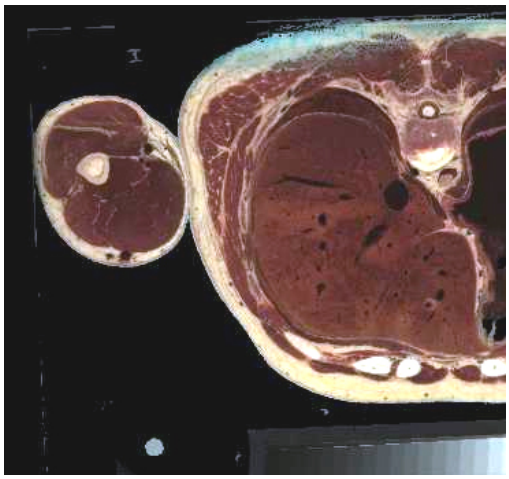


Figure 28: Cross section of a human body[VHP86I]

[NT98] to be difficult to apply. That is the reason why they provide a technique to resample the muscle shape.

Nedel and Thalmann [NT98] describe the deformation of a muscle due to mechanical laws of particles. The model incorporates parameters that describe the nature of the particle. The positions of adjacent particles influence the particle. To simplify calculation processes, they only take the surface of the muscle shape into consideration and exclude volume characteristics. The surface mesh is composed of points. Each point corresponds with a particle in the physical model. The set of particles has a mass density of m. The force on one particle consists of three forces that add up to one:

$$f_{result}(x_i) = f_{elasticity}(x_i) + f_{curvature}(x_i) + f_{constraints}(x_i)$$

Equation 31: Resultant force on each particle x

In their model Nedel and Thalmann use vectors to represent forces.

Elasticity is simulated by linking each particle to its four neighbours by springs (see Figure 29). The force of one so called Hook's [NT98] spring is calculated as shown in Equation 32 (1).

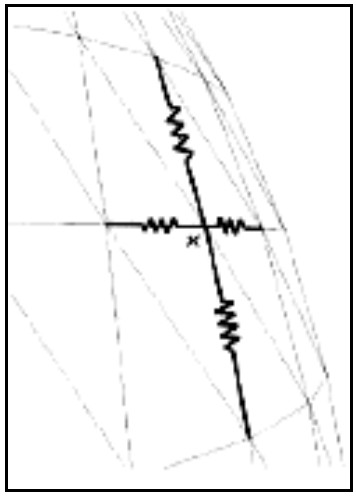


Figure 29: Elastic model of the muscle surface[NT98]

$$f_{spring}(x_i) = -ks(x_i - x_{rest}) \quad (1)$$

$$f_{elasticity}(x_i) = \sum_{j=0}^{3} f_{spring_j}(x_i) \quad (2)$$

Equation 32: Elasticity force

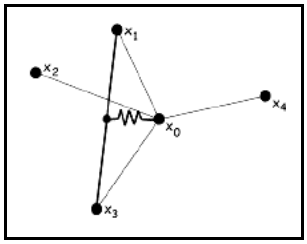


Figure 30: Angular spring[NT98]

The coefficient of the degree of a spring's elasticity is referred to by ks. *"x_i is the spring's oscillating extremity and x_{rest} is the rest position of this extremity."* [NT98] The elasticity force on one particle is then calculated by adding the forces of all springs connected to its four neighbours. We use the terms horizontal or vertical, width or height, to refer to the orientation of the action line.

The curvature or torsion force determines the bending and twisting of a muscle surface. Therefore, Nedel and Thalmann [NT98] create another kind of linear spring: the so called *"angular spring"*. Given an angular spring, each particle x_0 has four neighbours x_1, x_2, x_3, x_4. The particle x_0 has two angular springs. The first one with the angle defined by the vectors x_0x_1 and x_0x_3 (see Figure 30), and the second one with the angle between x_0x_2 and x_0x_4. See Figure 30, where an angular spring is spanned from x_0 to the midpoint of the line between x_1x_3 and from x_0 to the midpoint of the line between x_2x_4. The springs are set in rest position. Nedel and Thalmann denote the two generated vectors as vertical and horizontal relating to the action line representing the up-vector [NT98]. Including these angular springs into the spring network causes the shape to be preserved. They claim to be capable of controlling the muscle volume by changing the angular spring coefficients. Their model provides no volume preservation, but it enables to control the degree of deformation with the preservation of shape integrity and homogeneity.

The geometric constraint forces are the third kind of forces acting on the muscle shape. These constraints are used to control the muscle by external forces. Only some selected points are constrained this way. Two points, the insertion and the origin, are attached to bones or to the skin. Other points can be influenced by another internal structure such as organs or other muscles when colliding. Nedel and Thalmann [NT98] implement these forces by using inverse dynamics: *"We specify the constraints actuating on a point and calculate the force necessary to compel the point to obey these constraints."* [NT98]

Nedel and Thalmann [NT98] compute the muscle motion by simulating the action line motion. The action line influences the muscle deformation. For each particle of the muscle surface, laws of motion are used. Nedel and Thalmann apply the commonly used Lagrange's equation:

$$m_i\ddot{x}_i + y_i\dot{x}_i + f_{result}(x_i) = f_{extern_i}$$

x_i : position

$\dot{x}_i$: velocity

$\ddot{x}_i$: acceleration

m_i : nodal mass

y_i : damping factor

$f_{result}(x_i)$: result force over node i

f_{extern_i} : external force

Equation 33: Lagrange's equation of motion

They utilise a fourth order Runge-Kutta method (cf. Chapter 4.4.5.1) to integrate the ordinary differential equations as it is more stable than the Euler's method (cf. Chapter 4.4.5.1).

Rohr [Roh04] wrote his diploma thesis at the University of Applied Science Furtwangen and is now studying *"Technical Director"* at the Filmakademie Baden-Württemberg. In his diploma thesis about *"Virtual Characters"*, he presents another approach based on a mass-spring system. Rohr [Roh04] does not orient the muscle shape according to its action line. Instead, he distinguishes between two states of how the muscle may appear. One is the muscle in a relaxed, the other in a tense state. He provides a technique to blend between both states without the use of a conventional blend shape deformer (cf. Chapter 4.3.1). Rohr [Roh04] makes use of a mass-spring system to synthesise the muscle. A system based on mass-springs benefits from properties such as elasticity or the ability to react on external forces such as gravity or collision forces.

Rohr [Roh04] makes a distinction between a mass-spring system for 2D objects and 3D objects. Given a 2D object such as a triangle, the mass-spring system has each adjacent particle connected by springs. A triangle implies three springs between each particle. In 3D computer graphics, cloth patches are commonly represented in 2D networks. 3D networks are composed of at last six springs, for example a tetrahedron. 2D networks have their springs lying on their surface. Given a closed 3D object such as a muscle, the third dimension is essential to guarantee the mesh to preserve its shape (see Figure 32). Rohr [Roh04] attempts to preserve shape integrity by setting cross connection springs. The selection of particles, he connects in a three dimensional way, is almost arbitrary.

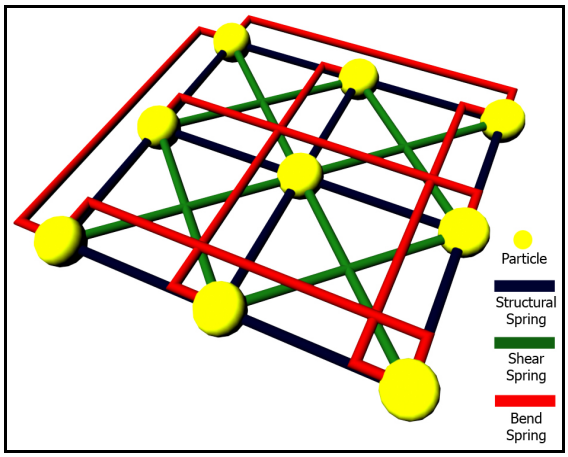


Figure 31: Two dimensional network

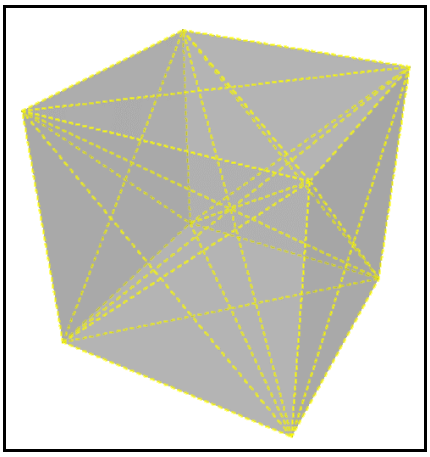


Figure 32: Three dimensional network

The network shown in Figure 31 acts in accordance to the model Lander [Lan99a] uses to describe a cloth simulation. Structural springs (see Figure 31) do not guarantee the object not to collapse. *"I really want to keep the model from shearing too much. That is, I want the space between diagonal elements of the model preserved"* [Lan99a]. Therefore, Lander creates shear springs (see Figure 31) running along the diagonals between adjacent particles. Tests revealed that structural and shear

springs in conjunction do not assure shape preservation since the shape is not maintained in extreme situations such as the cloth patch falling on the floor. He completes his model by including bend springs (see Figure 31) that describe the curvature of the shape at each particle. Due to the fact that Rohr works with a closed 3D object, he extends Lander's model [Lan99a] as previously mentioned.

Since Rohr [Roh04] does not use an action line to control the shape of the muscle, he proposes another model to deform the muscle adequately. He introduces a technique called dynamic blend-shapes. Dynamic blend-shapes work the same way as real muscles, as they contract via muscle springs. The contractile myofibrils (cf. Chapter 2.3) describe the counterparts of these muscle springs. As mentioned above, Rohr uses two states to describe the muscle deformation. Finally, dynamic blend-shapes interpolate between the two states the springs accept. A state of a spring is defined by its length. Therefore, two initial states for every spring are assigned. This can be achieved by modelling the muscle in a relaxed and a contracted state. Thus, every spring obtains two fixed values for its length. The dynamic blend-shapes are interpolated between the two values. The network of springs between the particles finally forms the muscle.

4.4.5.1 Mass-Spring Systems Analysis

In this section, we explicitly specify how a mass-spring system works as it plays a crucial role in the development of this software. We exemplify the mass-spring system utilising a two dimensional network (see Figure 31) on a polygonal mesh such as a cloth patch. Consider our elastic model is a mesh of m x n particles. See Figure 31 to comprehend the spring connections that are created:

Particle	Link to Particle	Spring Name
[i, j]	[i+1, j] and [i, j+1]	Structural Springs
[i, j]	[i+1, j+1]	Shear Springs
[i+1, j]	[i, j+1]	
[i, j]	[i+2, j] and [i, j+2]	Bend Springs

Table 2: Spring connections according Figure 31

The system is subject to following conditions: the fundamental law of dynamics: $F_{i,j} = \mu a_{i,j}$

μ denotes the mass of each particle $P_{i,j}$ and $a_{i,j}$ the acceleration caused by the force $F_{i,j}$.

Provot [Pro95] splits $F_{i,j}$ in internal and external forces. He claims the internal forces to be the result of the forces of its neighbours $P_{i,j}$:

$$F_{\text{int}}(P_{i,j}) = -\sum_{(k,l)\in\Re} K_{i,j,k,l}\left[l^0_{i,j,k,l} \frac{l_{i,j,k,l}}{\left\| l_{i,j,k,l} \right\|} \right]$$

$\Re$: a set regrouping all couples*(k,l)*such as $P_{k,l}$ is linked by a spring to $P_{i,j}$

$l_{i,j,k,l}$: equivalent to $\overrightarrow{P_{i,j}P_{k,l}}$

$l^0_{i,j,k,l}$: natural length of the spring linking $P_{k,l}$ to $P_{i,j}$

$K_{i,j,k,l}$: stiffness of the spring linking $P_{k,l}$ to $P_{i,j}$

Equation 34: Internal force

External forces can be omnipresent forces such as gravity (cf. Equation 35, (1)) or viscous damping (cf. Equation 35, (2)):

$$F_{gr}(P_{i,j}) = \mu g \quad (1)$$

$$F_{vd}(P_{i,j}) = -C_{vd} v_{i,j} \quad (2)$$

g : acceleration of gravity

C_{vd} : damping coefficient

$v_{i,j}$: velocity

Equation 35: Omnipresent external forces

The viscous damping defines the loss of mechanical energy of the model and could actually be considered as an internal force [Pro95].

A simple Euler method is applied to explicitly integrate the fundamental equation of dynamics through time:

$$\begin{cases} a_{i,j}(t+\Delta t) = \dfrac{1}{\mu} F_{i,j}(t) \\ v_{i,j}(t+\Delta t) = v_{i,j}(t) + \Delta t * a_{i,j}(t+\Delta t) \\ P_{i,j}(t+\Delta t) = P_{i,j}(t) + \Delta t * v_{i,j}(t+\Delta t) \end{cases}$$

Equation 36: Integration

In Equation 36, Δt denotes the time-step[2] and μ still defines the mass of each particle.

The calculation of the mass-spring system is commonly depending on more than one ordinary differential equation (ODE) such as Equation 34 or Equation 35. The combination of these equations may cause the system to diverge as independent variables are used [Lan99b]. Lander [Lan99a] claims that a system with high stiffness factors is also running the risk of getting instable and havoc. *"To combat this it's important to use a good numerical integrator. The midpoint method and Runge-Kutta integrators […] seem to do the trick nicely."* [Lan99a]

The Euler method is the simplest and fastest technique to solve the integration through time. Due to having small time steps, it can produce reasonable results. However, Press [Pre02] recommends not to apply Euler's method in practical use, as it shows a high error rate for large time steps and less stability. The midpoint or the Runge-Kutta method offer a promising approach to handle larger time steps.

The midpoint method also called the second-order Runge-Kutta (RK2) are defined as follows [Pre02]:

$$k_1 = hf(x_n)$$
$$k_2 = hf(x_n + \frac{1}{2}k_1)$$
$$x_{n+1} = x_n + k_2 + O(h^3)$$

Equation 37: Midpoint method / RK2

In the case of a mass-spring system, h denotes the time step Δt and $f(x_n)$ is defined by the integrator function described in Equation 36. $O(h^3)$ or $O(h^5)$ define the rounding error term. Better results are obtained by applying the fourth-order Runge-Kutta (RK4) method. This method evaluates the integrator function four times:

$$k_1 = hf(x_n)$$
$$k_2 = hf(x_n + \frac{k_1}{2})$$
$$k_3 = hf(x_n + \frac{k_2}{2})$$
$$k_4 = hf(x_n + k_3)$$
$$x_{n+1} = x_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)$$

Equation 38: RK4

Other methods exist that serve to solve a mass-spring system. Kang et al. [KCCLP00] propose an efficient technique that stably updates the mass-spring system with the use of an *"approximated implicit method"* [KCCLP00]. Furthermore, the predictor-corrector approach [DSB99] seems promising. It serves to compute a rapid approximation of the integration. Afterwards this estimate is corrected in a post-step process momentum. However, RK4 is still the one most commonly used as it enables large time steps, guaranteeing the system to stay stable in most cases [Pre02].

Mass-spring systems appear auspicious when applying a physically-based simulation. They comprise authenticity in terms of appearance and ensure physical validity. With increasing number of springs, finely woven muscle shapes require larger time steps to be evaluated. The interactive control of the muscles seems infeasible, considering the physical authenticity to be guaranteed. As a mass-spring system is time-dependent, interactive control has to take place in a special simulation environment where the system is computed over time. In Chapter 5.1.2.1, we will provide a method that bypasses the simulation environment and imitates a mass-spring system.

4.4.6 Model Based on Finite Element Method

Chen and Zeltzer [CZ92] utilise the finite element method (FEM) to simulate the geometry and underlying muscle material. The finite element method is *"a method for solving an equation by approximating continuous quantities as a set of quantities at discrete points, often regularly spaced into a so-called grid or mesh. Because finite element methods can be adapted to problems of great complexity and unusual geometry, they are an extremely powerful tool in the solution of […] mechanical systems"* [WolNDI]. The FEM and mass-spring model (cf. Chapter 4.4.5) are the most wide-spread approaches of physically-based models simulating human anatomy [Zhu98]. However, the mass-spring system cannot be applied to compute exact physical behaviour [Zhu98].

Chen and Zeltzer [CZ92] conclude that simulating human anatomy can be done using geometric, kinematical, and elastic models. Geometric models are used to synthesise a surface such as a polygonal mesh or rigid links such as bones. Kinematical models include skeleton driven surface deformations. In the elastic case, parts of the body are considered as consisting of non-linear visco-elastic-plastic composite materials that react to applied forces.

The elastic component of the system is implemented with the finite element method. Such as most physically-based mass-spring models, Chen and Zeltzer [CZ92] use a model based on the Lagrange equation (cf. Equation 33) to describe the entire system. Their equation of motion is controlled by factors like mass, damping, and stiffness between nodes (elements) which make up the muscle body. Having defined the constraints of the system, every element is combined in one system of differential equations according to its initial conditions. The equations of every

finite element are integrated to one sum. This creates a vast system of equations that may contain a great many of unknowns. Finally, this system can be solved via numerical operations.

As the system possibly includes many unknowns, the computation of the system is very time-consuming. Several efforts have been made to reduce the time to evaluate the system [Zhu98]. In fact, the performance of computers increases over the years. Hence, the FEM method will soon be applicable on a system that claims to be run in real-time. Furthermore, it has to be mentioned that interactive control of a muscle-based animation based on the finite element method is nowadays not practicable.

Several researchers adopted or improved the approach of Chen and Zeltzer [CZ92] such as Zhu [Zhu98] or Teran et al. [TBHF03] that used the finite volume method (FVM) that resembles the FEM in many ways.

4.4.7 Divers Muscle Models

4.4.7.1 Aubel

Aubel [Aub02] presents a model based on a triangle mesh representing the muscle. His model does neither require the mesh structure to be specifically organised nor a certain regularity of discretisation. Furthermore, a resampled or decimated mesh is not mandatory.

Aubel abstracts the muscle by two layers: action lines defining a skeleton and a surface mesh [Aub02]. The deformation of the muscle is exclusively controlled by lines of action as Nedel and Thalmann [NT98] previously propose. Fusiform muscles (cf. Chapter 2.3) are governed by one flat muscle approximated by several action lines. In the following, we adopt Aubel's notations, denoting a vertex of an action line as *"node"* and referring to a vertex of the surface mesh as *"vertex"*.

Aubel defines the action line as a polyline based on a set of nodes [Aub02]. The nodes serve as basis for a 1D mass-spring system whose end nodes define the rigid attachment points: the origin and insertion. He denotes this nodes as *"attachment nodes"*. The remaining nodes are dynamic always enclosed by attachment nodes. Every node of the mass-spring system has an equal mass assigned to. The spring stiffness is derived from the type of material it represents: either tendon or muscle belly. Aubel defines force fields by ellipsoids that control the mass-spring system by repelling or attracting nodes. These force fields prevent from gross penetration or from losing a certain shape.

To deform the muscle, the action line is generated by assigning *"local frames"* to every node. These frames have three coordinate axes. The z-axis is defined by the normal of the plane that is determined by consecutive nodes and the tangents of the end nodes. As Aubel [Aub02] omits the information how the x-axis is determined, we assume he uses the tangent at each node. The y-axis is identified by applying the cross product to the x- and z-axis (cf. Chapter 4.2.1.1). Aubel

[Aub02] makes several adaptations to the local frames. One adaptation serves to prevent local frames from flipping in extreme positions.

Since every node has a local frame assigned to, it can be considered as being a joint. The in-between segments are assumed to be bones such as Thalmann [TLT88] and Sun [SHSI99] previously proposed.

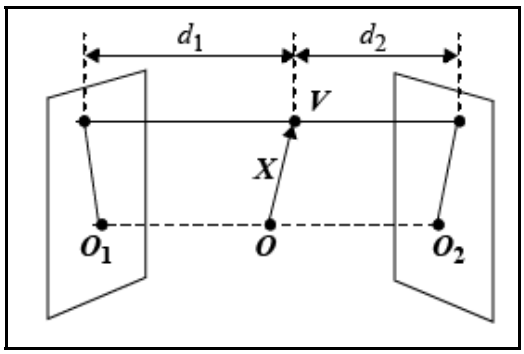


Figure 33: Mapping a vertex to an action line segment[SHSI99]

$$O = t * O_1 + (1-t) * O_2$$
$$\text{where } t = \frac{d_1}{d_1 + d_2}$$

Equation 39: Reference point

To map a vertex to an action line segment, Aubel [Aub02] projects the vertex onto the planes spanned by the x- and y-axis of the nodes. The two closest planes define the reference of the vertex. Aubel computes a reference point on the action line segment as shown in Equation 39. O_1 and O_2 refer the nodes of the planes that represent the closest reference. d_1 and d_2 are the distances between the vertex and its projected points. Afterwards, Aubel assigns vertices to the action lines. He defines zones of influence for every action line and assigns every vertex to each action line depending on this influence areas and the distance to the action lines. Aubel guarantees curvaceous action lines to be assigned properly by making several adaptations.

Finally, Aubel stores the parameterised reference points (cf. Equation 39 and Figure 33) for each surface vertex. Depending on its influence area, the vertex is translated driven by the deforming action lines.

Aubel [Aub02] claims his muscle to preserve its volume while contracting. Therefore, he considers a muscle adopting the shape of a generalised cylinder running along an action line. Wilhelms and Gelder [WG97] previously suggested *"to scale the width and height of each cross-section by a ratio related to the change in length of the axis of the cylinder"* [Aub02] [WG97]. Aubel defines the volume by multiplying the cross-section area at each node with the curve arc length from the node to its consecutive node [Aub02]. Changing the elongation affects the dimension of the cross-section areas. Depending on the contraction of the action lines and their influence on a certain vertex, they are either pushed away from or attracted towards the curve. Due to the fact that this computation method is just an approximation, Aubel states that the volume preservation varies

from 1-2 % to 9 % when the muscle shortens by 30 %, which corresponds to the maximal physiological compression rate [Ric81].

Aubel [Aub02] additionally provides a tool to create the action line semi-automatically. The user only has to determine the origin and insertion the number of bellies and tendons. The tool creates the action line with an adequate count of nodes. The nodes are distributed equally along the action line with a certain spacing. The tool automatically determines the parameters for the mass-spring system. Given a straight action line, the user has to define force fields to form the shape of the action line. The force fields provide further control to influence its trajectories. Utilising ellipsoids, two kinds of forces can be modelled. In *"radial mode"* the action line is attracted towards or repulsed from the centre of the closest ellipsoid. In *"orthogonal mode"*, the action line is attracted to its orthogonal projection on the closest ellipsoid so that it slides freely on the surface of the ellipsoid.

Utilising the force fields, Aubel [Aub02] is able to avoid complex collision detection. Collision with bones can easily be simulated using force fields. Muscle-to-muscle collision can be simulated by attaching force fields to action lines. Aubel emphasises that the use of force fields does not completely cover precise calculation of collision and that penetration cannot be entirely avoided.

Aubel distinguishes between isotonic and isometric contraction (cf. Chapter 2.3.7), where isotonic contraction is exclusively defined by the elongation of the action line. Isometric contraction means that the muscle is in a tense state not controlled by the length of the action line. Aubel allows the user to determine the deformation. In rest position, Aubel provides access to enable the user to remodel the shape by modifying the width and height at every action line node. The new shape is controlled by the so called *"activation curve"*.

To make the muscle motion appearing more believable, Aubel enables to simulate *"inertia induced oscillation"* [Aub02] of the muscle. Additionally to this effect, he involves viscosity effects in his model. All effects are simulated based on a 3D mass-spring system.

At last, Aubel shows how the mass-spring system is evaluated. *"The mass-spring system that governs the evolution of an action line deforms itself according to the Lagrange equation of motion"* [Aub02] (cf. Equation 33) *"The position of the i-th (dynamic) node of the action line is given by the following second-order differential equation:"* [Aub02]

$$\frac{d^2 X_i}{dt^2} + \gamma \frac{dX_i}{dt} + f_{i-1}^{elastic} + f_{i+1}^{elastic} + \sum_k f_k^{forcefield} = 0$$

Equation 40: Equation of motion

The mass value of the nodes can be omitted as they are equal throughout the system. The first term defines the node's acceleration, the second stands for the velocity where γ is the viscosity damping factor. The third and fourth term determine the force of the previous and next node, the fifth term includes the influence of the force fields. The elastic forces contain factors such as damping and stiffness.

Recall Chapter 4.4.5.1, where the analysis of mass-spring systems is exemplified in detail. Aubel compares the integrators introduced in Chapter 4.4.5.1 such as the midpoint method or RK4. He even tests the Runge-Kutta method fifth order (RK5). He finally decides to use the adaptive Runge-Kutta method whereas adaptive refers to the adaptation of the time steps for every action line causing the system stay stable. Statistics show that using RK4 is a better choice as it is much faster and shows almost the same results as RK5 concerning stability.

Aubel attempts to make the system usable in real-time. First he removes the elasticity of the action line. Therefore, he calls the resulting action line an *"admissible path along with muscle components spread"*. Next, he excludes some nodes from collision detection such as the insertion and origin. These two adaptations speed up the entire system so that Aubel claims that it can be run in real-time.

Our approach we discuss to a later date has several parts of Aubel's model adopted. We completely implement some of his techniques, others are adapted or improved. In fact, Aubel's model seems to be one of the best approaches that involves authenticity of appearance of muscles and applicability in real-time. However, Aubel's approach does not specialise in facial animation. Facial animation has some other restrictions that make his model insufficient in many aspects. A good example for another limitation is that in his approach the bones drive the muscles. In reality, it is the other way around. In facial animation, only the muscles control the deformation of the skin. In fact, they are attached to the skull but the skull stays rigid apart from the mandible.

4.4.7.2 Kähler

Kähler wrote his dissertation about *"A Head Model with Anatomical Structure for Facial Modeling and Animation"* [Kae03]. His model involves a muscle-based skin deformer. He provides two types of muscle representation. The first method uses sets of ellipsoids to embody the facial muscle structure. The second one restricts to box-like shapes to form muscles.

Kähler's muscle model [Kae03] distinguishes two kinds of facial muscles. A linear muscle that contracts along its elongation, and a sphincter muscle which contracts towards its centre. He combines several linear muscles into parallel groups to model sheets of arbitrary width. The linear muscle model works on the basis of a polyline (cf. action line in Chapter 2.3.7, 4.4.3, 4.4.5,

4.4.7.1) which has one point at the end staying rigidly attached to the skull. The sphincter muscle has an additional point that defines the centre which the muscle is contracting towards.

Similar to the action line model Aubel [Aub02] or Wilhelm and Gelder [WG97] use, Kähler [Kae03] defines a local coordinate frame (cf. Chapter 4.4.7.1) not for every control point of the line but for every segment. He uses overlapping ellipsoids to represent the muscle surface for every segment as *"the attachment process […] requires ray intersection tests, normal computation, and inside / outside tests to be performed on the geometry."* [Kae03] Ellipsoids are overlapped to avoid large holes in the approximated muscle surface. As the transitions between the ellipsoids show gaps, for every segment, Kähler uses a box-like shape whose corners are connected to adjacent shapes.

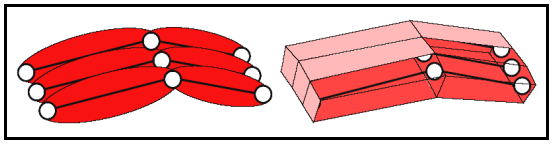


Figure 34: ellipsoid / box-like representation [Kae03]

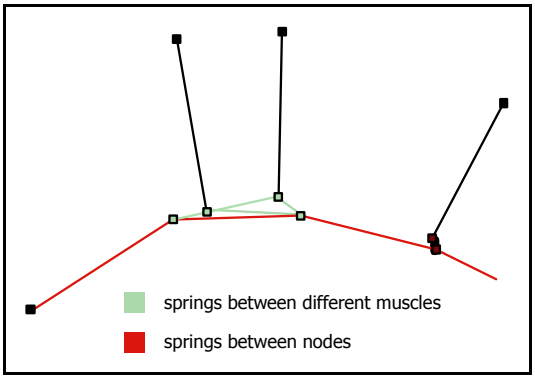


Figure 35: Spring structure [Kae03]

Kähler's model [Kae03] enables the definition of connection constraints between muscles. Since parallel muscles can be combined to a union, the corresponding control points of every single fibre are averaged to one. In addition to compound muscles, the free ends of linear muscles can be connected to other muscle segments. Kähler's model provides automatic detection of feasible muscle connections by testing on intersection of muscle shapes at the moment of initialisation. By determining connection constraints, the mass spring system is established. All adjacent nodes are linked by springs (see red lines in Figure 35). Next, muscles are connected to other muscles by springs (see green lines in Figure 35).

The contraction of a muscle is independent of the mass-spring system. Consider we have a linear contracted muscle. The polyline is contracted along its original shape in rest position. Every point of the line is pulled towards its end point attached to the skull or jaw. Given a sphincter muscle, every point of its closed polyline is displaced towards its ring centre when contracting.

Combined parallel muscles can now be deformed by moving the average control point that governs every muscle of its union. The corresponding control points are connected by springs

to each control point of the parallel muscle. If the muscle union is attached to another muscle, the average control point is connected to the other muscle by a spring.

When the control point of the polyline moves, its length changes. When the muscle is stretched, the polyline spreads along a flat line. Confer Figure 36 to see how nodes are translated that are not members of a constraint group:

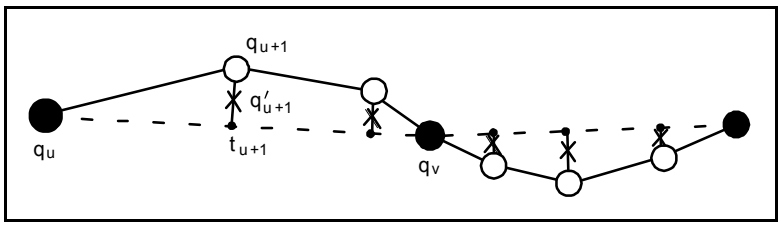


Figure 36: Straightening of a muscle, black nodes are constrained [Kae03]

According to the elongation of the muscle fibre, the nodes of the polyline are mapped along the line segments between the constrained nodes. In Figure 36, the crosses mark the new positions of the nodes while the line is not fully stretched. The nodes lie between the node in rest position and the point of the projected node on the segment (see Figure 36, dashed line). Based on the proportion of the current length with the rest length of the polyline, each node moves towards or away from its corresponding straight line segment.

Kähler [Kae03] differentiates two kinds of bulging and thinning of the muscles. Sphincter muscles thicken or thin evenly along their ellipse when contracting. In general, muscles are bulged due to the proportion of the current muscle's elongation to its elongation in rest position. Kähler [Kae03] induces linear muscles not to bulge evenly along their elongation. According to a simple quadratic function, the bulge factor increases towards the centre of the muscle. Approaching the end points of the muscle, the bulge factor decreases.

At last, Kähler [Kae03] provides several techniques to ease the creation of the muscle system. He provides a tool that allows the user to sketch the muscle shape above the skin. Kähler's tool projects this sparse shape and creates a proper muscle under the skin. His system aims for the quick setup of the muscle system.

Several features of Kähler's model [Kae03] are implemented in our tool (described in Chapter 5). In fact, Kähler's model has some lacks concerning authenticity and preciseness of the muscle simulation. The results show that it can well be improved. Kähler's main focus appears to be on an application that can be used in real-time implying a system that enables a fast construction. Our model claims to be interactively controllable but not necessarily able to run in real-time. Therefore, some parts of Kähler's model have been adopted, improved, or mixed with other models.

4.5 Muscle-Driven Skin Deformer Models

In Chapter 4.4.2, we describe a model that specialises in direct skin manipulation. As in this model the muscle geometry and the skin shape are not separated, it will not be commented on any further in this section. The most muscle-driven skin deformer models resemble in some ways, so we will not commit ourselves to every approach in detail. Besides to that, it has to be mentioned that this thesis has its major focus on the realistic animation of muscles and that skin deformation represents a secondary task. Some of the approaches are additionally specified in Chapter 5.3, as they are implemented in our software.

4.5.1 Anchoring Skin Vertices by Damped Springs

We have to emphasise that the skin deformer approach described in this section is based on the model by Wilhelms [Wil94] discussed in Chapter 4.4.4.

Wilhelms [Wil94] makes a distinction between generation, anchoring, modification, and adjustment of the skin while animated. Since we are dealing with skin deformer models we do not have to address the issue of skin generation.

Wilhelms [Wil94] iterates through the entire skin, determines the closest ellipsoid (cf. Chapter 4.4.4), and marks it as anchor ellipsoid. Based on the Newton-Raphson method, she computes the nearest point from skin on the ellipsoid in respect of the ellipsoid's dimensions. Having calculated this nearest point on the ellipsoid surface, she stores it parameterised by the dimensions of the ellipsoid. Additionally to the closest point, she retains the distance between the skin point and the closest point on the ellipsoid.

After the ellipsoid has changed its shape, the parameterised nearest point (anchor point) can be remapped by scaling the dimensions of the ellipsoid. As a consequence, the skin point and the anchor point move with the varying ellipsoid. Wilhelms [Wil94] additionally defines a virtual anchor that is represented by the skin point parameterised by the ellipsoid while being in rest state (see Figure 37):

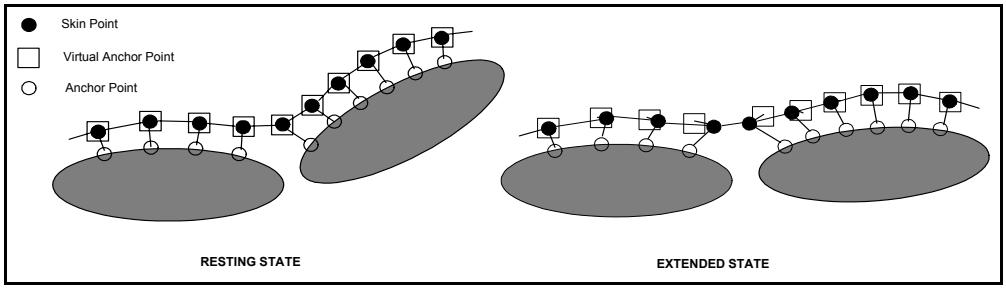


Figure 37: Attachment of skin points [Wil94]

Wilhelms [Wil94] allows the user to make several modifications to the skin deformer. The user is able to define the *"spring constant"* (k) that controls the stiffness of the connection between one point and its driving point. Furthermore, the rest length can be manipulated forcing a skin point to be farther from or closer to the ellipsoid.

During motion, the skin adjustment is computed by mapping the skin points from world to local space of the ellipsoid. Given the ellipsoid's shape modified or translated, the new skin points are remapped into world space via the inverse of the matrix that defines the transformation from world to local space. Since the skin point is connected to the anchor point, to the virtual anchor point, and to neighbouring skin points (see Figure 37), all these connections influence the skin point during variation of the underlying ellipsoid.

"The algorithm for skin adjustment is not using physical simulation, and no integration is used. However, the effect is much the same. Deviations from the rest length of edges connected to each skin point are used for the adjustment." [Wil94] Thus, Wilhelms does not require a time-dependent simulation environment as she evaluates the system iteratively. Such as a common mass-spring system (cf. Chapter 4.4.5), varying magnitudes are attempted to be balanced. Wilhelms applies the mass-spring system depending on the number of iterations the user defines. In the same manner as the mass-spring system, each point has a force applied to. She does not describe the influence on a skin point based on forces, but the skin point affected by the sum of changes of its connections. She denotes the connections as edges. Assume for now l_r is the rest length of a particular edge, l_c the current length of this edge, and P the vector from the skin point to its connected point. The *"spring constant"* is denoted by k that is by default set to 1. The change of this edge's position is computed as depicted in Equation 41:

$$Dp = k * \frac{l_c - l_r}{l_c}$$

Equation 41: Change in position due to one edge

$$P_{t+1} = P_t + \sum Dp_i$$

Equation 42: Change of a skin point

The next iteration step, the new position of the corresponding point is computed (in a time-dependent simulation this would be the next time-step). P_{t+1} receives its new position due to its connected edges that are specifically displaced (cf. Dp in Equation 42).

Wilhelms [Wil94] points out that they have implemented collision detection but that it slows down the system remarkably although having ellipsoids as a basis for detection. She emphasises that reacting on collision changes the visual appearance slightly and can therefore be omitted.

Wilhelms approach of skin deformation seems to achieve convincing results. In fact, her proposal of using ellipsoids is not the best choice as already noticed in Chapter 4.4.4. Thus, her model, whereby she parameterises all required data depending on ellipsoids' properties, is not sufficient for an approach that claims to be more precise.

Wilhelms refines her approach in collaboration with Gelder [WG97]. Recall Chapter 4.4.3, where the model of Wilhelms and Gelder is described. They specify the muscle surface based on a deformed cylinder. Instead of being parameterised by the dimensions of the ellipsoid, the parameters are controlled by the dimensions of the deformed cylinder. Wilhelms and Gelder [WG97] create the parametric dependency by relating the anchor and the virtual anchor (ut sup.) to the two closest slices of the deformed cylinder (see lighter skin vertex between slice 4 and 5 in Figure 25). A *"warped cell"* [WG97] is spanned between the two slices (see parametric trilinear cell in Figure 25) which defines the space for the anchors (similar to free-form deformers described in Chapter 4.2.2.2). By drawing a connection from the anchor position in world space to the space of the warped cell, the anchors are provided with the essential parameters. Wilhelms and Gelder [WG97] call this kind of parametric dependency *"parametric trilinear transformation"*. *"Trilinear"* denotes the three dimensions of the warped cell. Having the parametric position defined by (s, t, u) [WG97], the new world space position can be computed as soon as the muscle shape alters. Storing the parametric positions and changing the shape of the warped cell, it is allowed to remap the anchors into world space.

In addition to adapting the ellipsoid model to the deformed cylinder model, Wilhelms and Gelder [WG97] provide a method to automatically determine the spring stiffness k (ut sup.) of each edge (ut sup.). Finally, the technique to evaluate the new position of a skin point is similar to the approach introduced in Chapter 4.4.4 and is therefore not further discussed.

Besides to that, it has to be alluded to that Turner and Thalmann [TT93], Lee et al. [LTW95], and Ng-Thow-Hing [Hin01] apply an approach to deform the skin which is quite similar to Wilhelm's approach [Wil94].

4.5.2 Linear Elasticity Model

Aubel [Aub02] is one of the first to provide a method to resample a polygonal skin mesh. Given a rough sketch of the body skin, a rays are shot away from the entire skeleton in a *"star-shaped manner"* [Aub02]. The intersections of the rays serve as new points for a resampled shape. Difficulties may occur when a ray intersects with the skin of body parts that are not intended to be hit. To guarantee an appropriate assignment for each ray, the skin shape is divided into body parts. Smoothing the resampled mesh leads to a proper appearance of the skin. This method can

be applied to models generated by a 3D scanner or obtained by the National Library of Medicine [VHP86I], for example.

Aubel [Aub02] makes use of the Lamé equation that adequately synthesises the hypodermis (cf. Chapter 2.2.3) consisting of homogeneous, anisotropic, linearly elastic material (cf. Chapter 2.2.3, derived from Debunne [DDBC99]):

$$\rho\vec{a} = \mu\Delta\vec{d} + (\mu + \lambda)\nabla(div(\vec{d})) + \vec{f}_{ext}$$

ρ : mass density

μ : Lamé constant, Young's modulus, behaviour of material

λ : Lamé constant, Poisson coefficient, behaviour of material

$\vec{a}$: acceleration of an element of small matter

$\vec{d}$: displacement

$\Delta\vec{d}$: Laplacian of the displacement field

div: the vector formed by the divergence of each line(matrix)

$\nabla(div(\vec{d}))$: gradient of divergence(gradient of volume expansion)

$\vec{f}_{ext}$: set of external forces

Equation 43: Lamé equation

$$c_1 = \sqrt{\frac{\mu}{\rho}}$$

$$c_2 = \sqrt{\frac{\lambda + 2\mu}{\rho}}$$

Equation 44:Velocities of propagated longitudinal and transversal wave

To define the numerical integration (cf. Chapter 4.4.5.1) of the system, Aubel discretises Equation 43 in time and space [Aub02]. Equation 44 provides the time step for integrating on Equation 43. Consequentially, it is avoided that waves miss *"discretisation nodes"* [Aub02] while propagated through the linear elastic material. Missing discretisation nodes may cause the simulation to diverge. Compare Chapter 4.4.5.1 where methods are described preventing a mechanical system such as the mass-spring system from losing stability. These techniques can certainly be applied on this system. Just as his muscle model (cf. Chapter 4.4.7), Aubel decides to apply an *"adaptive high-order explicit scheme"* [Aub02] described in Chapter 4.4.7 (adaptive Runge-Kutta 4) and in Chapter 4.4.5.1. Furthermore, Aubel also considers artificial visco-elasticity that provides better stability to the system.

In order to anchor the skin to the muscle or to the skeleton layer, Aubel uses a triangular mesh and differentiates between inner (muscles, skeleton layer) and external border (skin). The hypodermis is located between both borders having its expansion controlled by the user.

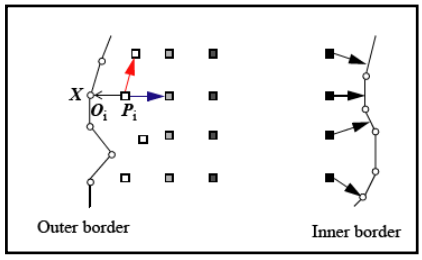


Figure 38: Anchoring skin to hypodermis particles [Aub02]

Next, Aubel discretises the hypodermis by defining particles lying inside the hypodermis (see Figure 38, left). These particles, located next to the inner border, are di-

rectly anchored to the triangular mesh of the muscle or skeleton layer by projecting the particle. This way, the closest point to the particle is computed. The anchor is parameterised by defining the barycentric coordinates (cf. Chapter 4.2.2.3) on the triangle which the closest point is located on. Skin vertices are also connected to adjacent particles. Aubel anchors the skin vertex not only to one but to several particles that are positioned nearby. He calculates the position X of the skin vertex anchored to a particle P_i as follows:

$$X = P_i + M_i O_i$$

Equation 45: Position of skin vertex

$$X = \frac{\sum_{i \in link}^{k} w_i (P_i + M_i O_i)}{\sum_{i \in links} w_i} \text{ where } w_i = \frac{1}{\|O_i\|}$$

Equation 46: Position of skin vertex connected to several particles

As depicted in Figure 38, M_i denotes a 3 x 3 matrix composed of the normalised vectors from P_i to three adjacent particles (*"triplet"*) lying in a range with a certain radius [DDBC99]. Every possible triplet is tested on singularity and the one most secure is chosen. Finally, the position of a skin vertex is computed utilising the weighted average of the local coordinates of k connected particles. O_i denotes the offset vector of a connected particle in respect of the skin vertex [DDBC99] (cf. Equation 46).

Since this technique to synthesise fatty tissue is expensive in terms of computational effort, Aubel exclusively applies this method to body regions where fat abounds such as the breast, for example. For the sake of speed, he invents a technique to deform the skin properly that is faster as the one previously described. Since our tool uses an approach that resembles Aubel's one, we will provide details thereof in Chapter 5. For now, we will only touch some aspects of Aubel's procedure.

Aubel differentiates between rough vertex positioning on the surface of the limb or trunk, and the extrusion by the musculoskeletal system. For the rough vertex positioning, he uses the joints of the skeleton system. Compare Chapter 4.3.3.1, where we described how joints influence the skin. He uses a *"smooth bind"* method whose numerical analysis is further discussed in Chapter 5.2.2.1. Smooth binding the skin means that each skin vertex is positioned based on a weighted transformation induced by every joint of the system. Aubel reduces the influence on every skin vertex by assigning the maximum of two influencing joints and depending on weights. This technique is completely independent from the shape of the muscle or the bone.

The displacement induced by the musculo-skeletal system is computed by storing the initial distance of every skin vertex to its underlying structure. When the underlying layers deform, the

initial distances are tried to be retained. This means that the muscles or the bones freely slide under the skin despite displacing the skin. Aubel speeds up the process by using hierarchical bounding boxes to isolate the required ray-casting that defines the displacement.

However, Aubel neither provides exact information about how fast his skin deformation system works nor to which degree it is interactively controllable. Hence in Chapter 5.3.2.2, we will explain why the displacement induced by the musculo-skeletal system requires an immoderate computational effort.

Many researchers dedicate themselves to the analysis of skin deformation. Only some of them describe muscle-based skin deformers. Since most of these approaches are similar in their principles, we picked two models described in this chapter. It has to be stated that intricate approaches such as the one Aubel [Aub02] claim to be physically precise. Results of simpler techniques show that they suffice in terms of rapid computation and authenticity of appearance. Simpler approaches have the deformation and the displacement of the skin separated. However, it has to be emphasised that most times extrusion is not resulting by skin being compressed based on the characteristics of the hypodermis' material. Skin is rather coarsely displaced exclusively driven by the muscle motion and deformation. Furthermore, simpler models do not allow to compute wrinkles as they depend on skin characteristics and can therefore not be generated on the basis of the muscle.

5 Implementation

"Nothing is more revealing than movement."[1]

The advent of diverse theories paved the way for many computational implementations of muscle-based systems. The growth of innovations focussing on facial animation lacks in comparison to remaining body parts. In this thesis we describe a wide variety of models that in totality form the foundation for creating credible, parameterised muscle-based facial animation.

In this section, we will derive a model for facial animation based on lifelike, anatomical characteristics. We will differentiate between the skull, the muscles, and the skin of the face. Since modelling the skull does not require sophisticated representations, it will only be referred to. The muscle which is utilised for the animation will be divided into three models: the action line model (cf. Chapter 2.3.7), the muscle shape model, and the skin deformer model. Both models will separately be discussed in detail.

We comment on our system on a theoretical basis and we additionally provide insight into the software we developed in Alias Maya®. Maya® serves as platform to experiment with existing models and to develop new approaches. We expand on the progression that leads to the final realisation. Utilising Maya®, we elaborate on the architecture of our software and show exemplary source code to some extent.

5.1 The Action Line

In Chapter 2.3.7, it shaped up as the force of a muscle runs along a certain line of action. This action line goes through the centroids of the cross sections of the muscle [JD75]. It has two attachment points: the origin and the insertion. Most of the body's muscles not belonging to the head are attached to bones.

In our approach, the action line is the component that drives the muscle shape. Conversely, the muscle shape does not affect the action line. Forces acting on a muscle, such as other attached muscles or the collision with bones or other muscles, cannot be propagated back to the action line. Therefore, all reactions to forces are handled by the action line itself.

[1] **Martha Graham**

Our final model provides two different kinds of deformation. One is driven by a contraction factor and the other by external components to which points of the action line are attached to. In the following chapter, we will describe several experiments we conducted to implement the action line.

5.1.1 Experiments

5.1.1.1 Geometric approach for Muscle Protraction

Kähler [Kae03] concludes that stretching an elastic muscle straightens the points of an action line according to its elongation. It must be stated that the action line consists of a polyline with straight segments; in other words a NURBS curve of degree one (cf. Chapter 4.2.3).

The muscle relaxation is performed in two steps. For the first step, we developed a model that stores the control points of the action line depending on its elongation. Stretching the action line preserves the original shape depending on its elongation (see Figure 39). The second step is the relaxation of the stretched action line along its elongation.

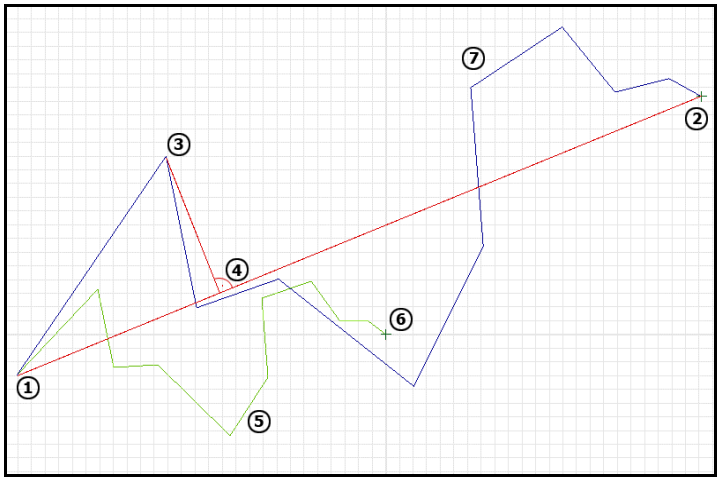


Figure 39: Stretching the action line

During the first step, the proportions of the polyline are computed. This can be seen in Figure 39, where the red line from origin (see Figure 39, 1) to insertion (see Figure 39, 2) defines the elongation of the stretched line. Using the original action line (see green line in Figure 39, 5), we project the control points onto the elongation line. We then store the projected control point (see Figure 39, 4) parameterised by the elongation line. The length of the red line between the control point and its projection on the elongation line (see Figure 39, line from 3 to 4) is stored as well. All parameters are stored based on the original action line. Stretching the action line affects the elongation. The line between the control point and its projection is now computed relative to the elongation utilising the theorem of intersecting lines:

```
// oCVs -> point array of original control vertices
// cPU  -> index of origin
// cPV  -> index of insertion
// rL   -> relative location, displacement of insertion

for(i=0;i<oCVs.length();++i) {

 MVector a, b, bProOnA, c;
 MVector aNew, bNew, bProOnANew;
 MPoint bProOnAPoint;
 double scaleOnA, stretchFactor;

 a = oCVs[cPV] - oCVs[cPU];                  // elongation vector
 b = oCVs[i] - oCVs[cPU];                    // vector to be projected
 bProOnA = (b * a.normal()) * a.normal();    // projected CV parameterised by a
 bProOnAPoint = oCVs[cPU] + bProOnA;         // coordinates proj. vector on a
 c = oCVs[i] - bProOnAPoint;                 // orthogonal vector on a to oCV
 scaleOnA = bProOnA.length();                // scale of projection vector on a

 // new elongation with displacement added
 aNew = oCVs[cPV] + rL - oCVs[cPU];

 // stretch factor proportion original to new elongation
 stretchFactor = aNew.length() / a.length();

 // re-scale and stretch normalised a -> new projected CV parameterised by a
 bProOnANew = aNew.normal() * scaleOnA * stretchFactor;

 // adding on new orthogonal vector
 oCVs[i] = oCVs[cPU] + bProOnANew + c * stretchFactor;

}
```

Source Code 2: Stretching of action line

After stretching the action line and leaving the original parameterised shape depending on the elongation, the action line is relaxed.

To relax the action line, it is interpolated between the stretched shape (ut sup.) and the original action line with every segment evenly spread along the straight elongation line (see dotted line in Figure 40). The normalised elongation vector is computed (see $\vec{e}$ in Figure 40). Now, every segment is placed along this vector in the same succession. With the control points in the relaxed state, it is interpolated (see grey dashed line in Figure 40) between them and the control points of the shape in the stretched state depending on the stretch factor of the elongation.

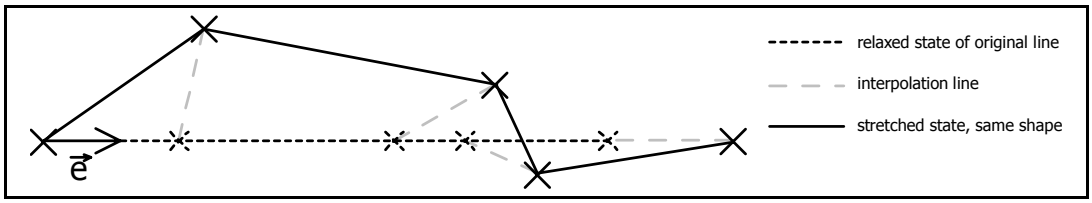


Figure 40: Relaxation of action line

The more the elongation of the action line converges to the elongation of the relaxed action line (see dotted line in Figure 40), the more the control points are displaced along the interpolation line (see grey dashed line in Figure 40).

However, this model lacks accuracy and authenticity. Approaching the relaxed state along the interpolation line (see grey dashed line in Figure 40) results in segments not maintaining their original length. This is due to the fact that the stretch and relaxation is computed as being dependent on the elongation line. The proportions referring to the elongation are correct, however, the dimensions of the single segments do not change proportionally.

A line of action of a real muscle does not behave like the action line computed in this experiment. Displacing the insertion should change the last segment of the action line, followed by the attached segments until having reached the origin. In this case, moving the insertion deforms all segments in the same manner, all according to the elongation. The behaviour of an action line of a muscle is similar to a string (irrespective of the muscle elasticity) driven by muscle forces. Imagine a string lying on the floor. If this string is not in a stretched position, moving one end does not necessarily deform the whole string at once. Rather the displacement propagates through the string from insertion to origin.

The idea of propagating the displacement of the insertion along the action line to the origin is the basis of an inverse kinematic (IK) system. In the following sections, we will improve the computation of the action line based on an IK-system.

5.1.1.2 IK-System-Based Muscle Protraction

As mentioned above, the application of an IK-system could solve the problem of a consecutive cycle through the action line for every segment. We will discuss two approaches that deal with the solution of an IK-system. The first is called the Jacobian transpose method [Wel93]. We have implemented this method only to a certain degree, as it turned out to be more complex to assure the system to be evaluated correctly. Hence, we will not excessively discuss every aspect of this approach. The second approach is called the cyclic-coordinate descent method (CCD) which overcomes some of the limitations of the Jacobian transpose method [Web02].

The Jacobian Transpose Method

An IK-system is typically applied when we assume the segments of the action line to be bones and the control points to be the joints. In this chapter, we will refer to the insertion control point as *"end effector"* [Par01]. We will show an example in 2D space which can easily be transferred into 3D space.

The Jacobian transpose method is an approach that consecutively computes the motion of each joint. It determines the change of every joint angle based on the variation of the end effector. To compute the changes, the partial derivatives in a matrix called the Jacobian matrix are required.

Consider a set of n equations depending on m variables: $y_i=f_i(x_1, x_2, ..., x_m)$, $0<i<n+1$. To describe the change of the output variables relative to the input variables, the differentials of y_i are defined using the chain rule. The matrix of these partial derivatives is called the Jacobian [Par01] (cf. Equation 47).

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix}$$

Equation 47: Jacobian matrix

$$V = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \varpi_x \\ \varpi_y \\ \varpi_z \end{bmatrix}$$

Equation 48: Position and orientation changes of end effector

$$\dot{\theta} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \vdots \\ \dot{\theta}_n \end{bmatrix}$$

Equation 49: Joint angle changes

For this IK-system, the input variables are defined by the changes of the joint angles (cf. Equation 49) and the output variables are the difference of the current to the goal's orientation and position of the end effector (cf. Equation 48). The Jacobian matrix (cf. Equation 47) associates the input and output variables:

$$V = J(\theta)\dot{\theta}$$

Equation 50: Relation of joint angles to end effector

The alteration of the end effector's orientation is defined by ϖ. It is determined by the changes of the given joint's angle around the axis of revolution (see Z_i in Figure 41):

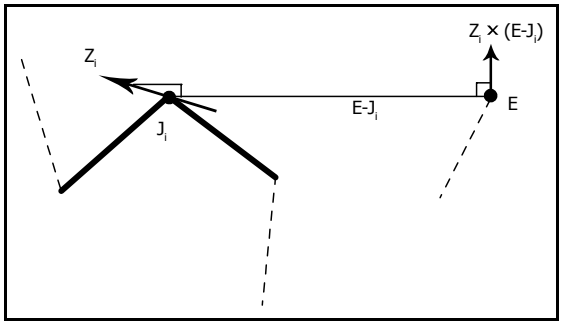


Figure 41: Change in position [Par01]

The cross product of the revolution axis and the vector pointing from the joint to the end effector represents the change of position (see. Figure 41). Figure 41 illustrates the linear change of position of the end effector at present, aroused by the change of the joints angle. With all required information gathered from one coordinate system, all joints angles can be determined. Figure 42 shows an example of a planar IK-system with three joints.

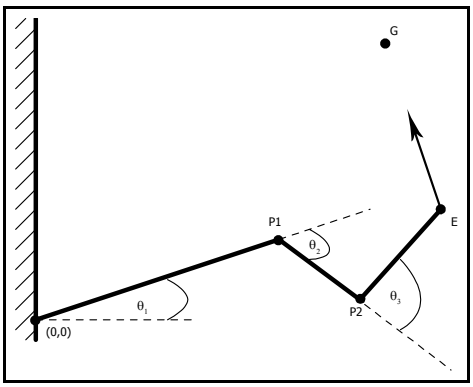


Figure 42: Planar, three joints [Par01]

As the IK-system in Figure 42 is planar, one axis of revolution is required to compute the change of the joint angles (cf. Equation 51, based on Equation 50):

$$\begin{bmatrix}(G-E)_x\\(G-E)_y\\(G-E)_z\end{bmatrix}=\begin{bmatrix}((0,0,1)\times E)_x & ((0,0,1)\times(E-P_1))_x & ((0,0,1)\times(E-P_2))_x\\((0,0,1)\times E)_y & ((0,0,1)\times(E-P_2))_y & ((0,0,1)\times(E-P_2))_y\\((0,0,1)\times E)_z & ((0,0,1)\times(E-P_3))_z & ((0,0,1)\times(E-P_2))_z\end{bmatrix}*\begin{bmatrix}\dot{\theta}_1\\\dot{\theta}_2\\\dot{\theta}_3\end{bmatrix}$$

Equation 51: Planar IK-system, three joints

In the case of a planar IK-system with three joints, the Jacobian forms a square matrix. Thus, the system of equations can easily be solved. Given a rectangular Jacobian matrix, the system can be solved using the *"pseudo inverse"* of the Jacobian matrix. If the Jacobian matrix J exists, the system is computed as follows:

```
JT = ~J;                          // Jacobian matrix transposed
Jsquare = J * JT;                 // Jacobian matrix made square
try{                              // Try-Catch statement to suppress singularity
 JsquareInverse = !Jsquare;       // Inverting square matrix
}
catch (const std::exception& e) {
 cout << "Error in actionLineDeformer: " << e.what() << endl;
}
JPseudoInverse = (~J) * JSquareInverse;   // Compute pseudo inverse of Jacobian
Theta = JPseudoInverse * V;               // Compute Theta, joint angle changes
```

Source Code 3: IK-system based on pseudo inverse of Jacobian matrix

In 3D space, the joints rotate around two axes instead of one. The Jacobian Transpose Method therefore becomes more complex in that the system must be solved for two angles. This is caused by the degree of freedom (DOF) of the joints that increases in 3D space.

We did not pursue the Jacobian transpose method any further due to its limitations. The first limitation is that we have to avoid controlling the IK-system in a time-dependent simulation environment. The system is interactively used. In this interactive system, large changes of joint angles or changes of the end effector may occur. *"This means that if a too big step is taken in joint angle space, the end effector may not appear to travel in the direction of the goal."* [Par01] Utilising the pseudo inverse causes the system to compute one of many solutions. The system does not solely behave realistically concerning joint rotation [Par01]. It can be improved to better control the kinematic model. These adaptations would go beyond the scope of this thesis. We therefore concentrate on models that can be solved faster and that are more intuitive in application. The main reason not to use the Jacobian transpose method is the singularity problem that some configurations may cause [Wel93]. The cyclic coordinate descent (CCD) method described next is an effective alternative that is often applied in real-time animations.

Cyclic Coordinate Descent method

The cyclic coordinate descent (CCD) method is an iterative heuristic approach. This approach is often used in game applications as its computation is simple and efficient. The CCD method

attempts to minimise position and orientation errors by varying one joint variable at time. As it is a heuristic approach, it does not claim to produce mathematically correct results.

Each iteration starts with the rotation of the deepest child bone in the skeleton. Its orientation is changed so that it points towards the goal position of the end effector (see Figure 43, left). Next, the parent of this bone is rotated so that the line from this bone to the end effector points towards the goal of the end effector. In the next iteration step, the same is performed for its parent bone.

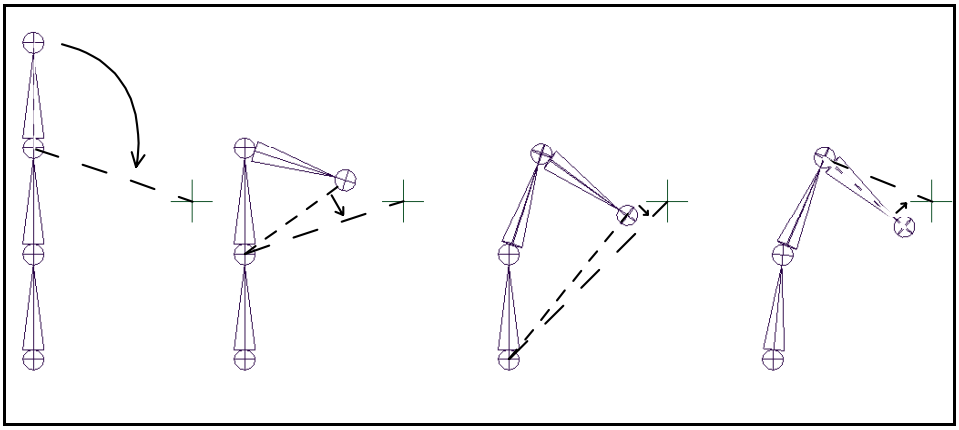


Figure 43: CCD method [Web02]

The algorithm processes every bone of the chain from the deepest to the top bone. Having cycled through the skeleton, the loop starts from the deepest bone (see Figure 43, right). By increasing the count of iterations, the final result can be further refined. The CCD method has the advantage that *"even if the effector cannot be reached, the solver can make a solid attempt without becoming shaky or unstable"* [Web02]

```
// oCVs -> point array of original control vertices
// cPU  -> index of origin
// cPV  -> index of insertion
// rL   -> relative location, displacement of insertion

MPoint goal = oCVs[cPV]+rL;              // goal of end effector

for(int iter=0; iter<iterDepth;iter++) {// iterations of CCD
 for(int k=cPV;k>cPU-1;--k) {            // iteration through skeleton

  MVector btt  = oCVs[cPV]-oCVs[k];      // base of bone to chain tip(end effector)
  MVector btg = goal-oCVs[k];            // base of bone to goal of end effector
  stat = btg.normalize();
  MVector btp = btg * btt.length();      // btt rotated on btg
```

```
  // store rotation of bone
  MQuaternion angleAndAxis = btt.rotateTo( btp );
  for(j=k+1;j<cPV+1;j++) {              // propagate rotation through chain until
   MVector btjp = oCVs[j]-oCVs[k];      // the end effector is reached

   // rotated vector from base of actual bone to jth bone
   MVector rbtjp = btjp.rotateBy( angleAndAxis );
   oCVs[j] = oCVs[k] + rbtjp;
  }

 }
}
```

Source Code 4: CCD algorithm

We chose the CCD method for solving an IK-system, as it does not have the problems around singular configurations as does the Jacobian transpose method. Furthermore, the method guarantees to converge [Wel93]. However, the CCD method has several restrictions that make it difficult to anticipate how many iteration steps promise the IK-system to converge. Due to the fact that we do not provide a time-dependent simulation environment to control the action line, the displacement of the end effector can vary remarkably. This means, the iteration steps can vary to achieve satisfactory results. Not only does the displacement of the end effector determine how many iteration steps are required to make the IK-system converge. The configuration of the whole chain rather influences the computation decisively.

Finally, we did not implement either of these methods for several reasons. At first glance, an IK-system applied on every segment of the action line might appear to be the right choice as it behaves correctly when displacing the end effector. However, it should be stated that the action line of a real muscle does not behave like a system of linked rigid segments. Moving the end effector changes the system from base to tip. It seems that through the whole chain and on every segment the same force is exerted. In fact, not every segment of the chain has the same applied force. Furthermore, every segment of the action line can vary its length. The action line attempts to balance the applied forces through every segment. Hence, if one segment is affected, this and the adjacent segments try to compensate the force. The entire chain is not automatically affected.

In the case of this IK-system, it is only possible to affect the tip of the chain. However, our system necessitates every segment being able to be influenced by external forces. A mass-spring system has less limitations and is therefore applied on the action line.

5.1.2 Final Realisation

First of all, to implement the action line, its requirements are defined. Since the behaviour of the action line is derived from the muscle properties, this necessitates taking a closer look at how a muscle works and how the action line is influenced:

Contraction: The task of a muscle is to contract. In case of the action line, this means that it contracts along itself towards the origin. In the case of a sphincter muscle, contraction is towards its centre.

Attachments: The muscle should be attached to the bone at the origin. The origin is the point of the action line that stays rigid. Since we are concerning facial muscles, the muscles do not have their insertion attached to the skull, except of muscles that govern the mandible.

Elasticity: Several forces act on the muscle. The muscle, for instance, can passively be stretched. In this case the muscle shows an elastic behaviour. Following the motion of the orbicularis oris (see Figure 6), the muscle contraction causes an elastic stretch in those structures connected to it. Therefore, it is required to be able to attach muscles to other muscles. Moreover, different muscles show different elasticity effects based on varying density and stiffness.

External Forces: The muscle can be affected by external forces induced by gravity, by motion of the entire head that causes the muscle to jiggle, and by collision with other muscles, bones, or skin. In addition to these forces, muscles can be influenced by forces external to skin.

Muscle Groups: Waters and Frisbie state that actions of intertwined muscles can be coupled [WF95]. The zygomaticus and the levator labii superioris (see Figure 6), for example, show this behaviour [Kae03].

In our software, we do not adopt every feature specified above as some of them are not automatically noticeable. In the following, it will turn out which features are compulsive and which ones can be omitted.

As alluded to above, we implement the action line based on a mass-spring system. The mass-spring system covers most of the principles governing how a muscle works like elasticity, external forces applied on the muscle, attachment to the bones, and the grouping of muscles.

5.1.2.1 Physically-Based Model

In this chapter, we expand on the model of the action line from a physical point of view. Commonly used physically-based models for muscles handle time-varying properties of the muscle parameters. Several researchers attempted to embed physical and dynamic properties into geometry. Our approach confines to model the action line and its physical behaviour. Our ap-

proach is utilising Elias' model [Eli03I] that presents a 1D-mass-spring system. To animate such a system, the explicit Euler integration scheme is commonly used as shown in Equation 36.

Elias [Eli03I] analyses this model and omits redundant parts. He describes the 1D-mass-spring system as a string, a line of points connected by springs. The springs are initialised accepting one value for a certain magnitude in rest position. Moving a point causes the magnitude to change. If this magnitude stretches during animation, the connected points experience a force towards each other. The force is applied inversely when points cause the springs to fall below their original length. Elias [Eli03I] does not comprise the mass of every particle in his approach. In the case of our action line, every particle has the same mass so that it can also be omitted. Another reason to omit the mass of each particle is that, at present, it does not have to swing. Hence, the aspect of an external force causing the muscle to jiggle is not considered at the moment.

All this system does is attempting to maintain the magnitude of springs in rest position. Translated into a geometrical point of view, the adjacent points try to maintain their distance from each other. The extensions of the springs attempt to balance by counter forces.

$$\vec{s}_{con} = x_{i-1} - x_i$$
$$l_{cur} = |\vec{s}_{con}|$$
$$d_{ext} = l_{cur} - l_{orig}$$

x_i : position of particle

$\vec{s}_{con}$: connected spring vector

l_{cur} : current magnitude of spring

l_{orig} : original magnitude of spring

d_{ext} : extension of spring

Equation 52: Extension and magnitude of spring

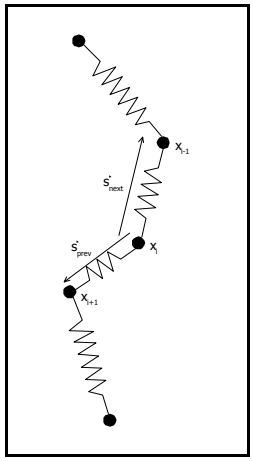


Figure 44: 1D-mass-spring system

Given the current magnitude and the resulting extension of every spring, each particle can be recalculated:

$$x_i^{new} = x_i^{curr} + \frac{\vec{s}_{con}^{\,prev} d_{ext}^{prev}}{2 l_{curr}^{prev}} + \frac{\vec{s}_{con}^{\,next} d_{ext}^{next}}{2 l_{curr}^{next}}$$

Equation 53: Compute next position

Every spring has the same stiffness and can therefore be regarded as equal. Iterating through the entire action line for every point on the basis of Equation 53, the system can be evaluated. It has to be mentioned that boundary points are treated differently. The force applied on a boundary point results from one adjacent point with which it is connected. In case of a sphincter muscle (cf. Chapter 2.3), the last and the first particle of the action line are connected by a spring.

Time-dependent, the system can now react on external forces. We claim the system to be interactively run, not in a time-dependent simulation environment. We therefore developed a mechanism that computes the entire mass-spring system in an iterative way so that it continuously is in a balanced state (compare the work of Wilhelms [Wil94]). It is then important to determine the interruption of the iteration. The system is situated in a balanced state when every spring has its original magnitude. Our system compares every present magnitude with the stored magnitude in rest position. When the original and the current spring magnitude diverge, the system is not balanced.

```
// lengthSimilar       -> boolean
// nCVs                -> point array of new control vertices
// currentSpring       -> vector
// originalSpring      -> vector
// MASS_SPRING_EPSILON -> Maximum divergence factor
lengthSimilar &= (abs(currentSpring[i].length() -
                  originalSpring[i].length())<MASS_SPRING_EPSILON);
```

Source Code 5: Balancing mass-spring system

When the current resemble the original spring magnitudes, the iteration is interrupted. Since this mass-spring system allows the springs to extend, the action line can be protracted so that the current spring magnitudes cause the maximum divergence factor (cf. Source Code 5) to be exceeded. However, the mass-spring system can be balanced. The mechanism described above does not interrupt the iteration. It has to be stated that the result improves by increasing the number of iterations. In fact, the system converges almost exponentially relative to the iteration steps. Our system allows the user to define the maximum iteration steps (see Figure 45). Benchmark tests proved that a variation of iteration steps in a range of 100 using 10 particles does not influence the performance noticeably. The value for iteration steps denoted as *"precision"* is by default set to 500 as we determined that this value is sufficient.

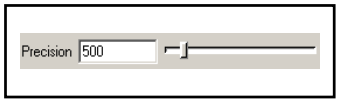


Figure 45: Iteration steps

The system is implemented in a deformer node (cf. Chapter 3.1.3, Chapter 5.1.2.4). The deformer enables the access on components controlling the action line. The action line is repre-

sented by a polyline, in other words a NURBS curve of degree 1 (cf. Chapter 4.2.3). The attempt to apply the mass-spring system on a cubic NURBS curve (cf. Chapter 4.2.3) failed. However, as the control vertices are not necessarily placed along the curve, the mass-spring cannot be used to simulate physical behaviour. Maya® exclusively allows access on the control vertices, not the edit points (cf. Chapter 4.2.3). Edit points are located along the cubic NURBS curve so we attempt to compute the control vertices by translating the edit points. The translation turned out to be impossible as Alias had a special computation method that they were not willing to release. Finally, we decided to use a workaround (cf. Chapter 5.1.2.3) that is based on a NURBS curve of degree 1 where edit points are equal to control vertices. In our workaround, we finally smooth the NURBS curve having the same effect as using a cubic NURBS curve from the initial point.

Given the access on control vertices, the computation is executed in two passes. In the first pass, the system is set up depending on whether being a sphincter or a parallel muscle. We introduce a matrix of objects each containing the required data to describe a particle of a mass-spring system. Each object contains data like the position the particle it is connected to and its spring magnitude to its connected particle. Next, a one-dimensional matrix containing these objects is initialised in the first pass. Thus, in the first pass the whole action line in its original rest position is captured and the mass-spring system is generated.

In the second pass, the deformed action line is evaluated based on the mass-spring system. At present, external forces and attachment points are taken into account. Jiggle, collision, or forces acting through the skin do not no longer influence the calculation as they should scarcely be visible. In Chapter 6.1, we will expand on these influence forces. First, for parallel muscles, one attachment is made. The origin that is represented by the first point of the action line is attached to the skull. This means that one static force is acting on the first particle of the action line that makes the particle stay rigid relative to the skull. This way, every particle of the action line can be driven by an external force. For example, the insertion can be attached to another muscle or to the lower jaw. Having two attachments made at both ends, both objects, the insertion and origin are attached to, can be relocated (see Figure 46). Likewise, other objects can force other particles of the action line. Assume for now, we have a sphincter muscle like the orbicularis oris (see Figure 6),

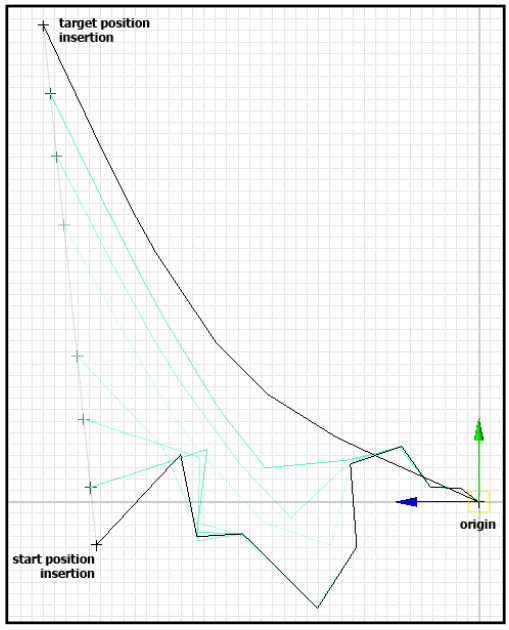


Figure 46: Action line, insertion driven by locator

connected to other muscles being responsible for expressing smile. How these connections are established is further commented on in Chapter 5.1.2.4. In the next section, we will provide a technique allowing the muscle to be contracted, the action line in particular.

5.1.2.2 Contraction

In this chapter, we describe how the contraction of the action line is computed. The contraction can be controlled by two means. One is the contraction factor that can be controlled by the user. The other way to control the contraction is based on the mass-spring system. Consider the insertion moves towards the origin. The muscle will be contracted along itself.

The system stores the original shape of the action line in rest position. After having the positions of the control vertices stored, the single segments (*"spans"*) are computed. Each segment length (cf. Equation 54) and each segment's accumulated segment lengths (cf. Equation 54) are defined:

$$l_i = \|x_i - x_{i-1}\|$$
$$l_i^{acc} = l_i - \sum_{j=0}^{i-1} l_j$$
$$l_i : \text{length of span}$$
$$l_i^{acc} : \text{accumulated length}$$

Equation 54: Span parameterised

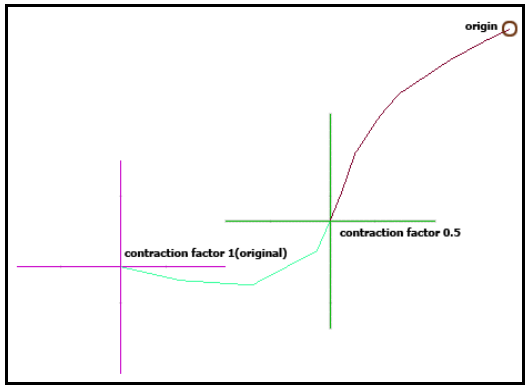


Figure 47: Contraction along action line

The accumulated length of the last span of the action line is equal to the length of the whole action line. With the information described above, multiplying by the contraction factor enables the new accumulated span lengths to be determined. Based on the new accumulated span lengths, the parameterised new position of every control vertex on the action line can be calculated.

Given the control vertex of the insertion, contracting the action line means that the insertion will be translated so that it is finally placed somewhere along the curve on an arbitrary span. Comparing the new accumulated span length of the insertion with all original accumulated span lengths, provides the information on which span the insertion must now be placed. The new accumulated span length of the insertion is now subtracted from the smaller accumulated span length of its corresponding control vertex. The received rest length is divided by the corresponding original span length. The length proportion obtained by this division serves to deter-

mine the new vector length from the corresponding control vertex to its adjacent control vertex. By adding this vector on the vector of the corresponding control vertex, the new position of the insertion can be defined.

Iterating through every control vertex based on the contraction factor, the new action line is computed (see Figure 47). Our system is not limited to have the action line exclusively contracted by specifying the contraction factor. Hence, if the insertion is driven by another component and is moving towards the origin, the action line is contracted as described above. To make the system shortening the action line, the elongation from insertion to origin is determined. The original elongation in rest position is also computed. If the new elongation is falling below the original elongation, the action line is contracted as exemplified above. The contraction factor is therefore defined by the proportion of the new to the original elongation.

Our system allows the contraction to be executed in two directions. The action line can be contracted towards the origin, as it is assumed by default, or towards the insertion. Hence, the action line deformer has an additional attribute: the *"contraction type"*. The calculation of the contraction is adapted depending on the direction of iteration. Changing the contraction type causes the iteration to begin with the last control vertex.

Our system additionally allows the user to contract a sphincter muscle. As stated earlier, the sphincter muscle contracts towards its centre. The centre of contraction is defined by an additional object whose position and orientation in world space serve as input for the action line deformer.

```
// oCVs                -> point array of original control vertices
// nCVs                -> point array of new control vertices
// contrFactor         -> contraction factor
// centre              -> centre of contraction
for(unsigned i=0;i<oCVs.length();++i) {
 MVector v = oCVs[i] - centre;         // vector from cv to centre
 nCVs[i] = centre + v * contrFactor;   // new cv
}
```

Source Code 6: Contraction of sphincter muscle

As shown in Source Code 6, the contraction is applied on every control vertex.

The calculation can be reversed. By changing the *"contraction type"*, the action line can be evenly displaced away from the centre. The experience proved that in the majority of cases, changing the contraction type of the circular action line does not make sense as the stretching is caused by other forces.

To distinguish between having a parallel or a sphincter muscle, our system examines the action line on being closed. In Maya® it is differentiated between three kinds of NURBS curves.

The NURBS curve can be open, closed, or periodic. If Maya® claims the NURBS curve to be open, having a parallel muscle is assumed. The application is interrupted when having a closed curve. A closed curve is a loop with coinciding start and end points [Ali04a] not necessarily continuous in this area. The third case, the periodic one, represents the correct kind of NURBS curve to define a circular action line.

The action line deformer is now capable of coping the basic tasks of an action line. At the moment, other features like collision detection and jiggling are not finally implemented. That is why they are not further elaborated on in this chapter. In the next section, we will discuss how to get a smooth cubic NURBS curve without having to deform its control vertices. Due to the fact that it is not possible to control edit points (cf. Chapter 4.2.3.2), we use a workaround that allows us indirect access on them.

5.1.2.3 Smoothing the Curve

In this chapter, we expand on making a NURBS curve of degree 1 smooth and round. The result will be a cubic NURBS curve that is attempting to approximate the NURBS curve embodied by straight segments.

The Maya® node (cf. Chapter 3.1.2) that provides this feature is called *"fitBspline"*. *"This node computes a NURBS curve that will fit through a list of given points. The list of points is defined as the control points of the input curve."* [Ali04a]

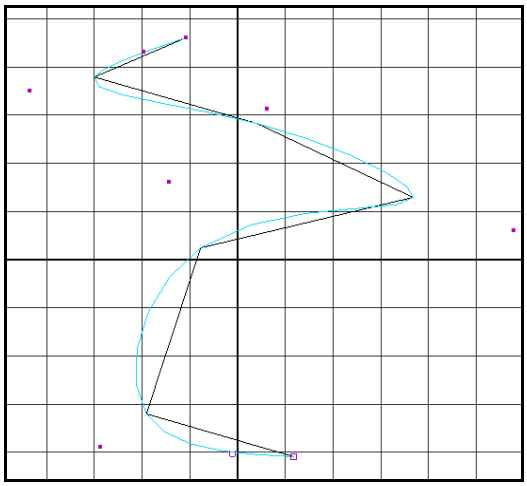


Figure 48: Smoothing with fitBspline

In Figure 48, it can be noticed that the blue cubic NURBS curve attempts to run through the control vertices that are lying on the NURBS curve of degree 1. This way, it is possible to control the control vertices of the NURBS curve of degree 1 that, as depicted in Figure 48, also lie on the cubic NURBS curve. Having a mass-spring system influencing the curve as previously

alluded to, the components specifying the curve are located along the curve, namely the edit points (cf. Chapter 4.2.3.2 and Chapter 5.1.2.1). Thus, using this model allows us to control a smooth cubic NURBS curve, by translating points placed along the curve. In Chapter 5.2, it will turn out why using a smooth action line makes sense.

5.1.2.4 Architecture

In order to understand the framework of the implementation of the action line, the basic software architecture will be an issue. As might have become evident, we are using the term *"action line deformer"*. In Chapter 3.1.3, we already mentioned the **`MPxDeformerNode`** which is responsible for deforming an input geometry and consequentially generating an output geometry. In Chapter 4.2.2.2, we specify how polygonal meshes are deformed.

Figure 49 depicts how the deformer works in terms of its object-oriented architecture.

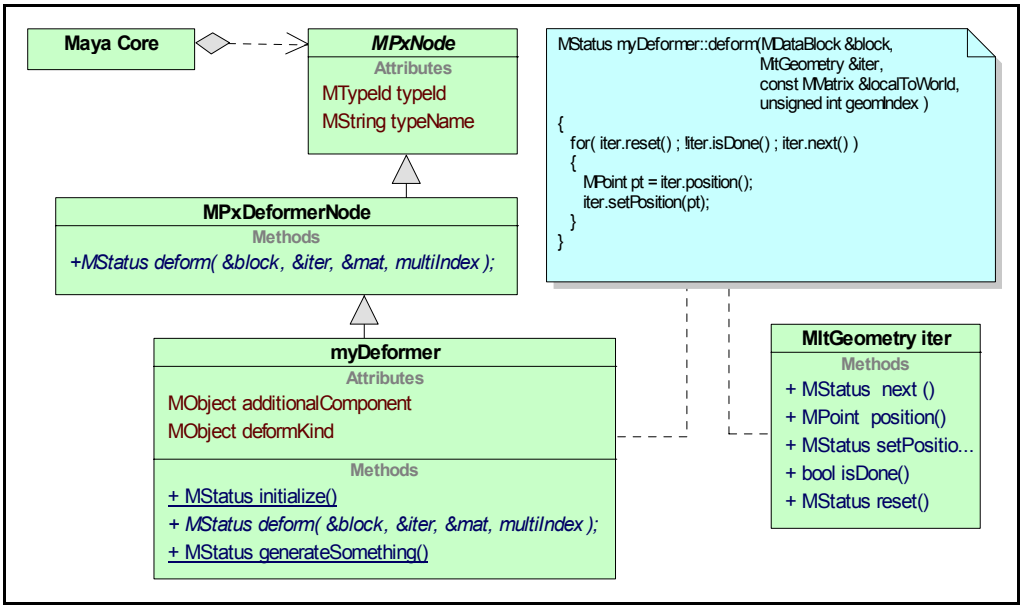


Figure 49: Architecture of general deformer in UML

The Maya® core heads the hierarchy. As described in Chapter 3.1.3, internally, a copy of the node that the user creates is used. This copy is part of the dependency graph. The user only accesses its proxy object **`MPxNode`** (see Figure 49). The important object to deform a geometry is the **`MPxDeformerNode`**. This object inherits from **`MPxNode`** including attributes that provide the node with a unique identification number and a name. Finally, we specify the **`myDeformer`** object that inherits the **`MPxDeformerNode`** and therefore the **`deform`**-method.

As shown in the bluish box in Figure 49, the **`deform`**-method is finally executed induced by the Maya® core. It obtains the following parameters necessary to deform the input object: the

data block (containing all necessary input and output data), an object to iterate over the input geometry, a transformation matrix to translate from local to world space, and the geometry index corresponding to the requested geometry when having multiple outputs.

Notice the green box at the bottom right in Figure 49. Given a NURBS curve as input, the handle on this geometry is provided by the iterator object (`MItGeometry`). This object enables to iterate over every control component of the geometry, in this case the control vertices of the action line. It allows us to read and to set the position of the control vertex and thus the manipulation of the entire action line.

To give the user control over the action line, additional attributes and input plugs are specified. By modifying attributes, the user is allowed to control additional parameters influencing the action line. As exemplified in Chapter 5.1.2.1 to 5.1.2.2, the action line can be influenced by the attributes *"precision"*, *"contraction factor"*, and *"contraction type"*.

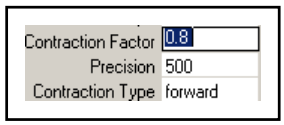


Figure 50: Attributes action line

To allow the user to control every control vertex and thus every particle of the mass-spring system applied on the action line, we provide an array of plugs to provide the input of a set of transform matrices. The user for example, is able to connect the world matrix of a locator object to the first element of the array of input plugs. This means the first control vertex of the action line is driven by the position of the locator:

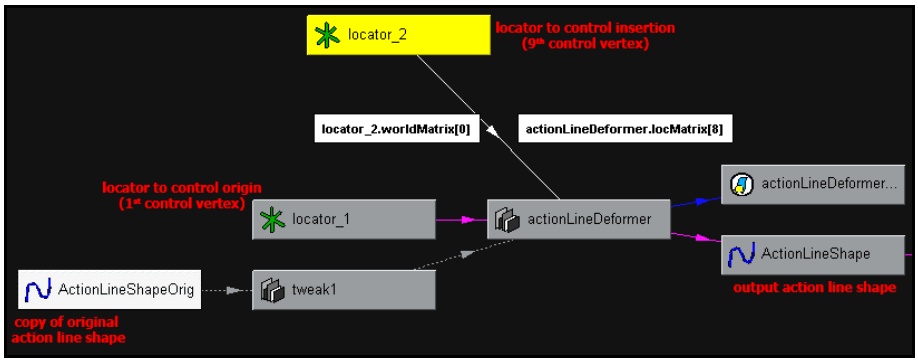


Figure 51: Input and output plugs of action line deformer

When applying a deformer, Maya® duplicates the original input object and uses a copy as input object. Finally, the object the deformer is applied on serves as output object (see Figure 51). The deformer is created using a MEL script that looks like the following:

```
// create action line deformer
$alDefName = `deformer -type actionLineDeformer -name "actionLineDeformer"`;

// attach action line shape as input geometry
```

```
deformer -e -g actionLineShape $alDefName[0];

// connect world matrix of locator_1 to the origin of the action line
connectAttr -f ("locator_1.worldMatrix") ($alDefName[0]+".locMatrix[0]");
```

Source Code 7: Set up action line deformer

Consider the action line has a number of n control vertices. The deformer generates an array of input plugs with the count of n. In addition to this array of plugs, the deformer provides an input plug to define the centre position of a sphincter muscle. This may also be the world matrix of a locator that is connected just like the drivers of the control vertices.

It must be concluded that this kind of architecture allows us the freedom to expand the system. Additional attributes and plugs can later be appended. We decided to write our own node for the action line because of the modularity it provides. The action line can be multiplied adopting different properties. The action line can be seen as one module that is independent from any other component controlling the facial animation.

5.2 Muscle

To animate facial expressions, the muscle plays a crucial role. The action line, the foundation for the muscle's animation, is already accomplished. In this section, we specify how the action line affects the deformation of the muscle. Moreover, we describe how the muscle attempts to preserve its volume based on the action line varying its length. The user will be able to modify the bulge caused by the contracted muscle. Furthermore, it will finally be possible to adjust the shape of the muscle bulge.

In the following chapter, we will elaborate the experiments we conducted that lead to the final realisation of the action line driven muscle shape.

5.2.1 Experiments

The first approach we use that provides a model to create a muscle shape is derived from the work of Kähler [Kae03]. He utilises ellipsoids to simulate segments of muscle fibres (cf. Chapter 4.4.7.2). As alluded to above, the ellipsoid model lacks accuracy concerning the continuity of the shape's surface. This model could be used to synthesise facial animation being scarce in terms of appearance. Given the demand for a realistic simulation, this model does not suffice in terms of appearance. First, we extend Kähler's model similar to the model described in Chapter 4.4.3.

5.2.1.1 Cylindrical Muscle Node

This model aims at generating a deformed cylinder surrounding the action line. Imagine for now, stepping through the action line with a certain step size. Around each point, a discretised ellipse is drawn. Depending on the local coordinate system of the corresponding point of the action line, the ellipse receives its orientation. The local coordinate system is obtained by determining the tangents. We add an arbitrary vector to receive a vector orthogonal to the tangent. This is done by computing the cross product. With the orthogonal vector, we calculate a third vector that defines the last of the three coordinate axes. With the local coordinate system in place, we can fill a transformation matrix that is multiplied on the ellipse. This way, the ellipse obtains its orientation. Having every ellipse computed, the ellipses are connected to create the polygons that make up the final model of the deformed cylinder.

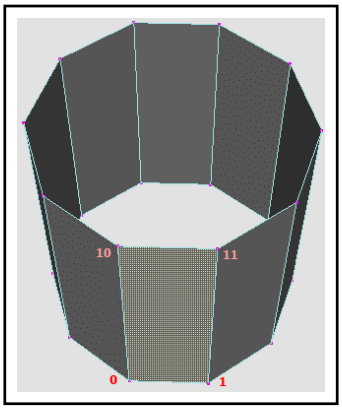


Figure 52: Cylinder setup

To model the cylinder as a polygonal mesh, the positions of the vertices are consecutively stored in an array. Next, the degree of every polygon is written into another array. The degree defines the number of vertices that specify this polygon. To generate the polygonal mesh, in the last array, the vertex indices of each polygon are stored depending on its corresponding degree. Unlike the action line we implemented, the muscle shape generator is an object that inherits from **MPxNode**. This node generates the geometry in its compute method (cf. Chapter 3.1.1 and 3.1.3). In the following, a small excerpt of the geometry generation is provided:

```
// subDivCap            -> sub divisions of every ellipse
// newCSMatrix          -> matrix for new coordinate system, depend. on action line

int index2To1Dim = row*subDivCap;                 // transl. from 2D to 1D array
for(t=.0, i=0;i<subDivCap;t+=tStep, ++i) {

 x = heightEllipse[row]*cos(t);                   // simple ellipse drawn in 2D
 y = widthEllipse[row]*sin(t);                    // every height and width
 z = 0;                                           // can be specified
 MPoint pMuscleNoOrient(x, y, z, 1.0);            // ellipse point in 2D
 MPoint pMuscle = pMuscleNoOrient * newCSMatrix;  // orientation and position

 // set up array of vertex positions
 vertices.append(pMuscle.x, pMuscle.y, pMuscle.z, 1.0);

 // set up array of vertex count for every polygon
 faceDegrees.append(4);
```

```
 // connecting vertices for polygons / set up vertex index array for every polyg.
 faceVertices.append(index2To1Dim-subDivCap+i);
 faceVertices.append(index2To1Dim-subDivCap+i+1);
 faceVertices.append(index2To1Dim+i+1);
 faceVertices.append(index2To1Dim+i);

}
```

Source Code 8: Setup cylindrical mesh

As shown in Source Code 8, the dimensions of every ellipsoidal cross section can be modified. By varying the cross sections, not only the muscle shape can be modified, but the muscle bulge can be generated. The height and width is multiplied with a factor varying along the action line. Near the tendons, the cross sections should not experience a change. In the centre of the muscle belly, the bulge should appear with the largest extent. In case of this experiment, we decided to use a flipped square function to simulate the muscle bulge. The function produces a curve that should approximate a real muscle bulge. The discretised values of this curve are multiplied to the height and width of each ellipsoidal cross section along the muscle. Varying the parameters of the square function changes the appearance of the muscle bulge. Furthermore, the system allows us to control the bulge direction, i.e. the extent of the affected height may exceed the extent of the affected width of the ellipses.

At first sight, this model seems to fulfil all requirements to synthesise a real muscle. The possibility to change the shape as described above should allow to approximate nearly every muscle shape.

However, the model lacks simplicity to handle and authenticity of appearance. To produce a realistic muscle as it should be the case in our simulation too many modifications have to be made. It is not convenient enough to generate a muscle shape by changing every cross section. If we are dealing with a flat muscle like the ones that can be found in the scalp area such as the frontalis (see Figure 6), this model cannot generate appropriate results.

In addition to that, the square function we used to simulate the muscle bulge does not approach a real muscle. It can be observed that the bulge increases its extent slowly from tendon to the centre of the muscle belly. Nearby the centre the muscle model should maintain its belly-like shape. In the next section, we follow these requirements. We provide a model that accepts any kind of polygonal mesh for the muscle shape.

5.2.2 Final Realisation

In this chapter, we discuss the final implementation of the muscle deformer based on the action line. Having experimented with the model previously introduced, we adopt and adjust some of the features it provides. The deformation of the muscle shape, as it is described in the previous chapter, can be conveyed on the following model. Recall having each 2D ellipse to be re-oriented and repositioned depending on the action line. Now, we transfer this model to a completely different three-dimensional shape that is independent of being based on a primitive like the cylinder.

5.2.2.1 The Muscle Shape Driven by the Action Line

Consider a Maya® deformer accepts the action line as additional input. After having the action line converted to a cubic NURBS curve, it can serve as additional input for the muscle deformer. Thus, the deformer can be applied on an arbitrary polygonal mesh. To have the action line controlling the muscle shape, we regard the action line as a chain of joints. This skeleton drives the deformation of the muscle shape. Maya® allows us to access the NURBS curve and read the positions of the points belonging to it. Stepping along the action line, the obtained points serve as joints to cause the muscle shape to deform. In an analogous manner, the muscle geometry is smooth bound (cf. Chapter 4.3.3.1 or 4.4.7.1) to the points of the action line. We adopted an approach by Alias [Ali03b] that released a description of the *"internal math for the smooth skinning interpolation"*. In the following, we describe the algorithm derived from Alias [Ali03b] with some adjustments.

Given a set of vertices, shaping the muscle geometry, each is repositioned according to the influence points of the deforming action line. First, we compute the local coordinate system of each influence point on the action line (in the following referred to as *"influence"*):

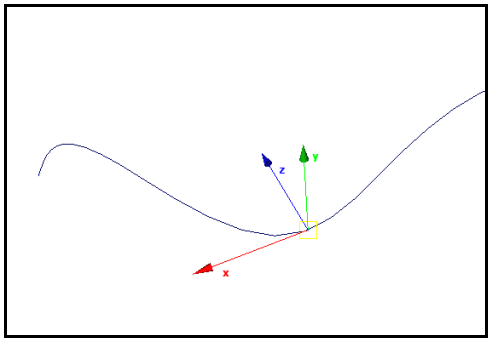


Figure 53: Local coordinate system of influence point

$$T_i = \begin{pmatrix} xAxis_x & yAxis_x & zAxis_x & cv_x \\ xAxis_y & yAxis_y & zAxis_y & cv_y \\ xAxis_z & yAxis_z & zAxis_z & cv_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

cv : position of control vertex

$xAxis$: x axis vector of local coordinate system(tangent)

Equation 55: Local coordinate system

Basically, the local coordinate system obtains its orientation from the tangent of the action line at the point of the particular influence (see Figure 53, x-axis, red). We provide two ways of computing the other axes. The first one serves to produce the z-axis (see Figure 53, blue) by applying the cross product to an arbitrary vector. This vector may not be parallel to the x-axis. The second method serves to avail the vector by disassembling the bias matrix that is another input to the muscle deformer node. The function of the bias matrix is discussed further down. Disassembling the matrix means that its y-axis is extracted and that it is used to compute the orthogonal z-axis (see Figure 53, blue) of the influence point's local coordinate system. Having identified the z-axis, the y-axis can be computed. The cross product of the x- and z-axis supplies us with the y-axis and thus rounds off the local coordinate system of the particular influence. See Equation 55 where the setup of the transformation matrix of the local coordinate system is illustrated. The initial transformation matrix of the influence point (B_i) serves as basis for further calculations of every vertex of the muscle shape. With this matrix and the current transformation matrix of the influence (Γ_i), the current position of every vertex can be computed:

$$L_i(p) = B_i^{-1} * W(p) \qquad (1)$$
$$N_i(p) = T_i * L_i(p) \qquad (2)$$
$$M_i(p) = w_i * N_i(p) \qquad (3)$$
$$NewWorldPos(p) = M_1(p) + M_2(p) \qquad (4)$$

$B_i(p)$: world transform matrix of influence at time of bind
$W(p)$: world position of P at time of bind
T_i : current world transform matrix of influence i
w_i : weight of vertex for according influence

Equation 56: Smooth bind algorithm

As shown in Equation 56, M_i denotes the transformation matrix of every influence for a particular vertex. Every vertex is affected by all influences depending on weights. If an influence is the weight of 1 assigned to, it represents the only influence that affects the vertex. The weights of all influences on a vertex sum up to 1. As it might be noticeable in Equation 56, only the maximum of two influences are assigned to each vertex. Due to the fact that the action line can be seen as a nearly straight chain of joints surrounded by the muscle shape, every vertex of the muscle shape is located not exceeding the influence of two joints. Maya® provides a function that serves to ascertain the closest point on the action line to a vertex (see Figure 54, grey line). The closest point is obtained parameterised and depending on the action line. Given an action line with a number of n control vertices and m points lying on the action line and being influence

points, we can obtain weights of an influence on a muscle shape vertex by applying the following equation:

$$w_1 = \left\lceil \frac{m * u}{n} \right\rceil$$
$$w_2 = \left\lfloor \frac{m * u}{n} \right\rfloor$$

$u \in [0, n]$: parameter of closest point on action line
m: count of influences
n: count of control vertices($\Rightarrow$ highest parameter)
$\lceil x \rceil$: ceil to next integer
$\lfloor x \rfloor$: floor to next integer

Equation 57: Weights for two influences

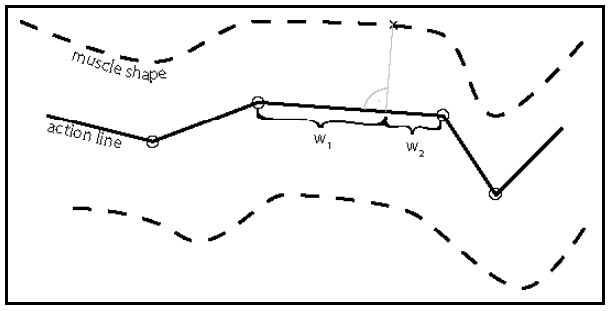


Figure 54: Weights by closest point

See Figure 54 where two influence points affect a muscle shape vertex depicted as a cross. Depending on w_1 and w_2, Equation 56 is applied for every vertex of the shape. Given a muscle shape vertex whose closest point on the action line is one of the end points, this end point exclusively influences this vertex with the weight of 1. If we are dealing with a sphincter muscle, the action line does not have end points so every point is throughout affected by two influence points. Figure 55 shows an example of the muscle deformer. Each of the three images displays the action line that has a fixed point on the right: the origin.

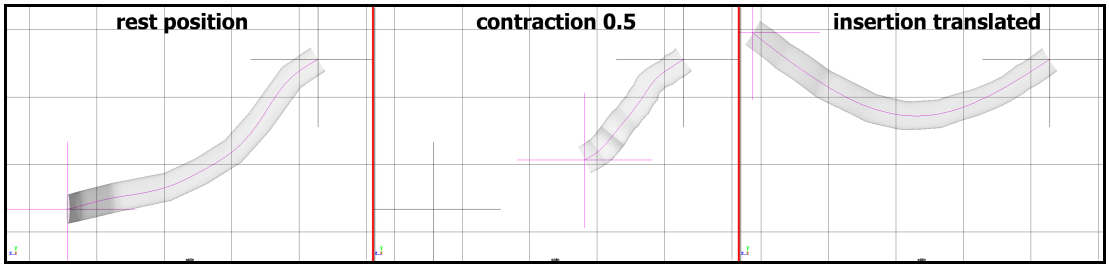


Figure 55: Muscle deformer

The pink locator on every illustration depicts the insertion that can be freely translated (see Figure 55, right) and that moves with the action line (see Figure 55, pink curve) when contracted (see Figure 55, centre).

At the moment, the muscle is functionally attached to the action line. It can be stretched, contracted, and inherits all other remaining properties of the action line. Another missing, important feature of the muscle is its volume maintenance and the muscle bulge the contraction causes. In the following chapter, we will detail the simulation of muscle bulge.

5.2.2.2 Muscle Bulge

Now that the muscle shape adequately deforms according to the action line, the muscle bulge can be modelled and parameterised. Having the muscle shape and its approximated volume stored in rest position, our system attempts to preserve the volume. As previously mentioned, the volume preservation of a muscle is not biologically proved [WG97]. However, the muscle attempts to approximate a muscle preservation while contracted. To simulate muscle preservation, we assume the shape of the muscle to behave like a deformed cylinder (cf. Chapter 4.4.7.1). The cylinder volume is computed by multiplying the cross section with its height.

We divide the muscle shape in sections. Each section embodies an imaginary single cylinder. The length of its corresponding action line segment defines the height of this cylinder. The area of the cross section is approximated by using the formula to calculate the area of a circle:

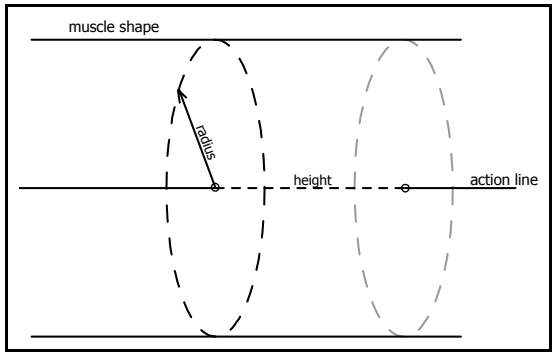


Figure 56: Cylinder section approximation

Figure 56 illustrates the approximated cylinder section. The volume of the cylinder is calculated as follows:

$$V = \pi * radius^2 * height$$

$$radius = \sqrt{\frac{V}{\pi * height}}$$

Equation 58: Cylinder volume / radius

The muscle bulge can be computed by utilising the cylinder model. Our system computes the closest point on the action line to every muscle vertex. The distance from vertex to its closest point defines the radius of the approximated cylinder. Thus, the cross section of each cylinder section is considered to be composed of several radii.

Similar to the action line deformer, the muscle shape deformer is executed in several passes. In the first pass, all cylinder section heights are stored. The cylinder section height is defined by

the segment between adjacent influences on the action line. For every vertex, its corresponding segment correlation is stored. Now, the formula to calculate the radius can be applied. Varying the height of the cylinder changes the radius depending on the square root of the height's reciprocal. Associating the original and current segment length (equivalent of cylinder height) creates the following equation that is similar to the formula provided by Wilhelms and Gelder [WG97]:

$$stretchFactor = \sqrt{\frac{l_{seg}^{original}}{l_{seg}^{current}}}$$

Equation 59: Factor, defining how far vertex is pushed away from action line

This stretch factor describes how far the vertex is displaced away from the action line. However, it does not guarantee exact volume preservation as we are not concerning muscle sections being entirely equal to cylinders.

The muscle does not behave evenly along its elongation. Muscle fibres taper near the tendons. Fibres of the muscle tend to shorten nearby their centre. Fibres in the side push outer lying fibres away (cf. Chapter 2.3.7). These properties are responsible for the muscle to bulge: the muscle extent increases approaching the centre and decreases running towards the tendons. The muscle thickens smoothly from tendon to tendon; the appearance resembles a smooth curve progression. In Chapter 5.2.1.1, we experiment with a square function to control the swell along the muscle. As we previously pointed out, the square function does not comply with the requirements of an authentic muscle bulge.

We use a Gauss-function instead and conclude that it best fits the curvature of a real muscle bulge:

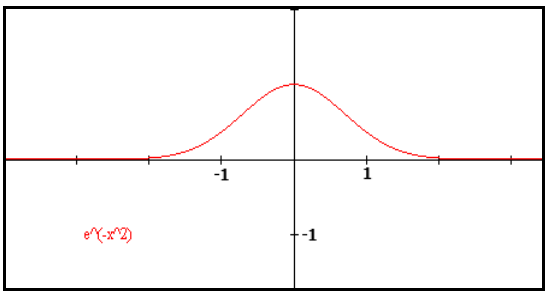


Figure 57: Gauss function

$$f(x) = e^{-x^2}$$

Equation 60: Gauss function

On order to give the user more control on the muscle bulge in addition to the volume preservation being approximately mathematically correct, the Gauss-function has several parameters:

$$f(x) = 1 + amp * 2^{-falloff*(x+offset)^{bf}}$$
$$x \in [0, lengthOfStretchFactorArray]$$
amp : bulge factor/amplitude
$falloff$: shape of peak
$offset$: displacement in x - direction
bf : bulge form($\in 1,2,4,6,...$)

Equation 61: Adapted Gauss function

With additional parameters, the user is allowed to further determine the shape of the muscle bulge. It must be mentioned that from now on, mathematical accuracy of the volume preservation is not guaranteed.

Equation 61 shows the adapted Gauss-function. This Gauss-function is finally applied on all stretch factors of each section previously computed. The Gauss function is further adapted in a way so that near the tendons, the volume of the muscle does not change. The offset can be additionally determined so that an assigned value of 0 causes the bulge to appear in the centre of the muscle. We modified the bulge form factor so that it accepts all positive integer values.

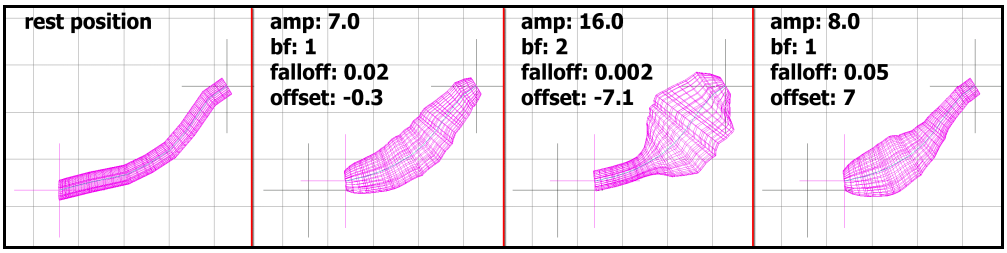


Figure 58: Muscle bulge with different parameters

In Figure 58, a contracting muscle is illustrated. We refer to a parallel muscle as this kind of muscle commonly occurs in the facial region.

If we are concerned with a flat muscle appearing in the scalp area for example, computing the muscle bulge evenly all around muscle may not be realistic. We therefore incorporate a function to suppress the muscle to bulge in every direction. The system has an additional input, the bias matrix, that determines the orientation of the muscle bulge. The system allows us to use one of the axes provided by the bias matrix to represent an up-vector of the muscle bulge. The muscle bulge in direction of the two remaining axes is reduced, the more the axes are approached. Recall the computation of the local coordinate system of the influence point. We utilised the bias matrix and the tangent as starting point for the calculation (cf. Chapter 5.2.2.1). Each influence point has an orientation referring to the bias matrix. The system then enables the

definition of the muscle bulge's up-vector. The additional attribute *"bias type"* allows the user to pick one of the three axes of the current transformation matrix (cf. T_i in Equation 56) of each influence (remind Chapter 5.2.2.1, Figure 53). Finally, the stretch factor is controlled by the angle that is defined by the current transformation matrix T_i of the selected axis to the vector pointing from the muscle vertex to its closest point on the action line. With decreasing angle the extent of the muscle bulge grows. If the angle approaches a degree of 180, the muscle bulge swells. Approaching the degree of 0 or 180 degree, it reaches its saturation point. A vector is temporarily computed to hold the chosen axis of the current transformation matrix. This vector is referred to as bias vector. It additionally depends on the two bias vectors which are defined by the influences. Based on the weights of each influence point on the according vertex (see Figure 54 and Equation 57), the corresponding bias vector is averaged.

```
// alToP            -> vertex to closest point on action line
// t2DivPI          -> 2 divided by PI
// sf               -> previously computed stretch factor
// biasVector depending on the bias vector of each influence and its weights
MVector biasVector = weight1*biasVectorInf1 + weight2*biasVectorInf2;

// previously added a value of 1, so it's always above 1. now subtracting to
// have the Gauss function symmetrical around the origin of (0,0)
sf = sf - 1;

// the closer the angle of alToP to the bias vector to 0 or 180(here in radian
// measure), the bigger the extent of the muscle bulge(stretch factor)
sf *= (float)(1 - abs(alToP.angle(biasVector)*t2DivPI-1));

// add 1 to have the Gauss-function always exceeding 1
sf += 1;
```

Source Code 9: Computing muscle bulge with orientation

If no bias matrix is specified, the muscle bulge is assumed to have the same extent evenly around the action line.

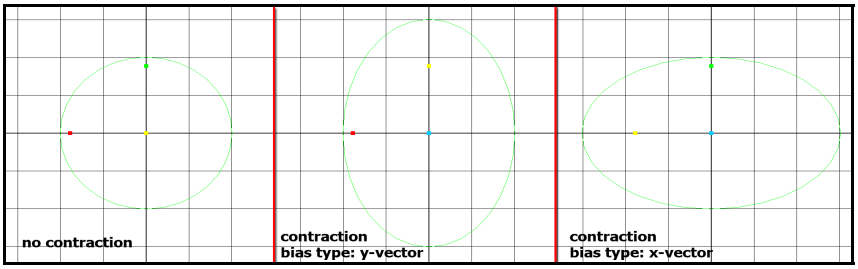


Figure 59 : Cross section of muscle, different bias types

In the previous illustration, a cross section of the muscle is depicted: the first one is not contracted; the others are contracted based on different bias types.

Given a sphincter muscle, it does not maintain its volume appearing with a bulge. The volume preservation is rather evenly spread along the action line. Thus, it is not possible to further specify the shape of the bulge caused by the volume preservation.

In addition to the bulge caused by the contraction, the system enables the user to control the isometric tension (cf. Chapter 2.3.7). The user is enabled to generate a muscle bulge separated from the action line's contraction factor. In Chapter 2.3.7, the isometric tension is explained. It denotes the tension while the muscle is in rest position. This attribute can be used to predefine a certain default muscle bulge before the muscle is contracted.

In the next section we discuss the architecture of the muscle deformer. We will exclude some of the background previously introduced.

5.2.2.3 Architecture

In this chapter, we elaborate on the architecture of the muscle shape deformer. Basic knowledge about Maya® deformers in general is prerequisite. In Chapter 5.1.2.4, the basic structure of a deformer is already described. In case of the muscle shape deformer, several adjustments are made.

This time, we are concerned with a polygonal mesh. The polygonal mesh consists of a set of polygons, defined by vertices. If a deformer is applied on the polygonal mesh of the muscle shape, the creator of the deformer is provided with a handle on the mesh vertices. Recall Chapter 5.1.2.4, where the deformer of the action line provides an iterator on the control vertices of the action line. This time, the deformer provides an iterator object to step through the list of vertices defining a polygonal mesh. Thus, it is possible to manipulate the shape of the muscle. Several attributes are introduced to further specify the muscle deformer. All attributes and their functions are described in the previous section. It has to be noted that the bias matrix is an attribute that is not writable, thus it is exclusively defined by its input connection. The same can be noticed considering the input plug for the action line.

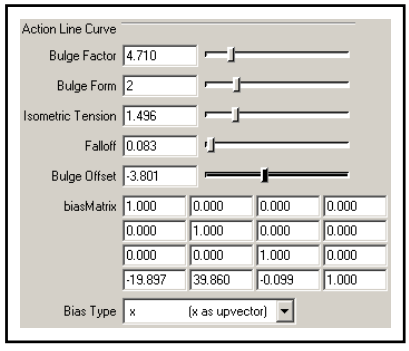


Figure 60: Muscle deformer attributes

In Chapter 5.1.2.4, an excerpt of the MEL script is shown that generates the dependencies of the action line deformer node. The MEL script creating the muscle deformer resembles the one of the action line deformer apart from its input connections. Following structure is created:

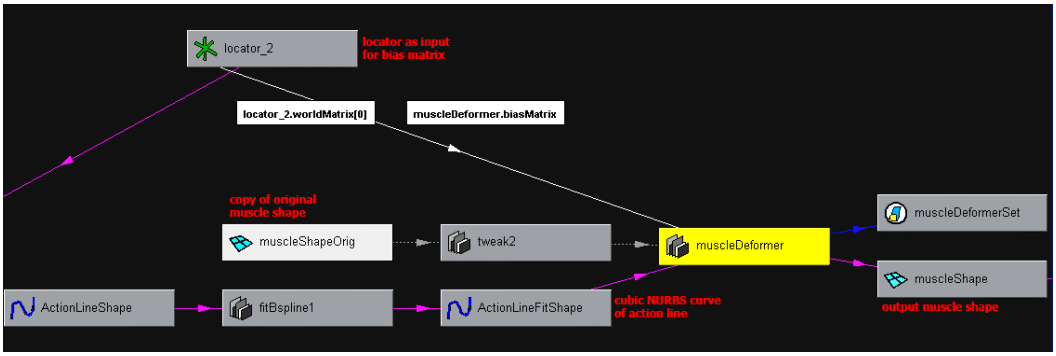


Figure 61: Input and output plugs of muscle deformer

The usage of a single node to deform the muscle is the foundation for duplicating muscles. Thus, the entire muscular structure of the face can be set up. In the next section we discuss how the skin shape is governed by the muscle shape. It will become clear that increasing the number of vertices defining the shape of the muscle improves the result. Therefore, it is recommended to use a smooth shape.

5.3 The Skin

In this chapter, the deformation of skin is made the main subject of discussion. The section concerning experiments about the skin deformer is quite short as we achieved satisfying results with the second model we implemented.

Recently, every endeavour has been made to deform the skin driven by an external component. We previously discussed the rigid bind method that allows a skeleton to control the skin deformation. This method does not include a three-dimensional voluminous object to drive the skin deformation. It is more likely to have joints that are parameterised by their local coordinate system and that control the deformation.

In case of a muscle-based skin deformer, we have to pay attention to the whole voluminous muscle geometry. In Chapter 4.5, we already commented on some approaches concerning muscle-driven skin deformation. The model of Aubel (cf. Chapter 4.5.1) appears to produce results close to human anatomy. However, Aubel's model lacks fast computation.

In the following, we expand on an experiment we conducted to approach realistic skin deformation without any loss of speed in terms of computation.

5.3.1 Experiments

5.3.1.1 Wrap Deformer

First of all, we experimented with the wrap deformer provided by Maya®. Since the source code of this wrap deformer is proprietary, only the function of this deformer can be described.

The wrap deformer allows us to control the surface of an object by a NURBS surface, a NURBS curve, or a polygonal mesh. These influence objects deform the driven object as soon as they vary their position, orientation, or shape. *"When you create a wrap influence object, Maya makes a copy of the influence object and uses it as base shape for the deformation. A wrap deformer can include one or more influence objects."* [Ali04a]

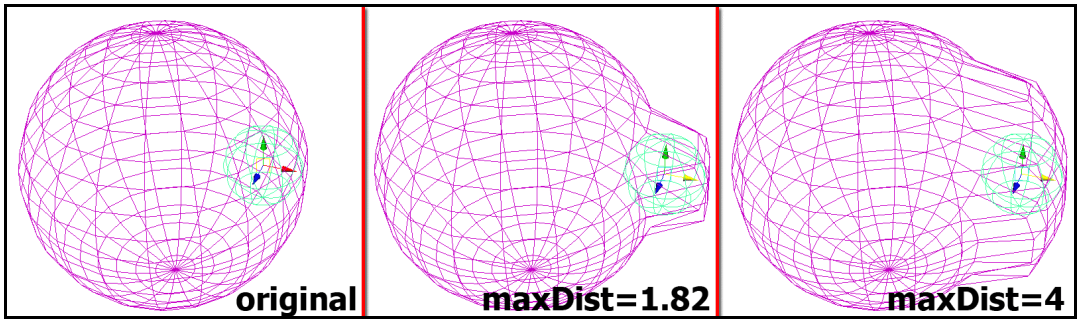


Figure 62: Wrap deformer, NURBS sphere influences polygonal sphere

See Figure 62 where a NURBS sphere influences a polygonal sphere. The left picture illustrates the two objects when the deformer is applied the first time. The wrap deformer provides the functionality to determine a maximum distance attribute that represents the threshold. Points exceeding this distance threshold are not influenced [Ali04a].

Consider the influence object being represented by a muscle shape. The skin of the face is the surface to deform. What is missing? The muscle deforms the skin depending on its initial position. The points inside the maximum distance range of every vertex are influenced evenly. However, real skin shows different behaviour patterns. Since we are considering visco-elastic material (cf. Chapter 2.3.7), the skin vertices are not tightly attached to the muscle. Besides that, the influence the muscle exerts to the skin is not exclusively defined by the distance between both but by the consistence of their materials. To allow the user determining the influence on every skin vertex, he is allowed to define the maximum influence. This is achieved by painting weights which is not provided by Maya® for this deformer.

Since the user is not able to access the interior of this deformer node, we decided to write our own skin deformer which is similar to the wrap deformer but provides some extra features.

5.3.1.2 Sculpt Deformer

The second deformer we experimented with is called the *"sculpt deformer"*. First, we require a NURBS sphere as influence object and a polygonal mesh as object to be deformed. It may be assumed that the sculpt deformer functions like the wrap deformer. In contrast to the wrap deformer, the sculpt deformer has not a set of points assigned to that depend on the maximum

distance threshold. Hence, the influence of the sculpt deformer is able to deform each point of the influenced object.

The sculpt deformer differentiates between three kinds of sculpting:

- **Flip mode**
- **Project mode**
- **Stretch mode**

However, in flip mode, only an implicit locator is provided to deform the surface. This locator serves to push away every vertex of the driven object until the centre of the locator passes the surface. The deformed surface then flips to the backside of the locator.

In case of being in project mode, the surface of the driven object is deformed by projecting the influence object on it. This means the surface of the driven object is deformed to appear like the influence object while being in its range of influence.

In stretch mode, the sculpt deformer pushes the vertices away. We assume the sculpt deformer to compute the distance of every vertex of the driven object to its influence object. If this distance falls below a certain value, the vertex is pushed away. Unlike being in flip mode, the sculpt deformer prevents the driven object to penetrate the influence object at any time. This way, we can assume having a certain gap assured between the influence object and every vertex of the driven object.

Deformation based on being in stretch mode is the kind that is considered to be applied for a muscle-based skin deformer. It shows properties that resemble the ones of muscle and skin. Assume for now we have fatty tissue between the muscle and the outer skin layer, the muscle pushes the skin away depending the distance of the fatty tissue. The fatty tissue can even be compressed and bulged.

The sculpt deformer lacks some compulsory features. Only a NURBS sphere can be used as influence object. Modifying the shape of the NURBS sphere allows us to rebuild a muscle geometry. However, the muscle deformer is not able to deform NURBS surfaces. This is clarified considering the issue of only having access on the control vertices and not the edit points (cf. Chapter 4.2.3.2 and Chapter 5.1.2.3). Furthermore, painting weights to exclude vertices from being computed is not allowed. Finally, accessing the interior of the deformer is not enabled.

In the next section, we detail how we finally implemented the skin deformer. We will divide the chapter in separate sub-problems as the skin's behaviour cannot be concerned at once, as already concluded in Chapter 2.2.4.

5.3.2 Final Realisation

In this chapter, we elaborate on the final implementation of the skin deformer. As discussed in Chapter 2.2.4, skin shows visco-elastic properties: *"Viscosity is originally a fluid property. Elasticity is property of solid materials. Therefore, a visco-elastic material combines both fluid and solid properties"* [Gla03] In our system, these two properties are separately implemented. It has to be mentioned that we implement the Maya® deformer node applied on the skin shape and with the muscle shape as additional input.

5.3.2.1 Elasticity (Skin Attachment)

The work of Jian [Jia04I] serves as basis for the implementation of the skin deformer, showing elasticity effects. Jian introduces a model that concerns a muscle model based on ellipsoids (cf. Chapter 4.4.4). Since we are not dealing with ellipsoids to synthesise the muscle, some adjustments are made. A polygonal mesh is used to model the skin shape.

To simulate elasticity effects, we have to attach the skin shape to the muscle at first. This is done by computing and storing the closest point on the muscle geometry to each skin vertex. Every computed point lies somewhere on the skin surface, not compelled to have a valid vertex position but located somewhere on a skin polygon.

The deformer provides a handle on an iterator to step through every vertex of the polygonal skin mesh. For each skin vertex, the closest point on the muscle geometry is computed. Our system determines the closest point and the polygon it is located on. The vector pointing from the closest point to the skin vertex is stored as well:

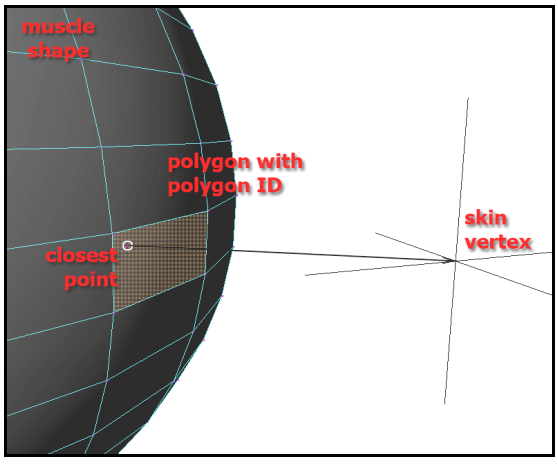


Figure 63: Closest point to skin vertex

Next, we create a handle on the muscle shape that allows us to iterate over each polygon (`MItMeshPolygon`). Utilising the previously stored closest points, the system associates them with

each polygon so that a parameterised position can be stored. Recall Chapter 4.2.2.3 and 4.2.2.4 where this parameterisation is further described: either we compute and store the barycentric coordinates or we determine corresponding UV coordinates. We decided to apply the second method as it does not depend on the vertex count per polygon.

To receive the UV position, several adjustments are made to guarantee proper calculation. However, in Maya®, obtained UV coordinates must not necessarily be correct. Maya® provides a method to obtain the according UV position for each closest point. It is recommended to have no overlapping polygons in UV space. Maya® provides a feature called *"Layout UVs"* that remaps UV coordinates in order to avoid overlapping polygons in UV space. Every point can now be uniquely assigned to the surface of the muscle. However, the system has to bypass another peculiarity of Maya®. See Figure 64 where the UV coordinate in (0,0) is displayed. Given the closest point lying on the muscle surface with the UV coordinate (0,0), a computation error can be provoked. The UV coordinate of a point near the border of UV space (borders are illustrated by the sides of the grey square, Figure 64) are possibly flipped to the facing border. For example, in case of receiving the UV coordinate (0,0), it may occur that Maya® computes the UV coordinate (0,0.9999999). Therefore, Points close to a border can be miscalculated. Thus, remapping the UV coordinate to a point on the muscle surface may provoke an error.

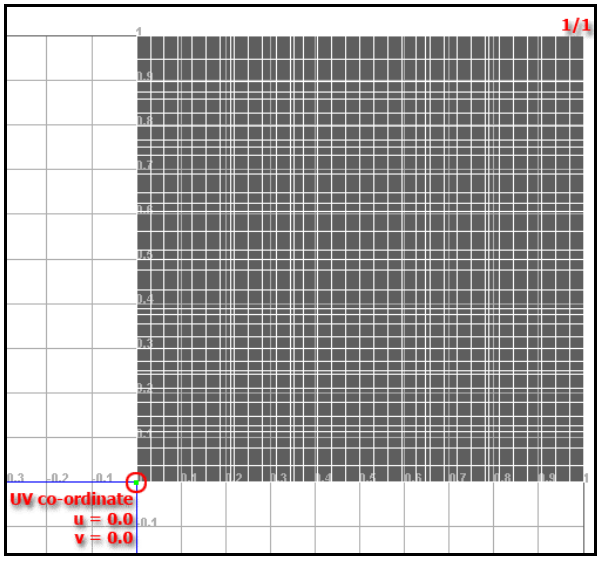


Figure 64: UV set of muscle shape

How do we avoid producing further errors? After having determined the UV coordinate of a point, we test UV coordinates on being acceptable by remapping them to a point. If Maya® produces an error, we act on the assumption that the U or the V coordinate is flipped to the opposite border. We either experience the U or the V coordinate to flip. Therefore, we check

the point based on each of the four borders exclusively. Each time we test the UV coordinate on producing a valid point by remapping. If it turns out positive, the test is interrupted and the computation can be continued.

Such as the previously described deformers, this deformer is evaluated in separate passes. In the first pass, all necessary data is initialised. In the next pass, we generate the ultimate skin deformation. Such as the other deformers, it is iterated through every vertex of the polygonal skin mesh. To exclude vertices from computation, the weight of each vertex is ascertained. Each vertex with a weight of 0 is obviated for further computation. The weight will play a more crucial role at a later date.

The skin deformer provides two attributes: *"minDist"* and *"maxDist"*. They refer to the maximum and minimum distance a skin vertex may have to its closest point on the muscle surface. This causes the skin deformer to exclude points that are far-off the reach of the muscle due to anatomical properties. At a later date, *"minDist"* and *"maxDist"* will account for the smooth transition between the deformation of skin vertices lying far away or close to the muscle.

Previously, for every skin vertex we stored the ID of the polygon whereon the closest point is located. Additionally, the UV coordinate for every closest point was retained. With both information, the point on the surface is determined. It has to be noticed that the skin deformer is not running through the first pass at the moment so that the shape of the muscle may not be in rest position but contracted. It is important to remind that varying the shape of the muscle does not change its configuration of UV coordinates. Hence, the previously computed closest points change their position but stay on the some place relating to the surface of the muscle.

In the next step, the system computes the smooth transition of influence on skin vertices. If the initial distance of the skin vertex to its closest point is approaching *"minDist"*, the influence on the skin vertex is the most. Otherwise, if it is approaching *"maxDist"*, it is affected with a low intensity:

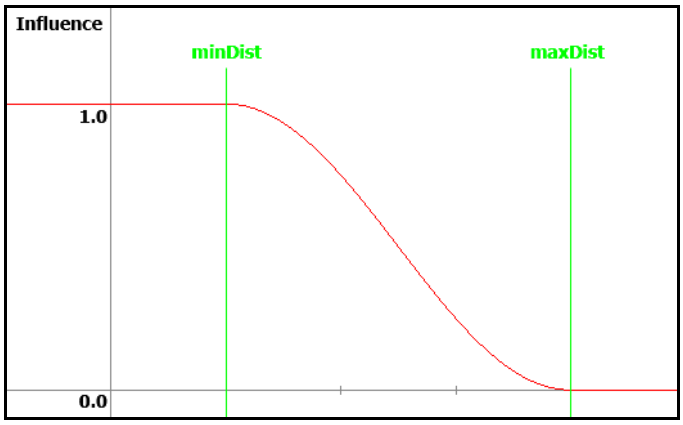


Figure 65: Influence on skin vertex, falloff [Jia04I]

Between the minimum and maximum distance, a falloff (shown in the diagram of Figure 65) generates a smooth transition within the deformation.

Finally, the new position of each skin vertex is computed. The influence falloff is from now on referred to as *"alpha"* [Jia04I].

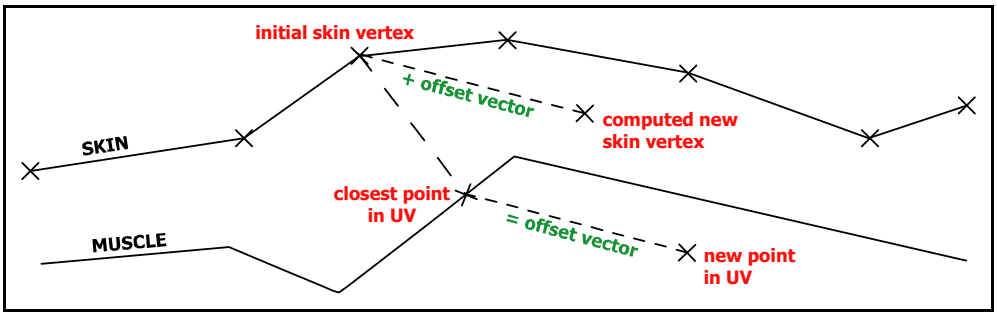


Figure 66: Skin deformation based on varying muscle shape

See Figure 66 where the muscle affects the skin. Given the initial closest point on the muscle surface, deforming the muscle influences its position. The position alteration is stored in the offset vector. Afterwards, the offset vector is added on the initial skin vertex depending on alpha and the vertex weight:

```
// initDist          -> initial distance from skin vertex to closest point
// ptAtUv            -> new point on muscle parameterised by UV coordinates
// clsPts[i]         -> initial closest point on muscle
// pt                -> skin vertex

// falloff for a smooth transition in the deformation
float alpha = ( initDist - minDist ) / ( maxDist - minDist ) * PI - PI / 2.0;
alpha = 1 - ( sinf ( alpha ) + 1 ) / 2 ;

// offset vector, new point on muscle to initial closest point
MVector offset_vector = ptAtUv  -  clsPts[i];

// new skin vertex
pt = (pt  + ( offset_vector ) * weight * alpha) ;
```

Source Code 10: Computation of skin vertex

Jian [Jia04I] includes the computation of a rotation vector for each skin vertex depending on the orientation of the whole muscle. Since he uses an ellipsoid to represent the muscle, it is reasonable to have the orientation of the muscle embedded in the calculation. In case of this system,

there is no reason to create a dependency to the orientation of the muscle as every shape is accepted anyway.

The model described in this chapter simulates the elastic attachment of the skin to the muscle. It is adequate for the task of generating muscle-based facial animation. The motion of the skin appears smooth, and authentic and in addition to that it obeys the underlying muscle. At the moment, the system claims to be able to run the animation in real-time. However, the skin deformer model lacks physical correctness at the moment. The system does not consider viscosity effects caused by the layers between the muscle and the surface of the skin. Until now, the skin is tightly attached to one point on the muscle. Since we have elasticity effects, not every skin vertex is deformed proportionally to its corresponding muscle including the point offset. In consequence, the muscle may penetrate the skin, primarily when it thickens. This is due to the fact that the skin deformer considers solely one reference point on the muscle for each skin vertex. Nothing prevents other muscle points from moving close to the skin that initially should not affect them. Thus, the muscle may penetrate the skin.

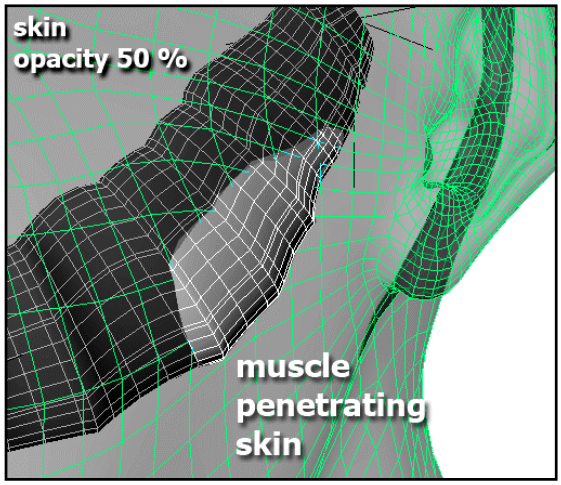


Figure 67: Muscle penetrating skin, contraction factor 0.7

To avoid muscle polygons to intersect with skin polygons, a certain distance between skin and muscle is guaranteed. In the next chapter, we discuss the previously mentioned behaviour of the intermediate layers between muscle and skin. We will describe viscosity effects of the fatty tissue in particular. These effects cause the skin to be pushed away.

5.3.2.2 Viscosity (Slide Bulge)

In Chapter 2.2.4, the function and characteristics of the skin is expanded on. Having Chapter 2.2.4 in mind and referring to the previous section, it emerges that it makes sense to consider visco-elasticity effects. The viscosity effect characterises material that is fluid but has an internal resistance to flow. In case of the layers between muscle and surface of the skin, the viscosity effects make the muscle and the skin to be attached to each other not with tight references. In fact, the intermediate layers prevent the surface of the skin to touch the muscle. Besides to that, the intermediate layers are compressible so that bulge effects may be provoked.

In the following, the viscosity effects are described considering the displacement of the skin's surface. The following model is not capable of being used in an interactive environment since it lacks fast computation.

Recall the previously considered skin penetration by the muscle. How is a certain distance between skin and muscle guaranteed? As the skin deformer previously excluded some points from computation, only the important vertices are considered. It is iterated through every skin vertex and each vertex is tested on being too close to the muscle geometry. This task is performed by shooting a ray away from the vertex along its normal in both directions. To avoid shooting two rays, a fake point is determined located along the line of the normal but more distant:

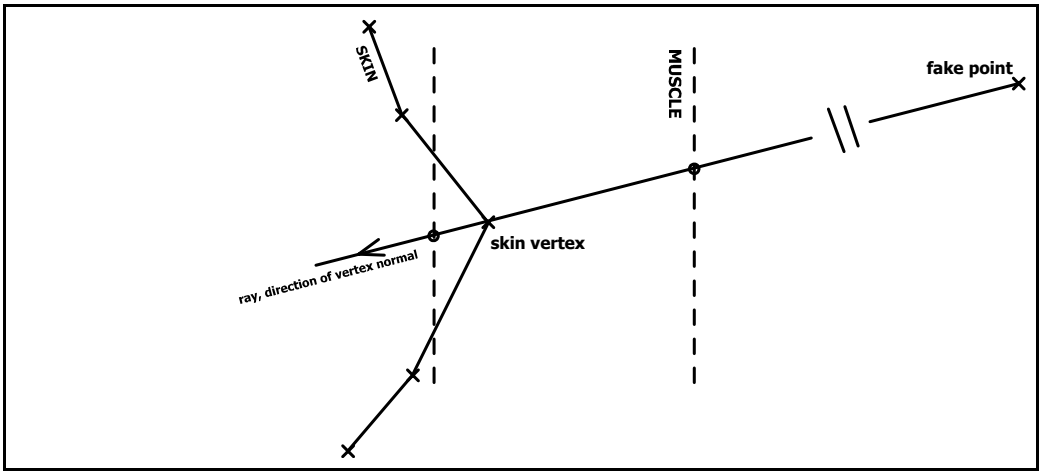


Figure 68: Computation of fake point to only shoot one ray

Given its origin in the fake point, one ray is shot to determine the intersections with the muscle (see Figure 68). Testing on intersection with the muscle for two rays per vertex would double the time-consuming computational effort. Since it is not possible to access the interior of Maya®'s intersect method, it can only be speculated about the computation steps of this method. In all probability, every polygon of the muscle is tested on intersecting with the ray.

However, it has to be considered that every skin vertex in the range of interest is affected. These vertices are tested on maintaining the right distance to the muscle. Hence, for each vertex, it has to be checked if its normal line is intersecting with a muscle polygon. Given a muscle with a convex polygonal shape, not more than two intersection points may result.

See Figure 68 where the intersection points (in the following referred to as *"hit points"*) are marked with a circle. For further handling, we have to consider the hit point that lies on the normal, pointing away from the skin surface and being the one most distant (if the skin is lying 'under' the muscle). The user is allowed to define the average thickness of the intermediate layers - in the software denoted as *"fatty tissue"*. When a skin vertex is lying inside the muscle or closer to the muscle surface than limited by the thickness of the fatty tissue, the skin vertex is displaced along its normal. The new skin vertex is computed as follows:

$$P_{skin} = P_{hit} + \hat{v} * fattyTissue$$

$$\hat{v} : \text{normalised normal vector in } P_{skin}$$

Equation 62: Displaced skin vertex

Iterating through every skin vertex causes the skin to behave similar to viscous material. As only vertices are taken into account for computation, polygon edges of the muscle may still appear outside the skin. By modifying the thickness of the fatty tissue, this can be avoided:

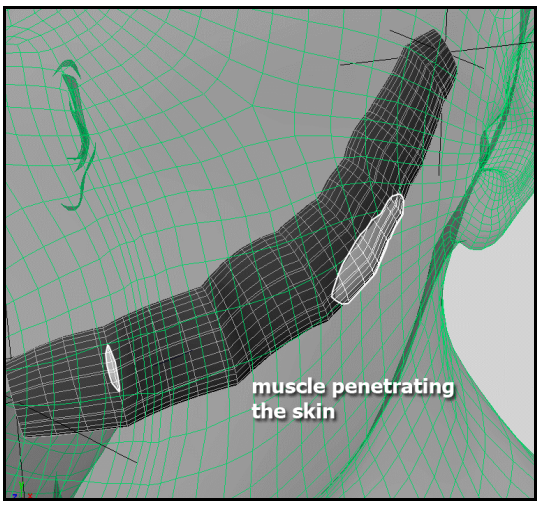


Figure 69: Slide bulge disabled

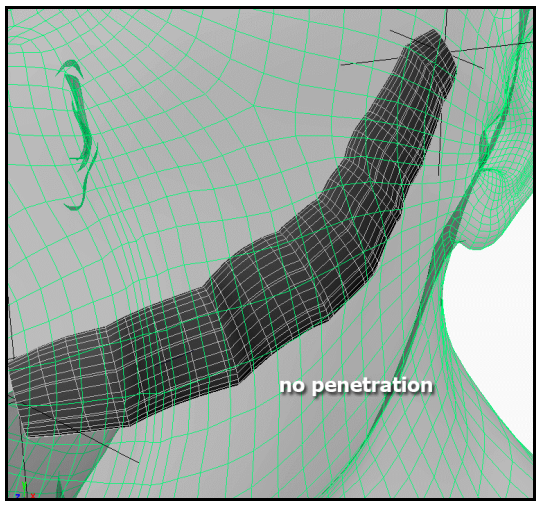


Figure 70: Slide bulge enabled

In the next chapter, the architecture of the skin deformer is discussed. Such as the architecture of the muscle deformer, the skin deformer will be described in a nutshell as it resembles the muscle and action line deformer.

5.3.2.3 Architecture

In the previous sections, it might be noticed that this deformer exclusively accepts a polygonal mesh. Remind Chapter 5.1.2.4 and Chapter 3.1.3 where the structure of a deformer illustrated. Once again, similar to the muscle deformer, we are concerned with a polygonal mesh whose vertices are manipulated to shape the polygonal mesh. First the skin shape is modelled and then the deformer is applied to. To have the skin deformer governed by a muscle, the polygonal mesh of the muscle serves as additional input for the skin deformer. In the previous chapter, the additional attributes were introduced that provide further control over the skin deformation. These attributes can be particularly modified by changing their values. Unlike the muscle attrib-

ute represented by an input geometry, these attributes do not require an extra input connection. They can be varied by changing the values in the attribute editor for this deformer:

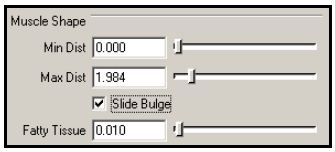


Figure 71: Attributes, skin deformer

The MEL script that establishes the skin deformer can be derived from the MEL script responsible for establishing the action line deformer (cf. Source Code 7). The structure of one skin deformer is shown in the following diagram:

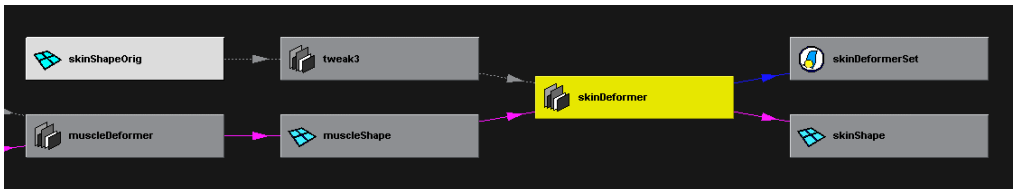


Figure 72: Structure of skin deformer

For each muscle, one skin deformer is applied to create the structure of the system. To avoid muscles counteracting each other, weights should be painted in order to have the influence areas separated. However at this stage, we have no implementation of an automation that assumes the task of subdividing influence areas. A Maya® deformer allows one weight value per vertex so that just one influence area can be defined.

The implementation of our system for muscle-based facial animation is appropriately discussed in this chapter. The system is developed being granular and modular so that it enables further enhancements. Each deformer node uses common object-oriented design patterns allowing a software developer to reuse, derive, or extend existing functionality.

In this chapter, we elaborated on the status quo of our software and its underlying theories. However, the system can be further improved. In the following chapter, first the results of this software will be discussed. We will provide some proposals to improve and optimise the system. In addition to the proposals, we will address some points concerning this system in terms of future work. We will bridge to other fields of research.

6 Conclusion

"To finish a work? To finish a picture? What nonsense! To finish it means to be through with it, to kill it, to rid it of its soul, to give it its final blow the coup de grace for the painter as well as for the picture"[1]

In this chapter we assess a system for muscle-based facial animation. In addition, we go one step further and provide ideas and theories how the system can be improved and extended. The system allows us to animate facial expressions on the basis of anatomical characteristics. As the time to develop the software was limited, the implementation is subject to restrictions in terms of its extent. Developing additional features that we considered to be unessential would have gone beyond the scope of this thesis as well as adding features that are not immediately concerned with the task of muscle-based facial animation.

The following, results of our implementation will be under consideration. Based on the results, in Chapter 6.2, we will have a foresight of how the system can be extended and conveyed into other scientific areas. Altogether it will turn out why muscle-based animation of facial expressions makes sense.

6.1 Results

The concept we developed for the software could not always be immediately realised. Some models we discussed in the experiment sections did not suffice to produce convincing results. However, the results of the software we ultimately implemented are promising so far. Two major guidelines were crucial in the development of the software. The software should produce persuasive facial animation in terms of appearance. Furthermore, the system should be usable in real-time. The software now meets both requirements.

The software is technically mature enough to establish a system of several muscles to animate facial expression. At the moment, it is a time-consuming task to establish a muscle-based system incorporating each and every facial muscle. To this date, it is an elaborate task fine-tuning the system to guarantee authenticity in appearance of facial animation. In Chapter 6.2, we provide some approaches that speed up the development of a sufficiently established system.

[1] **Pablo Picasso**

Previous research has already been undertaken in the realms of anatomically-based animation. Even anatomically-based facial animation was already concerned. This thesis achieves to balance different existing approaches, analysing them to compose a completely new model. The results of other systems concerning muscle-based facial animation made it possible for us to develop a more sophisticated model whose promising results are illustrated in Chapter A.1.1.

In the following, we recapitulate our final proposals we implemented and tested in a system for muscle-based facial animation:

- We implemented the action line by applying a mass-spring system that allows us to stretch the action line at an arbitrary point and to generate swing and collision effects.
- Our system enables the action line to contract along itself.
- We introduced a model that enables the use of Maya® NURBS curves of degree 1 to control a smooth cubic NURBS curve.
- We provided a model that allows the action line to control the muscle shape based on the *"smooth bind"* method.
- We created an implementation for generating the muscle bulge based on a Gauss curve.
- Our software enables the skin to be attached to the muscle based on elasticity effects.
- The system allows viscosity effects to be simulated by maintaining a certain distance between muscle and skin. The user can chose whether to compute these effects as they are rather laboriously computed.
- Finally, we described how to create a muscle-based system for facial animation in terms of a modular and granular software architecture.

In the next chapter, we discuss the work that can be done in the future. We bridge to other scientific areas. First, we provide information about how to extend the present software.

6.2 Future Work

As previously noted, this system can be ported and extended. This section comments on three different issues. The first concerns the optimisation of the system, the second is about portability of the setup and the last will address other research activities.

6.2.1 Optimisation

This section concerns our software in terms of optimisation. Extensions provided in this chapter were not deemed essential for a muscle-based system. Nonetheless, to perfect the existing system, we provide some approaches that could serve as vantage points.

6.2.1.1 Collision Detection

At the moment, collision detection is not incorporated. Even if facial muscles intertwine or repel each other, our system concerns this kind of collision by grouping muscles. However, to adhere to physical accuracy, each muscle has to react on colliding with other components. As previously commented on, the muscle shape itself cannot react on collision.

One way to detect a collision is to test every edge of a muscle if it intersects with the polygon of another component such as another muscle or the skull. This can be done by using a ray-polygon-intersection method similar to the one provided in Chapter 5.3.2.2. The information of the collision detection is propagated back to the action line. Having established a connection from the muscle deformer to the action line deformer, a dependency cycle may occur consequentially leading to an error. The action line deformer itself has to detect collisions. To avoid creating a dependency cycle, the action line could work on an instance of the muscle shape so that in case of collision detection only the original muscle shape is affected. However, as alluded to above the ray-polygon-intersection method is a very slow computation method since it uses complex iteration steps.

To react on collision detection, the point of collision is computed. This point is projected onto the action line. Translating this point on two adjacent control points, the weighed forces on these control points can be defined. Due to these forces, the mass-spring system of the action line reacts on collision.

Another simpler and less time-consuming algorithm represents the collision detection by the action line itself. Therefore, rough bounding boxes for each action line segment are initially determined. Their dimension depends on the segment length and the average distance of each muscle vertex to its projected action line point. By referring to the world space coordinate axis, each segment gets an approximate bounding box assigned.

Having pre-estimated each adjacent muscle's bounding box, it can be tested if it is inside a bounding box of the particular muscle. When a collision is detected, the action line reacts by exerting a force to the corresponding point of the action line as previously described.

Aubel provides an approach that can well be used to compute collision detection (cf. Chapter 4.4.7.1). He uses force fields that are formed by ellipsoids. They immediately affect the

mass-spring system. He differentiates between repulsive and attractive force fields. Repulsive force fields prevent the action line from penetrating the ellipsoid; the attractive force fields are applied to refine the shape of the action line.

In contrast to Aubel's model, we recommend the use of locators that define spherical force fields and that can be connected to the action line deformer. This would simplify the organisation and computation of collision detection.

6.2.1.2 Jiggle

The system is not able to react on forces that cause the muscle to swing. External forces induced by shocks such as a punch in the face, abrupt actions of the whole head, or gravity may jiggle a muscle or the skin. Since in our system the muscle is already based on a mass-spring system, we only consider the swing of the muscle.

Recall the description of the mass-spring system that controls the action line (cf. Chapter 5.1.2.1). The system runs in an environment not depending on time. However, swinging is a time-dependent action. Unlike in our model, a swinging mass-spring system cannot be evaluated iteratively. The time factor must be concerned by the system. To finally implement swinging effects, the position of a particle of the action line does not exclusively need to be computed based on its previous position. For each particle, the velocity and acceleration factor are introduced (cf. Chapter 4.4.5). The external forces are conveyed to be embodied by the acceleration of an action line particle. Having defined additional factors like damping and spring stiffness, the jiggle of the muscle can further be refined. Forces are applied to particles affected by other objects. These particles are commonly considered to represent the origin or the insertion.

6.2.1.3 Skin Deformation Improvement

In this chapter, we elaborate on the improvement of the skin deformation in terms of physical accuracy. The model provided in Chapter 5.3.2 allows us to be perfected. The elasticity of the skin is described considering its attachment to the muscle.

Given a skin composed of a polygonal mesh. Every vertex of the mesh represents a particle in a mass-spring system. This mass-spring system is applied on each particle whereas particles are connected to each other. Referring to Chapter 4.4.5, where Nedel and Thalmann [NT98] presented an elasticity model for a muscle. This model can be applied to the skin. If the mesh consists of polygons with a count of three or four edges, every vertex is linked by four springs to its adjacent vertices. The force on the particular vertex is defined by the sum of all four forces. Utilising angular springs (cf. Chapter 4.4.5), the system can prevent the skin from twisting or bending. In addition to our existing system, the appearance of the skin can be kept

smooth. It has to be pointed out that applying a mass-spring system on each skin vertex slows down the system tremendously.

Another way to improve the deformation of the skin is to extend the simulation of viscosity effects. The system attempts to preserve the distance between muscle and skin. Therefore, the distance to the muscle is computed for every skin vertex and at every moment. The same can be done reversely: each muscle vertex can be tested on being distant enough from the skin. The computational effort would nearly be the same. Finally, to guarantee physical correctness, the maintenance of distance between the muscle and the skin is considered in both directions: muscle vertex to skin and skin vertex to muscle. For example, if the mesh of the skin is less finely woven than the muscle mesh, the muscle may penetrate the skin when being deformed. To create viscosity effects, the approach provided by Aubel (cf. Chapter 4.5.2) is considered best but most complex.

6.2.1.4 Wrinkles

"The process of wrinkle formation can only be simulated by including viscosity, though." [Kae03] We deem this statement as being true to certain degree. The simulation of wrinkle formation can only be done considering viscosity if we adhere rigidly to physical correctness. For instance, Wu and Magnenat-Thalmann provide such an intricate model to generate wrinkles based on plastic-visco-elastic effects [WM98].

In Chapter 4.4.2, we exemplify a model for generating wrinkles based on a single layer. This model can be utilised in our system. The model described in Chapter 4.4.2 uses a vector to represent the muscle. This vector and its influence area serves to apply a certain wrinkle function to the skin (see Figure 23, right and Equation 26). Likewise, the wrinkle function can be applied in our system. Therefore, the action line is projected onto the skin. The wrinkle function can then be aligned to the projected action line. The wrinkle function should be exclusively applied when the action line contracts. The amplitude of the wrinkle function depends on the contraction factor of the corresponding action line.

The generation of wrinkles shapes up as being difficult to display when having a coarse polygonal mesh of the skin. The wrinkle is only visible if its generator function is able to grab a vertex for each feature point of its function curve (see Figure 23, right). Otherwise, the wrinkles do not appear properly.

The wrinkles can also be displayed by using bump maps. Wrinkles created by facial expressions appear by driving in bump maps that exclusively show wrinkles of the according expression. To automatically generate these bump maps for each facial expression, wrinkles are com-

puted using the function previously described. The wrinkles are translated from 3D to 2D space of by producing a relief picture.

Wrinkles are an important part of 3D facial animation as they enormously contribute to believable facial expression. In addition to that, skin aging can be simulated via simulating wrinkles. Wrinkles also play an important role in dermatological and forensic contexts (cf. Chapter 6.2.3)

6.2.1.5 Application of Motion Capture Data

At the research project *"Artificial Actors"* at the Filmakademie Baden-Württemberg, the idea of using motion capture data to animate facial expressions already exists [HBSL04]. Based on this idea, it must be possible to translate motion capture data to our system to control muscles. However, motion capturing can only take place on the surface of the facial skin.

Focusing on the two attachment points of a parallel muscle, especially the insertion, these are the major feature points which we have to pay attention to. These feature points are projected on the skin to obtain the target position of the markers which are motion captured. This process can be reversed, by starting from markers at arbitrary positions or anthropometric landmarks and translating these back to the control points of the muscle.

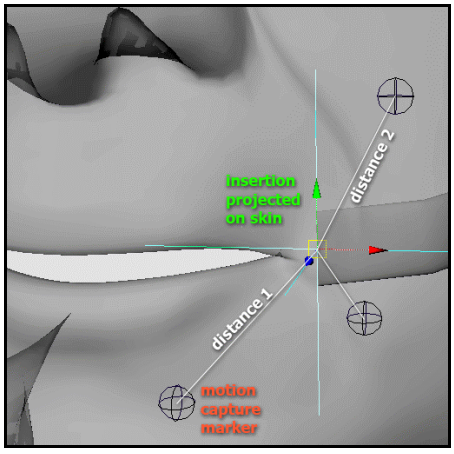


Figure 73: Computing trajectory by translating closest motion capture markers

See Figure 73 where the computation of the trajectory of the insertion is sketched. Given a set of motion capture markers, for each marker, a certain trajectory is captured. The trajectory of the insertion is computed similar to the *"smooth bind"* method (cf. Chapter 5.2.2.1). In addition to the method we previously described, the control points of the action line can now be driven by more than two markers. This is due to the fact that the markers and the control point of the action line are located in 3D space. It has to be stated that computing the trajectory of the inser-

tion must not necessarily yield a correct result as we omit the thickness of the skin in this calculation. It would only be correct if the projected point of the insertion on the skin's surface is taken into account. To have the insertion controlled by the closest markers, the influence of the markers on the insertion are computed by determining differing distances from the markers to the insertion projected on the skin surface (see Figure 73). This way, weights can be computed in order to smooth bind the insertion to markers (cf. Chapter 5.2.2.1, particularly Equation 56 and Equation 57).

Another indirect way to bind the insertion to the markers is to transfer the trajectories of the markers to joints. These joints are smooth bound to the skin mesh (cf. Chapter 4.3.3.2). The insertion of the muscle is finally constrained to the skin mesh which has the same effect. However, the skin mesh may only serve as intermediate reference object since the final deformation caused by the muscles may not affect this skin mesh. Affecting this skin mesh it could run the risk of having a cycle dependency.

Finally, the trajectories of the markers can be adapted using the same mechanism as introduced in Chapter 4.3.4. Having captured all basic expressions based on Ekman's FACS [Ekm78] presented in Chapter 4.1.1, the system is extended in a manner that allows us to create almost every facial animation in a quality that can compete with real facial actions. Ultimately, the system is user-friendly in such a way that it works almost automatically.

In the next section, we will address the issue of portability of the system. At present, constructing the muscle-based system for the whole face is very time-consuming. Instead of providing a method to easily revise the entire system, we introduce a mechanism that simplifies the transfer from a head geometry to a completely different target head geometry.

6.2.2 Portability

The research project *"Artificial Actors"* at the Filmakademie Baden-Württemberg is a combined effort of artists and computer scientists creating tools that ease the generation of authentic facial animation. This includes not only to facilitate the handling of animation tools but to make them usable on every target object. On the following pages, we will demonstrate some approaches that allow cloning the present system on a source geometry to a different target shape.

First, we will make some proposals considering the easy set up of the system. To establish the system it may be useful to refer to *"The Visible Human Project®"* [VHP86I] that provides realistic 3D voxel data models of the anatomy of the head. These models allow the facsimile of a genuine human. Thus, the skull, the muscles, and the skin can be prepared. Next, the action lines of each muscle are defined. Finally, all deformers are applied including all required locators that control the animation.

Realistically, setting up the parameters of the entire system can be a cumbersome task. However, if it is once made it can be reused. In the following we will present a method that enables the system to be transferred on a completely different target geometry.

6.2.2.1 Geometry Matching

Consider Chapter 4.3.2.1, where the thin plate spline RBF morpher is discussed. This approach serves to morph a geometry onto a completely different target geometry. Based on locators placed on anthropometric landmarks on the source and target head, the source head is fitted to the target head. In Chapter 4.3.2, we devoted the use of this method to clone expressions with the use of this method. This time, the technique is extended to fit each component of the system to animate muscle-based facial expression on a target head. Each element controlling the facial animation is based on a set of vertices located in world space. The skull, the muscle, and the skin consist of polygonal meshes hence they consist of a set of vertices. The action line is a NURBS curve of degree 1 that means it is composed of a set of vertices as well. Finally, the system includes locators that constrain components and stick the system together.

Having placed the locators (cf. Chapter 4.3.2.1 and see Figure 17), the RBF morpher can be applied. Now, as we are concerned with a more complex setup, the RBF morpher is applied on each component. The source setup is fitted into the target geometry of the head. To make the system usable, all parameters controlling the action line, muscle and skin deformation have to be further tweaked. After all of these adjustments, one component is missing. In Chapter 5.3.2.1, the deformation of the skin concerning elasticity effects is discussed. To exclude skin vertices from being driven by a muscle, a weight for each vertex is defined. Maya® provides a tool that allows to paint weights. In the next chapter, we will show how to clone skin weights to the target geometry.

6.2.2.2 Skin Weights Cloning

To define the degree of influence the muscle has on the skin, each skin vertex has a weight assigned to it. The weights of the entire skin can easily be defined by using the *"Paint Attributes Tool"* provided by Maya®. The weights for a muscle, for example the zygomaticus major (see Figure 6), are depicted in Figure 74.

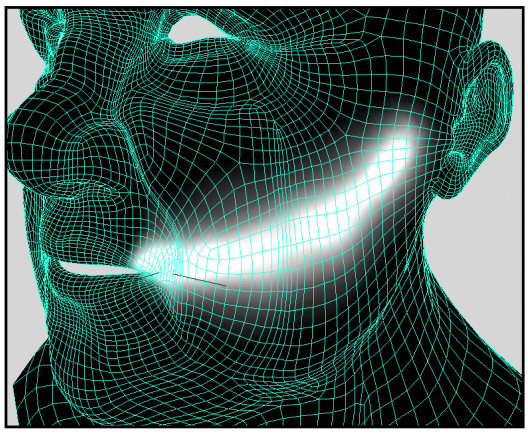


Figure 74: Weights of zygomaticus major

In the previous chapter, we already fitted the system into a target head geometry. Having all necessary information gathered to apply the RBF

morpher, cloning weights of the source skin to the target skin can also be performed. To clone skin weights not only the RBF morpher is applied, but the skin mesh is fitted to the target skin mesh. This process is exemplified in Chapter 4.3.2.2.

If the source skin is fitted to the target skin, the target skin vertices lie on the source skin but not necessarily on source skin vertices. Resembling the approach in Chapter 4.3.2.3, the source skin mesh is triangulated. To give the target skin vertices a unique assignment to the source skin triangles, barycentric coordinates (cf. Chapter 4.2.2.3) are computed. With the barycentric coordinates, the resulting weight of the target skin vertex can be calculated. See Figure 12 which signifies how the point in the triangle obtains its skin weight. Therefore, the weight of a target skin vertex is calculated using its three surrounding points of the triangle of the source skin mesh. All surrounding triangle vertices have one value for the weight and by utilising barycentric coordinates as coefficients, the resulting weight of the target skin vertex can be summed up (similar to Equation 23) :

$$w_r = b_0 * w_0 + b_1 * w_1 + b_2 * w_2$$

w_r : resulting weight of target skin vertex

b_i : barycentric coordinates

w_i : weights of surrounding triangle vertices of source skin

Equation 63: Calculation of skin weight

Having all components of the system and the skin weights fitted to the target geometry, the cloned system is ready to use. Thus, the workflow of setup creation is tremendously expedited. The system can therefore be applied on every character that resembles anatomical structure.

6.2.3 Other Research Activities

6.2.3.1 Criminology and Forensic Reconstruction

In the domain of criminology, many scientists are engaged in recognition, construction, and extension of facial models. Up to the present, if a witness has to identify a face, artists draw an image based on the instructions of the witness. Therefore, identikit systems serve to ease creation. If the witness instead has to recognise a face out of a group of similar looking persons, resembling test persons have to be hired.

Meanwhile, computer generated simulations make it possible to find a remedy. In the majority of cases, it is a matter of 2D graphics. Several efforts have been made to improve computer generated 2D images to create convincing faces based on identikit systems. Nowadays, 3D computer graphics come into operation to simulate head models. Several techniques have been

introduced in this thesis to create (cf. Chapter 4.2.2.1) such head models and to finally animate them (cf. Chapter 4.3). In reality, these models do not suffice to guarantee anatomical accuracy. Furthermore, these models have no easy to handle parameterisation that allows us to vary the appearance of the face and its resulting facial animation.

Based on our system, faces looking like living humans can be created and animated. Showing animated facial expressions may simplify the identification of a person. Additionally, movies can be produced with real actors apart from the face, which can then be replaced by a computer-generated model created by our system.

Remember the tsunami disaster that devastated South-East Asia in December of 2004? Countless humans lost their life. Afterwards, some of the dead could hardly be identified, as portions of the face were destroyed. Furthermore, a cadaver decays over time and eventually all that remains is the skeleton. The skeleton is only an indicator how the face might have looked during life. Hence, another field of application for our system is presented: forensic reconstruction.

Based on a 3D scan of the skull, our setup can be applied. Recall the chapter about portability of our system (Chapter 6.2.2). The RBF morpher is applied by the use of locators that are placed on anthropometric landmarks on the skin. However, the same outcome can be achieved by placing the locators on feature locations on the skull. Thus, the system is fitted to a target skull. Modifying parameters of the system may generate a facsimile of the original face. Now the face can be identified by dependents or other concerned persons.

Another area of application for our system is the reconstruction of creatures for the realms of palaeontology, archaeology or other historical and scientific purposes. Famous dead persons or, for example, Oetzi the Iceman can be revived.

6.2.3.2 Plastic Surgery and Dermatology

Planning plastic surgery or minimal invasive medical procedures is a complex task as anatomical behaviour of body parts is laborious to simulate. Our system can serve to imitate physical characteristics of the muscles and the skin. For plastic surgery, the skin deformation and final appearance can be realistically simulated. Traditional preoperative planning and predictions of results of surgical invasions was limited to 2D space. Complicated minimal invasive medical procedures may cause damage if they are not adequately premeditated. Using our system we do not have restrictions in space. Since scaling is an affine transformation, we can zoom into our system of muscles and skin ad finitum. Based on geometries in 3D derived from a model generated by computer tomography (CT) scanning or magnetic resonance imaging (MRI), our system accounts for guaranteeing anatomical and physical accuracy. *"However, the main goal of computer assisted surgery (CAS) is to simulate physical interactions with virtual bodies. In particular, the realistic simula-*

tion of soft tissue deformations under the impact of external forces is of crucial importance." [Gla03] As our system can be interactively run, not solely planning is made possible but also the accompanying simulation of the invasion is enabled.

Another important field of research where our system can be applied is dermatology. Skin aging can arise from both biological properties and from environmental impacts. The age of the skin can be recognised by its colour or its texture. Wrinkles usually signify an advanced age. Recall Chapter 6.2.1.4 where a model generating wrinkles is introduced.

The older a person grows, the less elastic the skin gets (cf. Chapter 2.2.4). That means that wrinkles caused by facial actions may remain. Consequentially, the wrinkles and thus skin aging can be modelled utilising our system. Consider a young person is missing: the appearance of the person after many years can be simulated. Hence, we can bridge to criminology too as in this domain it may occur that pictures of missing people have to be generated; even after many years have passed.

6.2.3.3 Robotics and Artificial Intelligence

Our system may be employed in the area of robotics and artificial intelligence. Human-machine systems play an increasingly important role in dealing with artificial intelligence. In this context basically in the area of robotics, facial expressions have to be considered when creating human-machine interaction. One good example is the development of *"Kismet"*, a robot that is provided with visual, auditory, and proprioceptive sensory inputs [MIT00]. This robot is able to engage people in natural and expressive interaction.

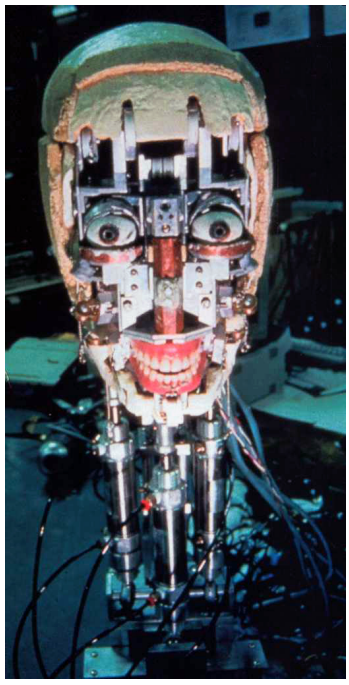


Figure 75: "Roberta" [Har00]

Another example of a robot having the ability to show facial expressions is the first *"face robot"* called *"Roberta"*, created by Professor Fumio Hara of the Science University of Tokyo [Har00]. The robot is controlled by 24 hydraulic cylinders (see Figure 75) which allow the robot to show the six major facial expressions (cf. top categories of facial expressions in Chapter 4.1.1). Based on the six common expressions, *"Roberta"* is able to show a realistic imitation of human facial actions. This is one component to approach to creating authentic human-computer interaction.

Plantec presents his work about *"Virtual Humans"* that attempts to provide guidelines how to create the illusion of intelligence [Pla03]. This is done by believable facial expressions and the

ability to handle question and answer interaction. Plantec refers to avatars that are virtual representations of humans, particularly in the space of the internet. He describes how to provide them with human behaviour and how to make them interact.

To create realistic human-machine interaction, the machine has to adopt human attributes. Showing facial expressions is one of the most significant components involved in creating the illusion of a real human being. Our system is able to help reduce the effort involved in creating a construction plan for something like an avatar or a robot being able to interact with humans. It helps to virtually simulate facial expressions based on anatomical and physical accuracy. Utilising a real robot also allows for the adherence to the laws of physiques. To save time in testing and experimenting, the system can reasonably be utilised to predict results.

6.3 Final Prospects

The model and the additional information provided in this thesis offer an incentive to further research in the domain of facial animation. The results of this work encourage continuing the investigation in muscle-based facial animation. The capability of extending our work in terms of computational efficiency, usability, and authenticity has become evident. Over the years, computing power increases so that some aspects mentioned in this thesis can be reconsidered in time.

At present, we are satisfied that this work presents the best methodology considering all demands that we previously made on this process. All approaches detailed in this work offer various opportunities for development. As sure as fate, many researchers will follow and meet the challenge of creating authentic facial animation.

A. Appendix

A.1. Imagery

A.1.1. Maya® Plug-In for Muscle-Based Facial Animation

In the following, some illustrations are depicted that show the bottom to top development of a muscle-based facial animation:

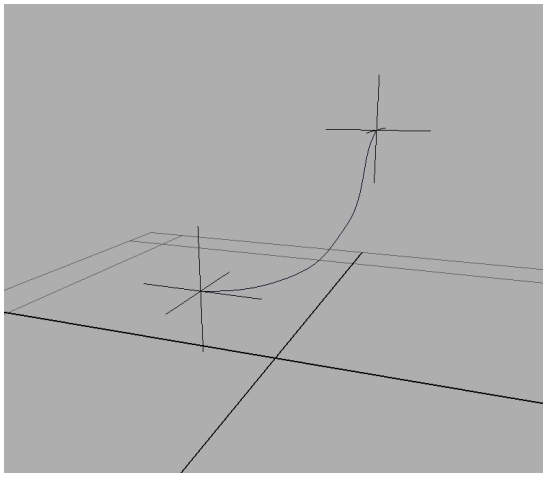


Figure 76: Action line

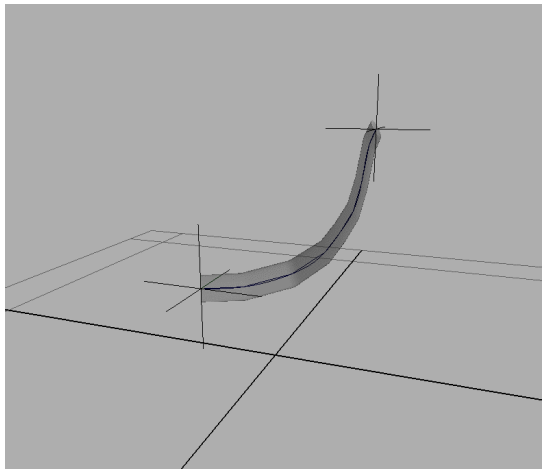


Figure 77: Action line and muscle

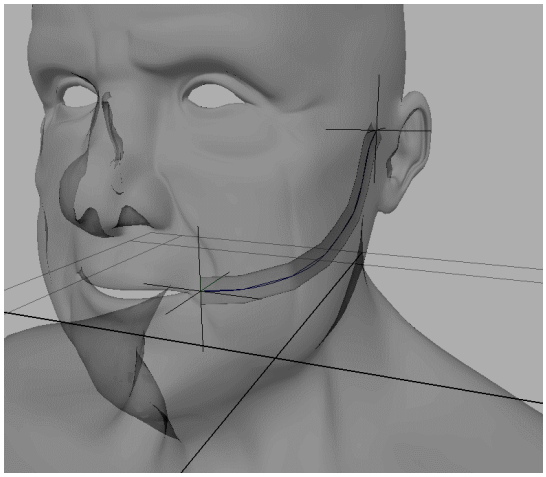


Figure 78: Action line, muscle and skin

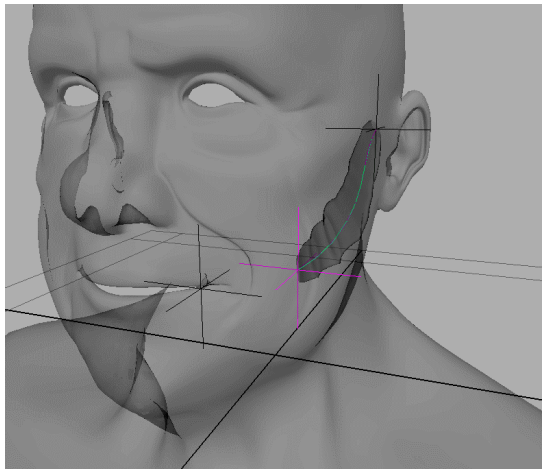


Figure 79: Action line, contracted muscle and skin

A.1.2. Expression Repertoire

In this section we show exemplary image concerning the expression repertoire that is created at the research project *"Artificial Actors"* at the Filmakademie Baden-Württemberg and that is in a stage of completion (→http://research.animationsinstitut.de). The following shows the six major emotional expressions:

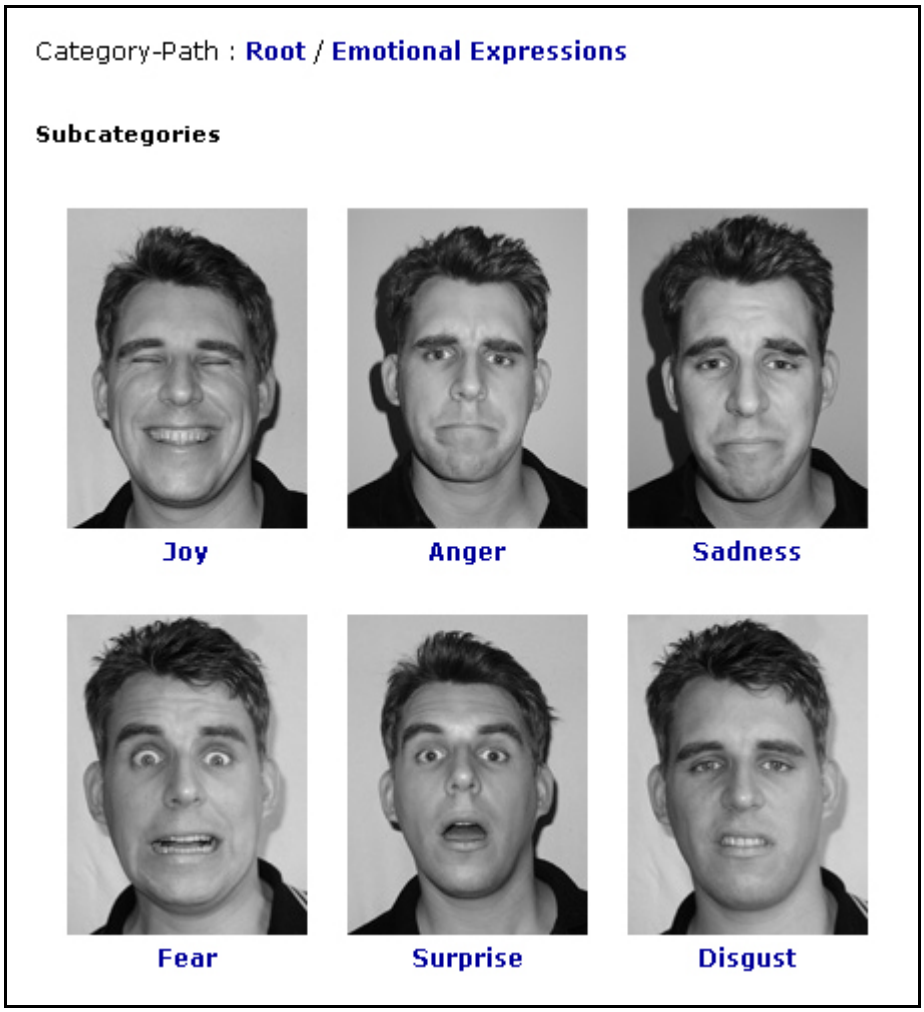


Figure 80: Six major emotional expressions

A.2. Single Actions Units of FACS

In the following, each action unit (AU) is listed. This table is taken from the investigator's guide of the FACS [Ekm78]:

AU Number	FACS Name	Muscular Basis
1	Inner Brow Raiser	Frontalis, Pars Medialis
2	Outer Brow Raiser	Frontalis, Pars Lateralis
4	Brow Lowerer	Depressor Glabellae; Depressor Supercilli; Corrugator
5	Upper Lid Raiser	Levator Palpebrae Superioris
6	Cheek Raiser	Orbicularis Oculi, Pars Orbitalis
7	Lid Tightener	Orbicularis Oculi, Pars Palebralis
8	Lips Toward Each Other	Orbicularis Oris
9	Nose Wrinkler	Levator Labii Superioris, Alaeque Nasi
10	Upper Lip Raiser	Levator Labii Superioris, Caput Infraorbitalis
11	Nasolabial Furrow Deepener	Zygomatic Minor
12	Lip Corner Puller	Zygomatic Major
13	Cheek Puffer	Caninus
14	Dimpler	Buccinnator
15	Lip Corner Depressor	Triangularis
16	Lower Lip Depressor	Depressor Labii
17	Chin Raiser	Mentalis
18	Lip Puckerer	Incisivii Labii Superioris; Incisivii Labii Inferioris
20	Lip Stretcher	Risorius
22	Lip Funneler	Orbicularis Oris
23	Lip Tightner	Orbicularis Oris
24	Lip Pressor	Orbicularis Oris
25	Lips Part	Depressor Labii, or Relaxation of Mentalis or Orbicularis Oris
26	Jaw Drop	Masetter; Temporal and Internal Pterygoid Relaxed
27	Mouth Stretch	Pterygoids; Digastric
28	Lip Suck	Orbicularis Oris
38	Nostril Dilator	Nasalis, Pars Alaris
39	Nostril Compressor	Nasalis, Pars Transversa and Depressor Septi Nasi
41	Lid Droop	Relaxation of Levator Palpebrae Superioris
42	Slit	Orbicularis Oculi
43	Eyes Closed	Relaxation of Levator Palpebrae Superioris
44	Squint	Orbicularis Oculi, Pars Palpebralis
45	Blink	Relaxation of Levator Palpebrae and Contraction of Orbicularis Oculi, Pars Palpebralis
46	Wink	Orbicularis Oculi

Table 3: Single Action Units (AU) [Ekm78]

A.3. How to Establish the Development Environment

A.3.1. Installation Alias Maya® 6.0

In order to install Maya® and successfully run our plug-in, the system has to fulfil some requirements:

- **Operating system:** Windows® XP Professional, Service Pack 2
- **CPU:** Intel® Xeon™ CPU 2.66 GHz
- **RAM:** 2 GB
- **CD-ROM Drive:** Yes
- **Graphics Card:** NVIDIA® Quadro FX 1000 (Hardware-Accelerated OpenGL®)
- **Disk Space:** 450 MB

Being provided with administrator privileges, the installer can be run.

A.3.2. Installation of Visual Studio .NET for C++

Since it does not demand to be run in real-time, the system has less special minimum requirements. The only requirement that has to be mentioned is a minimum of 900 MB disk space.

Visual Studio .NET can now be installed. It has to be emphasised that Visual C++ has to be chosen. If Maya® is installed after having Visual Studio .NET established, Maya® automatically creates a wizard that helps to set up a default Maya® plug-in. To finally develop a Maya® plug-in, the wizard guarantees that all required libraries are linked to the created project.

A.4. Operating Instructions

Having .NET and Maya® installed, all necessary files can be copied from the enclosed CD-ROM. In the *"MBFA"* folder, five subfolders can be found: *"extra", "dev", "docs", "files", "scripts"*. In *"docs"* the PDF-File of this thesis is located. In *"extra"*, supplementary information can be found such as film material. In *"scripts"*, the MEL script to apply the deformer can be found. *"files"* contains the Maya® binaries that show the results we yielded in this work, the muscles in particular.

First of all, we devote ourselves to the *"dev"* directory that contains necessary source code to develop the plug-in presented in this thesis. The entire folder can be copied to hard disk. In this folder, open *"dev.sln"*. This will open the entire project in .NET.

Based on the location Maya® has been installed to, some adjustments have to be made concerning the project properties: The *"Additional Include Directory"* have to be adapted to conform to the present Maya® location. Furthermore, *"Output File"* and *"Additional Library Directories"* has

to be concordant with the existing Maya® installation. Now, the project can be compiled generating the plug-in.

This process can be bypassed by directly copying the file *"mbfa.mll"* to the Maya® plug-ins folder:

```
%MAYA_LOCATION%/bin/plug-ins
```

The plug-in file can be found on the CD-ROM in the directory *"MBFA/dev/Release/.."*. Next, the system settings have to be modified. In the next step, set the environment variable *"MAYA_PLUG_IN_PATH"* to the folder you copied the plug-in to. The plug-in can now be loaded into Maya®. Choose *"Windows→Settings/Preferences→Plug-in Manager→mbfa.mll"*.

Having loaded the plug-in, you can pay attention to the MEL script setting up the according deformers. To do so, copy *"MBFA/scripts/mbfaCreateSystem.mel"* to:

```
C:/WINNT/Profiles/%USER%/maya/6.0/scripts
```

In your system settings, the environment variable *"MAYA_SCRIPT_PATH"* has to be set to the directory the script is copied to.

Copy the Maya® binary files *"MBFA/files/*.mb"* to a scenes directory. For example:

```
C:/WINNT/Profiles/%USER%/maya/projects/mbfa/scenes/
```

If Maya® is started and the plug-in is loaded, the file *"cleanHank.mb"* can be opened. The script "*mbfaCreateSystem.mel"* has to be sourced. This is done by opening the *"Script Editor"* and clicking *"File→Source File…"*. Choose the script and execute it by typing *"mbfaCreateSystem;"* in the *"Script Editor"*. The entire system is now established taking some time to initialise. The weights for the skin deformer are not defined yet so that the animation does not look persuasive. Use the *"Paint Attribute Tools"* to define the influence of the muscle on the skin. By modifying the deformer attributes, the animation can further be tweaked.

If setting up the system is required to be avoided, open *"finalHank.mb"*. All weights are painted and adjustments are made that guarantee a proper muscle-based facial animation.

In "*mbfaCreateSystem.mel"*, several other functions can be found that allow to create different kinds of muscles. All necessary information how they have to be used can be gathered from the descriptive inline documentation in this script.

A.5. Glossary

In the following, some terms are listed that occur in this thesis and that might be incomprehensible. Most of the definitions are taken from Wikipedia [Wik01I], MSN Encarta [MSN05] and mathworld of Wolfram [WolNDI]:

anthropometric	Anthropometry literally means *"measurement of humans"*. In physical anthropology it refers to one aspect of human variation: The different body sizes and proportions of individuals belonging to different populations.
affine transformation	An affine transformation is any transformation that preserves colinearity (i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances
bumpmapping	Bump maps are textures in greyscale that are put on objects to create the illusion of surface relief without reshaping the surface
collagen	The main protein of connective tissue. It has great tensile strength, and is the main component of ligaments and tendons. It is responsible for skin elasticity, and its degradation leads to wrinkles that accompany aging. Collagen also fills out the cornea where it is present in crystalline form. (De.: das Kollagen)
convex	Shape of a surface curving outward, or toward the eye. Every interior angle of a convex polygon is less than 180°
FK	Forward kinematic: Positions of particular parts of a hierarchy of limbs and joints to a particular time are computed from the orientation and position of the object, together with any information of the joints.
heuristic	In computer science, a heuristic is a technique designed to solve a problem that ignores whether the solution is provably correct, but which usually produces a good solution or solves a simpler problem that contains or intersects with the solution of the more complex problem. (De.: Heuristische Methode)
IK	Inverse kinematic: Computation of the required behaviour of a hierarchy of limbs or joints, so that the end effecter can be moved to a certain location
isosurface	An isosurface represents a surface of constant value (e.g. pressure, temperature, velocity, density) within a volume of space.
mastication	The process of chewing (De.: Kauvorgang)
minimal invasive	Medical procedure is defined as one that is carried out by entering the body through the skin or through a body cavity or anatomical opening, but with the smallest damage possible to these structures.
NURBS	Non-uniform Rational B-spline, cf. Chapter 4.2.3
ODE	Ordinary Differential Equations is an equation that contains a function like y=f(x) and is derivable. It involves x, y, y', y", … (De.: Differentialgleichung)
partial derivative	The derivatives of a function of multiple variables (De.: Partielle Ableitung)
physiognomy	A pseudoscience, based upon the belief that the study and judgement of a person's outer appearance, primarily the face, reflects their character or personality. (De.: die Physiognomie, der Gesichtsausdruck)
proprioceptive	Proprioception is the sense of the position of parts of the body, relative to other neighbouring parts of the body

singularity	The failure of a manifold structure. For example the non-existence of differentiability. A matrix is said to singular when two or more rows are linearly dependent (De.: Singularität)
UML	Unified Modelling Language, a non-proprietary, third generation modelling and specification language to visualise construct and document the artefacts of an object-oriented software.
vertices	A point in 3D space with a particular location, usually given in terms of its x, y, and z coordinates. It is one of the fundamental structures in polygonal modelling: two vertices, taken together, can be used to define the endpoints of a line; three vertices can be used to define a planar triangle. (De.: Scheitelpunkt, Eckpunkt)
visco-elastic	Having viscous (De.: dickflüssig) as well as elastic properties.
voxeling	A voxel is a volume element, representing a value in three dimensional space. This analogous to a pixel, which represents 2D image data.

B. References

B.1. Literature

[Ali03a] **Alias|Wavefront (2003);** Learning Maya® 5 with DVD Foundation; Tutorials for Maya® 5; Sybex Inc; p. 3

[Ali03b] **Alias|Wavefront (2003);** Support knowledge base; Support case #3005: *"What's the internal math for the smooth skinning interpolation"*

[Ali04a] **Alias Systems Corp. (2004);** Maya® Help; Online help of Maya® 6

[Ali04b] **Alias Systems Corp. (2004);** Maya® API White Paper; Introduction to Maya®'s architecture; p. 4

[All05] **Debbie Allen (2004);** How To Make Your Skin Care Count; Royal BodyCare

[Aub02] **Amaury Aubel (2002);** Anatomically-Based Human Body Deformations; Dissertation; Ecole Polytechnique Federale de Lausanne

[Bra04] **Heather Brannon (2004);** Dermatology: Skin Anatomy; About Inc.

[BBP01] **Gaspard Breton, Christian Bouville, Danielle Pelé (2001) ;** FaceEngine a 3D Facial Animation Engine for Real Time Applications; France Télécom; Proceedings of the sixth international conference on 3D Web technology

[BHPN03] **The Duy Bui, Dirk Heylen, Mannes Poel, Anton Nijholt (2003);** Exporting Vector Muscles for Facial Animation; University of Twente; Springer Verlag Berlin Heidelberg; Proceedings International Symposium on Smart Graphics 2003

[Bui04] **The Duy Bui (2004);** Creating Emotions and Facial Expressions for Embodied Agents; Dissertation; University of Twente

[Cha89] **John E. Chadwick (1989);** Layered Construction for Deformable Animated Characters; Proceedings 1989 ACM SIGGRAPH Computer Graphics Conference, pp. 243 - 252

[CZ92] **David T. Chen and David Zeltzer (1992);** Pump It Up: Computer Animation of a Biomechanically Based Model of Muscle Using the Finite Element Method; Massachusetts Institute of Technology Cambridge; Proceedings ACM SIGGRAPH 1992 Conference, pp. 89 - 97

[DDBC99] **Gilles Debunne, Mathieu Desbrun, Alan Barr, Marie-Paule Cani (1999);** Interactive multiresolution animation of deformable models; Computer Animation and Simulation '99, Milano; iMAGIS/GRAVIR – Caltech

[DON04] **DON (2004);** Department of the Navy, Naval Dental Centers; Muscles of the Head;

[DSB99] **Mathieu Desbrun, Peter Schröder, Alan Barr (1999);** Interactive animation of structured deformable objects; The Caltech Multi-Res Modeling Group; Morgan Kaufmann Publishers Inc.; Proceedings of the conference on Graphics interface '99

[Dub04] **Mark Dubin (2004);** Molecular Cell Physiology MCDB 3280; Muscle Myosin at the molecular level; University of Colorado Boulder

[Duc00] **Mike Ducker (2000);** Matrix and Vector Manipulation for Computer Graphics; Article for GameDev.net

[Ekm78] **Dr. Paul Ekman (1978);** Facial Action Coding System – Investigator's Guide; Guide for categorisation of facial expressions; Research Nexus

[GC84] **Henry Gray, Carmine D. Clemente (1984);** Anatomy of the Human Body; Atlas; Lea & Febiger; Fig. 199, Fig. 188,

[Gla03] **Evgeny Gladilin (2003);** Biomechanical Modelling of Soft Tissue and Facial Expressions for Craniofacial Surgery Planning; Dissertation; Freie Universität Berlin

[Gou03] **David A. D. Gould (2003);** Complete Maya® Programming; An Extensive Guide to MEL and the C++ API; Morgan Kaufmann Publishers; C++ API, Introduction; p. 273

[Har00] **Professor Fumio Hara (2000);** *"Roberta"*; A robotic head of a Japanese Woman; Stage of being able to show understandable facial expressions in 2000; Science University of Tokyo

[HBSL04] **Volker Helzle, Christoph Biehn, Thomas Schlömer, Florian Linner (2004);** Adaptable Setup For Performance Driven Facial Animation; Proceedings ACM SIGGRAPH 2004 Conference

[Hin01] **Victor Ng-Thow Hing (2001);** Anatomically-Based Models for Physical and Geometric Reconstruction of Humans and other Animals; Doctor thesis; University of Toronto

[JD75] **R. H. Jensen, D.T. Davy (1975);** An investigation of muscle lines of action about the hip: a centroid line approach vs. the straight line approach; Journal of Biomechanics, v.8, pp. 103-110

[Jer04] **Jason Jerald (2004);** Muscle Based Facial Animation; University of North Carolina at Chapel Hill; Lecture

[Kae03] **Kolja Kähler (2003);** A Head Model with Anatomical Structure for Facial Modeling and Animation; Dissertation zur Erlangung des Grades Doktor der Ingenieurwissenschaft; Max-Planck-Institut für Imformatik Saarbrücken

[KCCLP00] **Young-Min Kang, Jeong-Hyeon Choi, Hwan-Gue Cho, Do-Hoon Lee, Chan-Jong Park (2000);** Real-time Animation Technique for Flexible and Thin Objects; Conference Proceedings, WSCG 2000, pp. 322-329

[Lan99a] **Jeff Lander (1999);** Devil in the Blue Faceted Dress: Real-time Cloth Animation; Article for the 1999 Game Developer Magazine

[Lan99b] **Jeff Lander (1999);** The Trials and Tribulations of Tribology; Article for the 1999 Game Developer Magazine

[LTW95] **Yuencheng Lee, Demetri Terzopoulos, and KeithWaters (1995);** Realistic Modeling for Facial Animation; University of Toronto; Proceedings ACM SIGGRAPH 95 Conference, pp. 55-62

[MIT00] **MIT Artificial Intelligence Laboratory (2000);** *"Kismet"*; An expressive robotic creature with perceptual and motor modalities tailored to natural human communication channels; The Sociable Machines Project

[NN01] **Jun-yong Noh, Ulrich Neumann (2001);** Expression Cloning; ACM SIGGRAPH 2001 Proceedings, 2001, pp. 453 - 469

[NT98] **Luciana Porcher Nedel, Daniel Thalmann (1998);** Real Time Muscle Deformations Using Mass-Spring Systems; Swiss Federal Institute of Technology; Proceedings of the Computer Graphics International 1998, p. 156

[Pai99] **John Paitel (1999);** All the Muscle Physiology; Article in SlowTwitch.com

[Par74] **Ferderic I. Park (1974);** Parameterized Model for Facial Animation; IEEE Computer Graphics & Applications, pp. 61 - 68

[Par01] **Rick Parent (2001);** Computer Animation, Algorithms and Techniques; Morgan Kaufmann Publishers

[PB81] **Stephen M. Platt and Norman I. Badler (1981);** Animating Facial Expressions; Proceedings ACM SIGGRAPH Computer Graphics, pp 245 - 252

[Pla03] **Peter Plantec (2003);**Virtual Humans: A Build-It-Yourself Kit, Complete With Software and Step-By-Step Instructions; American Management Association

[Pre02] **William H. Press (2002);** Numerical Recipes in C++: The Art of Scientific Computing; pp. 710-714

[Pro95] **Xavier Provot (1995);** Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior; Institut National de Recherche en Informatique et Automatique; Graphics Interface, 1995, pp. 147-155

[Ric81] **Paul Richer (1981);** Traité d'anatomie artistique; Bibliotheque de l'image in French 1996

[Roh04] **Andreas Rohr (2004);** Virtual Characters; Entwicklung von Lösungen und Tools für eine effiziente, anatomie-orientierte Erstellung und Animation virtueller humanoider Charaktere; FH Furtwangen; Diplomarbeit

[SHSI99] **Wei Sun, Adrian Hilton, Raymond Smith and John Illingworth (1999);** Layered Animation of Captured Data; Animation and Simulation '99 (10th Eurographics Workshop Proceedings), Milano, Italy 1999; University of Surrey

[SPCM97] **Ferdi Scheepers, Richard E. Parent, Wayne Carlson, Stephen F. May (1997);** Anatomy-based modeling of the human musculature; CSIR South Africa; SIGGRAPH 97 Conference Proceedings, January, 1997

[Sum83] **Quentin Summerfield (1983);** Analysis, Synthesis and Perception of Visible Articulatory Movements; Academic Press Inc

[TA04] **Sina Taghvakish, Jalal Amini(2004)**; Optimum Weight in Thin Plate Spline for Digital Surface model Generation; Proceedings of Intercontinental Athenaeum Athens, Greece, May 22–27, 2004

[TBHF03] **Joseph Teran, Sylvia Blemker, Victor Ng Thow Hing, and Ron Fedkiw (2003);** Finite Volume Methods for the Simulation of Skeletal Muscle; Stanford University, Honda Research Institute USA; Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation, pp 68 - 74

[TLT88] **Nadia Magnenat-Thalmann, Richard Laperrière, Daniel Thalmann (1988);** Joint-Dependant Local Deformations for Hand Animation and Object Grasping; Université de Montréal; Proceedings of Graphics Interface '88; pp. 26 - 33

[TT93] **Russell Turner, Daniel Thalmann (1993);** The Elastic Surface Model for Animated Character Construction; University of Maryland Baltimore County; Proceedings of CG International, Lausanne, Switzerland. pp 399 - 412, Springer-Verlag 1993

[Wat87] **Keith Waters (1987);** A Muscle Model for Animating Three-Dimensional Facial Expressions; National Centre for Computer Aided Art and Design Middlesex Polytechnic England; Computer Graphics (SIGGRAPH '87), p. 21, pp. 17 - 24

[Web02] **Jason Weber (2002);** Constrained Inverse Kinematics; Game Programming Gems 3, edited by Dante Treglia; Charles River Media, Inc.; pp. 192-199

[Wel93] **Chris Welman (1993);** Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation; Master thesis; Simon Fraser University

[WF95] **Keith Waters, J. Frisbie (1995);** A Coordinated Muscle Model for Speech Animation, In Proceedings Graphics Interface '95, pp. 163 - 170

[WG97] **Jane Wilhelms, Allen van Gelder (1997);** Anatomically based modelling; University of California; ACM SIGGRAPH 97 Conference Proceedings, Annual Conference Series, pp. 173 - 180

[Wil94] **Jane Wilhelms (1994);** Modeling Animals with Bones, Muscles, and Skin; University of California

[WM98] **Yin Wu, Nadia Magnenat-Thalmann (1998);** A Plastic-Visco-Elastic Model for Wrinkles in Facial Animation and Skin Aging; MIRALab, CUI, University of Geneva; Proceedings 2nd Pacific Conference on Computer Graphics and Applications, Pacific Graphics '94

[Zhu98] **Qinghong Zhu (1998);** 3D Voxel-Based Muscle Volume Deformation by Finite Element Method; Master Project Final Report; SUNY at Stony Brook

[ZST04] **Yu Zhang, Terence Sim, Chew Lim Tan (2004);** Rapid Modeling of 3D Faces for Animation Using An Efficient Adaptation Algorithm; National University of Singapore; Proc, International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (GRAPHITE 2004), June 2004, Singapore

B.2. Internet

[AltNDI] **Markus Altmann**; About Non-Uniform Rational B-Splines – NURBS; a summary; http://www.cs.wpi.edu/~matt/courses/cs563/talks/nurbs.html

[Eli03I] **Hugo Elias (2003);** The good looking textured light-sourced bouncy fun smart and stretchy page; Collection of CG tutorials;
http://freespace.virgin.net/hugo.elias/
http://freespace.virgin.net/hugo.elias/models/m_string.htm

[FacNDI] **Face Up Exercise (n.d.);** Facial Muscles;
http://www.face-up-exercise.com/facialmuscles.htm

[Gra05I] **James Grass (2005);** Human Anatomy and Physiology; College of San Francisco; Fig. 6.5; http://fog.ccsf.cc.ca.us/~jgrass/Links/musclelink.html

[Hac00I] **Hacettepe Üniversitesi Tıp Fakültesi (2000);** Course containing viscerocranium and neurocranium;
http://www.medinfo.hacettepe.edu.tr/ozetler/detay_en.asp?dersno=87
http://www.medinfo.hacettepe.edu.tr/ozetler/detay_en.asp?dersno=84

[Hop00I] **The Johns Hopkins Hospital (2000);** Skull Basics; Anatomy; Regions of the Skull;
http://www.hopkinsmedicine.org/craniofacial/LynmProject/BSC/BSC1.HTM

[Jia04I] **Zhang Jian (2004);** Implementing A Skin Deformer; Maya® API tutorial;
http://www.zjprogramming.com/html/Implementing_A_Skin_Deformer.html

[Kat03I] **Eugenii Katz(2003);** Guillaume Benjamin Amand Duchenne - Biography ; Institute of Chemistry; http://chem.ch.huji.ac.il/~eugeniik/history/duchenne.html

[May04I] **Bonny Mayes(2004);** Muscles of the Body; Austin Community College;
http://www.austin.cc.tx.us/bmayes/chap11.html

[MceNDI] **Kevin J. McElwee (n.d.);** Keratin - Introduction to skin and hair biology; http://www.keratin.com/aa/aa006.shtml

[Mil04I] **Dr. Ted Milner (2004);** Muscle-Tendon Mechanics; Muscle Fiber Geometry; http://www.sfu.ca/~tmilner/aclb3.pdf

[MSN05] **MSN Encarta (2005);** Online Encyclopaedia, Dictionary, Atlas, and Homework; http://encarta.msn.com/

[Ort96I] **Gerardo Ivar Sanchez Ortiz(1996)**; Thin-Plate Splines and Deformation Analysis; Motion and Deformation Analysis Description; http://www.doc.ic.ac.uk/~giso/pubs/leedsok/node5.html

[SchNDI] **Jason Schleifer;** Interview with Alias about Lord of the Rings – The Return of the King; http://www.alias.com/eng/community/customer_stories/weta_return_of_the_king.jhtml

[SE04I] **Judd Simantov and Mark Edwards (2004);** cgmuscle; Collaborative project for the development of an open source muscle system; http://www.cgmuscle.com

[VHP86I] **The Visible Human Project (1986);** National Library of Medicine; Three-dimensional representations of the normal male; http://www.nlm.nih.gov/research/visible/visible_human.html

[Wik01I] **Wikipedia (2001);** The Free Encyclopaedia; Collaboratively written by contributors from all around the world; http://www.wikipedia.org

[WolNDI] **Wolfram research (n.d.);** Mathworld; Extensive mathematics resource; http://mathworld.wolfram.com